A LINGUISTIC METHOD FOR ROBOT VERIFICATION, PROGRAMMING, AND CONTROL

A Thesis
Presented to
The Academic Faculty

by

Neil T. Dantam

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Robotics

Robotics and Intelligent Machines
Georgia Institute of Technology
December 2014

A LINGUISTIC METHOD FOR ROBOT VERIFICATION, PROGRAMMING, AND CONTROL

Approved by:

Professor Henrik I. Christensen,
Committee Chair
Interactive Computing
Georgia Institute of Technology

Professor Mike Stilman, Advisor
Interactive Computing
Georgia Institute of Technology

Professor Magnus Egerstedt
Electrical and Computer Engineering
Georgia Institute of Technology

Professor Irfan Essa
Interactive Computing
Georgia Institute of Technology

Professor Andrea L. Thomaz
Interactive Computing
Georgia Institute of Technology

Professor George J. Pappas
Electrical and Systems Engineering
University of Pennsylvania

Date Approved: 27 October 2014

ACKNOWLEDGEMENTS

I am grateful to my advisor Mike Stilman. Thank you, Mike, for your guidance, support, and patience. Your encouragement pushed me to do more than I realized I could, your lucid and engaging style is a standard to which I will always strive, and your perpetual enthusiasm will always be an inspiration.

Thank you to my thesis committee members: Henrik Christensen, Magnus Egerstedt, Irfan Essa, George Pappas, and Andrea Thomaz. Your helpful insight and feedback has given me better focus and improved the quality and presentation of this work.

Many thanks to my collaborators for all of the fascinating discussions and fun experiments. Thank you, Heni Ben Amor, for your substantial feedback and your help implementing the work in chapter 5. Thank you, Henrik Christensen, for always giving the right context in which to present a topic. Thank you, Irfan Essa, for helping to identify good demonstrations and steering me towards suitable perception. Thank you, Aaron Ames and Ayonga Hereid, for working with me to develop a software synthesis pipeline, and then entrusting your robot in College Station to code generated in Atlanta. Thank you, Paul Oh, Dan Lofaro, Michael Grey, and Matt Zucker, for your work showing some of my software underpinning the control implementation on a cool robot, the DRC-Hubo. Thank you, Carlos Nieto, for helping to put together a fun demo despite a flaky robot. Thank you, Magnus Egerstedt, for your help in understanding hybrid dynamics and developing some proofs in chapter 4. Thank you, Pushkar Kolhe, for implementing the perception side of the experiments that started the work in this thesis.

Thank you to all who stepped in to give their help and support in our time of loss, especially to Annie Antón, Aaron Bobick, Frank Dellaert, Henrik Christensen, Andrea Thomaz, Heni Ben Amor, and Cynthia Matuszek.

Thank you to my friends and fellow members of the Humanoids Lab. Thank you in particular to Jon Scholz for helping bootstrap our lab from day one, to Saul Reynolds-Haertle for countless crazy and entertaining discussions, and to Tobi Kunz for his prescient feedback on many papers and presentations. Thank you also to Pushkar Kolhe, Misha Novitsky, Ana Huamán, Martin Levihn, Michael Grey, Can Erdogan, and Sungmoon Joo. You have all made this lab a tremendously exciting place to work.

All the professors I have talked and studied with have shaped my understanding and views. Thank you, in particular, to Aaron Bobick for always asking the most interesting questions and to Eric Feron for showing me a different perspective on software engineering.

I am thankful to our research engineer Dan Walker for all the sturdy equipment he built to support this work.

Thank you to my mentors at Purdue University – Antony Hosking, Monika Ivantysynova, and Peter Meckl – for first introducing me to the excitement of research. Today, I could not imagine taking any other path.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY

There are many competing techniques for specifying robot policies, each having advantages in different circumstances. To unify these techniques in a single framework, we use formal language as an intermediate representation for robot behavior. This links previously disparate techniques such as temporal logics and learning from demonstration, and it links data driven approaches such as semantic mapping with formal discrete event and hybrid systems models. These formal models enable system verification – a crucial point for physical robots. We introduce a set of rewrite rules for hybrid systems and apply it automatically build a hybrid model for mobile manipulation from a semantic map.

In the manipulation domain, we develop a new workspace interpolation methods which provides direct, non-stop motion through multiple waypoints, and we introduce a filtering technique for online camera registration to avoid static calibration and handle changing camera positions. To handle concurrent communication with embedded robot hardware, we develop a new real-time interprocess communication system which offers lower latency than Linux sockets.

Finally, we consider how time constraints affect the execution of systems modeled hierarchically using context-free grammars. Based on these constraints, we modify the LL(1) parser generation algorithm to operate in real-time with bounded memory use.

CHAPTER I

INTRODUCTION

Programming robots would be easier if it were automatic. In this thesis, we develop a method to automate robot software development through the use of formal language.

General-purpose, autonomous robots offer tremendous potential across diverse areas from manufacturing to domestic service to disaster response. There are many successful teleoperated and single-task robots and even multifunctional robot hardware, yet multifunctional robot *systems* remain an elusive goal. The key challenge lies in developing robot *policies* that are general enough to cover the range of situations a robot may encounter yet specific enough to be executable.

Robot policy development has been approached from a variety of angles. Behaviors have been specified through logical descriptions, hierarchical decompositions, and demonstrations. Analyzing system models to ensure stability and correctness is a key focus of work in hybrid dynamic systems. Online execution is often a reactive control system ultimately implemented in software. Superficially, many of these approaches seem very different, with their integration presenting a time-consuming and costly challenge. However, there is a common thread we can apply to automate integration and policy development. Underlying all of these techniques for robot policies is the concept of formal language, which provides a unifying basis to connect these previously disparate approaches.

Adopting a formal language representation for robot policies bridges high-level reasoning with low-level control and enables diverse techniques to automate model construction, system verification, and code generation.

Formal language deals with sets of sequences of symbols. This enables reasoning about policies using set operations: subset, union, intersection, and difference. It also enables policy translation, from initial descriptions as logical specifications or acquired data to intermediate representations as automata and grammars and finally to executable form as synthesized software.

This thesis also applies several existing techniques. We adopt the hybrid dynamic systems view of robots as having both continuous and discrete dynamics, and connecting this with other methods for automated specification and code generation. We directly apply the discrete event systems techniques of model checking and supervisory control. To detect objects in our manipulation experiments, we use existing perception algorithms and software [31, 141, 150].

1.1 Challenges

The difficulty in precisely specifying what a robot should do has spawned a variety of largely-disconnected techniques: logical planning, behavior-based architectures, robot programming languages, temporal logics, hybrid systems, learning from demonstration, etc. Each of techniques excels in a particular domain, e.g., procedural tasks are easily described through demonstrations and safety properties are succinctly expressed in temporal logics. However, integrating many of these approaches has previously been a manual process.

Safety is important for physical robots where failures impose physical costs. Software development and verification is a significant and costly challenge – particularly for real-time and safety-critical domains. While a software developer writing business applications may produce up to 100 lines of code per hour, real-time software developers generally produce between 0.1 and 8 lines per hour [127]; given developer salary plus overhead of \$200,000 per year, this is a cost of \$10 to \$1000 per line. Software errors are the leading cause of medical device recalls [182], and for commercial airplanes – huge mechanical systems – a quarter of the development cost goes to software verification [13, 186]. Autonomous robots are especially software-intensive devices, so automating software development and verification would significantly reduce development time and cost.

A challenge in representing robot policies for complex tasks is the required memory usage. Analogous to the curse of dimensionality with increasing continuous degrees of freedom, discrete policies can succumb to combinatorial explosion as discrete options increase. This issue has been widely addressed in the development of plans – where only a single execution path is necessary – but remains a significant issue for the development of policies – where multiple execution paths and response to unpredictable events are crucial.

1.2 Approach

We propose an approach for policy robot development that covers multiple levels of the robot abstraction hierarchy (see Fig. 1) and multiple phases of system design (see Fig. 2). Converting various types of specifications to intermediate represents as grammars and automata provides an automated pipeline from specification to execution and links task-level logical descriptions with low-level control.

1.2.1 Hierarchy of Abstraction

A hierarchy of abstractions are used in robot control, Fig. 1. Robot hardware, such as servo controllers and sensors, is at the lowest level. Above that are low-level control modes for simple tasks such as trajectory tracking or specific walking gaits. These low-level controllers are referred to with a variety of names, including skills, behaviors, and motion primitives. Then, the low-level control elements are sequentially composed – forming a *language* of controllers. On top of the compositional layer is the higher level reasoning or planning to perform the desired task.

Viewing high-level reasoning as formal language directly links this abstraction level to real-time control, avoiding the need to separately implement planning and execution layers. To address the potentially large memory usage of the resulting automata, we introduce

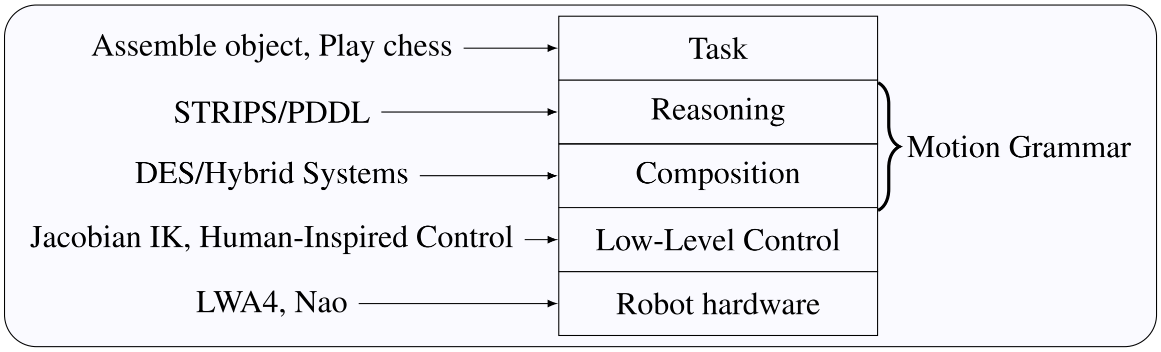


Figure 1: An abstraction hierarchy for robot control. At the lowest level is the robot hardware. Above the hardware are low-level controllers – sometimes called *behaviors*, *skills*, or *motion primitives*. These low-level controllers are sequentially composed in a language, e.g., using discrete event or hybrid systems approaches. Above this composition is high-level reasoning or planning to perform the desired task. The linguistic approach described in this thesis bridges the compositional and reasoning levels of this hierarchy.

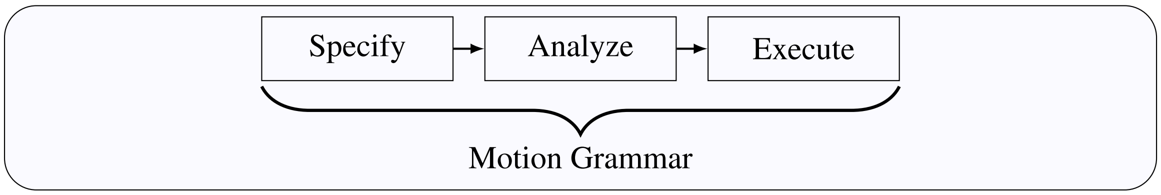


Figure 2: System design approach. The linguistic framework in this thesis integrates specification, analysis, and execution.

algorithms for direct generation of minimum state forms and hierarchical compaction of policies.

1.2.2 Phases of System Design

We consider three phases of system design: specification, analysis, and execution (Fig. 2).

1.2.2.1 Specification

In the specification phase, the system designer describes and models the system and desired behavior. For traditional dynamical systems, this amounts to producing the equations of motion and performance criteria. For robots, a variety of specification formats are commonly used: finite state machines, behavior-based architectures, STRIPS-like domains, temporal logics, task demonstrations, etc. Different formats may be better suited for different tasks or users, e.g., temporal logics are convenient for safety criteria and procedural tasks may be conveniently specified through demonstrations, particularly for non-expert users. This specification gives a precise definition for how the system should operate.

1.2.2.2 Analysis

The analysis phase is the typical focus of control system design. The system model is used to determine suitable control inputs and performance guarantees.

1.2.2.3 Execution

In the execution phase, we go from the theoretical model to the physical implementation. The abstract control system is concretely implemented, typically in software, and the system runs in real-time.

1.2.3 Assumptions

We make certain assumptions in this framework to simplify the analysis the experimental design.

1.2.3.1 Fully Observable

In our manipulation experiments, we assume the immediate state of our system is fully observable.

1.2.3.2 Non-stochastic Events

We assume that the events or terminal language symbols are not stochastic. Events are assumed to be detected with perfect certainty. This limits the way in which uncertainty may be handled so that any *possible* outcome must be handled rather than dealing with sufficiently *likely* outcomes.

1.2.3.3 Serial Tasks

We assume that the high-level task is executed sequentially and consequently adopt a serial language model. However, low-level communication and control occurs in parallel (see chapter 6).

1.2.3.4 Kinematic Manipulator Control

In our manipulation experiments, we assume kinematic control of the manipulator. In reality, all manipulators are subject to forces and torques which restrict acceleration. However, the Schunk LWA3 and LWA4 manipulators used have large gear ratios which permit high acceleration, and we do consider continuity of the computed trajectories (see chapter 5).

1.3 Overview

1.3.1 Contributions

This thesis develops an integrated pipeline for producing robot control software. We combine previously distinct specification approaches: logical planning domains, semantic mapping, and human demonstrations. We address memory usages issues with algorithms to produce compact policies. We support general physical execution with a highly-efficient real-time communication method, and execution of kinematic manipulations with a new direct-motion, multipoint interpolation approach. We connect the formal models to executable software by adapting parsing to operate in real-time.

The contributions of this thesis are summarized as follows.

Integrate specification, analysis, and execution We present a linguistic framework that combines system specification, analysis, and execution.

Data-Driven Specification We show how data-driven approaches for modeling – learning from demonstration and semantic mapping – can be connected to formal verification and code generation.

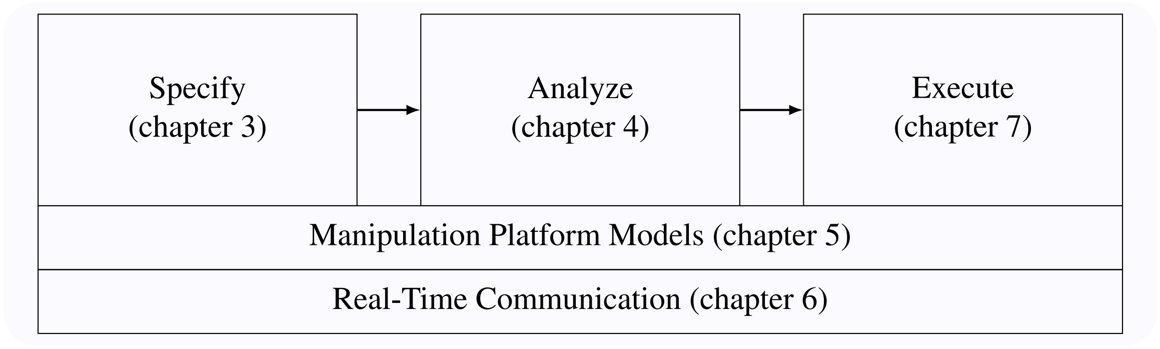

Specify
(chapter 3)
Analyze
(chapter 4)
Execute
(chapter 7)
Manipulation Platform Models (chapter 5)
Real-Time Communication (chapter 6)

Figure 3: Organization of this thesis.

Logical Domain Policies We give an algorithm to produce minimum finite-state policies directly from logical planning domains, accounting for faults and uncontrollable events

Hierarchical Policy Compaction We give an algorithm to reduce the memory usage for linguistic policies by inferring hierarchies.

Generating Real-Time Software We generate real-time control software directly from the mathematical model.

Real-Time Communication We present a high-performance, real-time, concurrent communication software library that outperforms Linux sockets

Direct, Nonstop Workspace Interpolation We develop an interpolation scheme that produces direct, constant-axis motion through multiple workspace waypoints without stopping at each waypoint.

1.3.2 Outline

This thesis is organized as follows (see Fig. 3).

chapter 2 – Related Work surveys related research in the areas of discrete event and hybrid systems, formal methods for robotics, parser generation.

chapter 3 – Specifying Language Models introduces a context-free grammar based representation for robot policies and shows how these grammars can be generated from hierarchical task decompositions, human demonstrations, and logical planning domains.

chapter 4 – Analyzing Language Models discusses the formal guarantees and analyses possible with this framework and introduces a rewrite system for hybrid models.

chapter 5 – Platform Models details the underlying kinematic control approach used for the manipulation experiments in this work, including an online approach for camera-manipulator registration.

chapter 6 – Modeling and Programming Concurrency presents a programming approach to handle the inherent concurrency in physical robots and a high-performance communication library for real-time control.

chapter 7 – Executing Language Models analyses the constraints imposed during the online execution of a linguistic model and presents an algorithm for real-time LL(1) parsing.

chapter 8 – Conclusion and Future Work summarizes the contributions of this work and discusses future extensions to relax the currently-imposed assumptions.

CHAPTER II

RELATED WORK

We survey related work in the areas of formal language, discrete event and hybrid systems, and robot policy specification, and we discuss how they relate to this thesis.

2.1 Formal Language

There is a large body of literature on grammars from the Linguistic and Computer Science communities, with a number of applications related to robotics.

Languages and grammars are widely used for perception. Fu did some early work in syntactic pattern recognition [69]. Han, et al. use attribute graph grammars to parse images of indoor scenes by describing the relationships of planes in the scene according to production rules [76]. Koutsourakis, et al. use grammars for single view reconstruction by modeling the basic shapes in architectural styles and their relations using syntactic rules [110]. Toshev, et al. use grammars to recognize buildings in 3D point clouds [177] by syntactically modeling the points as planes and volumes. The syntactic approach is applied to human activity recognition by [93, 129, 131, 140]. Our goal here is not to just recognize or classify an activity in isolation, but to combine perception and action online.

B. Stilman's Linguistic Geometry applies a syntactic approach to deliberative planning and search in adversarial games [168]. Our focus is on real-time robot control.

Parser generation is an established technique with many successes. Recursive Descent parsing was popular for early compilers [82] and has more use recently as well [34, 70]. Lewis and Stearns developed LL grammars [118]. Knuth developed LR parsing [108]. DeRemer developed LALR [55] and SLR parsing [54], with LALR methods popularized by Johnson's Yacc tool [94]. The Earley [59] and CYK [99, 187] algorithms produce parsers for any Context-Free Grammar. Compared to these parsing methods for program translation, online parsing for robot control presents some restrictions due to time constraints and the potential for very long input strings. We address these issues by developing a specially optimized LL(1) parser generator.

Grammatical inference is an ongoing field of research focused on developing language models from example strings and learner queries [51]. While there are a number of positive results in the field, trivial grammatical inference problems are often undecidable. For example, the class of regular languages cannot be learned solely from positive examples. To develop a workable system given these challenges, we initially focus in inferring grammars for finite languages. However, our overall approach is also amenable to more powerful forms of inference such as informed learning.

2.2 Discrete and Hybrid Systems

Hybrid Control is a quickly advancing research area describing systems with both discrete, event-driven, dynamics and continuous, time-driven, dynamics. Ramadge and Wonham [151] first applied Language and Automata Theory [86] to Discrete Event Systems. Hybrid Automata generally combine a Finite Automaton (FA) with differential equations associated with each FA control state. This is a widely studied and utilized model [6, 25, 81, 88, 122].

Model checking and supervisory control formally relate the behavior of a system model with a given specification. Ensuring correct operation is important for physical robots where errors may cause damage or injury. Model checking verifies correctness, and supervisory control enforces it. Model-checking has been successfully applied to software verification [84], and supervisory control approaches have also been used to ensure correct software synchronization [185]. We can also apply supervisory control to context-free grammars [86].

To model-check a hybrid system, we must know the feasibility of discrete transitions resulting from the continuous dynamics. In other words, it is important to know whether or not discrete transitions from continuous region *A* to *B* are possible. This is particularly important in the case where region *B* is a system failure state that should be avoided. The general answer to this question can be determined by solving the Hamilton-Jacobi-Isaacs partial differential equation (HJI PDE) to compute the backwards reachable set from the region of the transition [57, 130]. However, solving these HJI PDEs can be very difficult. The method of Barrier Certificates is a simpler approach that verifies avoidance of unsafe regions using a local test for uncrossable boundaries [148]. We apply this method in our approach for deriving safe system paths.

Model checkers also use the simulation and bisimulation relations between two systems, which show that one system may match the stepwise behavior of the other [13]. These relations are useful because they allow properties proven for one system to transfer to the other. Bisimulation for continuous and hybrid systems is studied in [74]. We use a simplified simulation relation in subsection 4.1.2 to determine allowable steps in the stepwise derivation of a correct system.

The Motion Description Language (MDL) is another approach that describes a hybrid system switching though a sequence of continuously-valued input functions [19, 87]. This string of controllers is a *plan* whereas we focus on *policies* representing the robot's response to any feasible event.

Harel statecharts [77] and SysML [138] are popular visual modeling representations. We consider the specification of models not only through manual decomposition but from a variety of formats including STRIPS-like domains and through data-driven approaches.

Discrete event and hybrid systems models are widely applied to robots. Grammars were used to represent robot tasks in [183]. Lyons and Arbib introduced a linguistic control model for robots [123] based on port automata. Kosecka directly applied the discrete event systems approach to mobile robots [109]. Rawal, et al. use a class of Sub-Regular Languages to describe robotic systems [153]. Maneuver Automata use a Finite Automaton to define a set of maneuvers that transition between trim trajectories [68]. [7] describes a hierarchical modeling language for hybrid systems. Temporal logics such as LTL have gained popularity in robotics [64, 106, 111, 172]. [111] uses an English-like syntax for Linear Temporal Logic in mobile robot motion planning. Controller synthesis for probabilistic environments is considered in [180]. The combination of motion planning and logical specifications is explored in [144] which checks safety properties of hybrid systems and [98, 125] which generate motion plans for hybrid systems. Livingston et. al. present a method to locally modify automata to handle changes in the environment [119]. These works demonstrate the utility of discrete event and hybrid systems methods in robotics.

The focus of this thesis is on the extending levels of abstraction (Fig. 1) and phases of system design (Fig. 2) to which these formal methods apply. Rather than using formal language only to reactively compose control modes, we consider efficiently integrating high-level planning traditionally viewed as a distinct, deliberative step. Rather than operating on manually-specified models and producing abstract controllers, we consider techniques to partially-automate specification and synthesis and execution of real-time software from formal models. Using formal language as an intermediate representation, we can encode a variety of specification formats – including specifications derived from data or typically reserved for deliberative planning, – analyze the resulting formal model, and generate and execute real-time software.

2.3 Logical Planning

Logical planning was pioneered with STRIPS [65] and has been studied in detail [18, 80, 83, 100]. The general planning problem operates on a description of the domain, an initial state, and a goal state to produce an execution path from the initial state to the goal.

Different aspects of the relationship between planing, temporal logics, and language have also been explored. In this work, we focus on the direct representation of planning domains as formal language, introducing methods for finding compact representations of planning domain languages. SAT-solvers have proven effective for both planning [100] and model checking [33]. Generating plans with loops is analyzed in [165]. Compared to this approach, we propose a single formal language which combines both the planning and execution, which may include loops. [24] considers performing planning within a discrete event system simulation. In contrast, we consider the language-theoretic properties of planning and show that simple planning domains can be expressed entirely within a discrete event systems and without external planners invoked at runtime. [5] describes a temporal logic for planning problems. In this work, we focus on the connections between logical domains and formal language to construct policies. [50] considers use of infinite-string Büchi automata for solving planning problems, and [38] shows a translation from Linear Temporal Logic to the Planning Domain Description Language (PDDL). However, Büchi automata minimization is NP-complete [157], and minimal forms are not canonical [166]. We instead use finite-string regular automata. We can find canonical minimum state representations of deterministic finite automata in $O(n \log n)$ [85], and we use hierarchical automata to further reduce representational size. Identifying the canonical forms of subtask finite automata is key to inducing the task hierarchy.

Hierarchical Task Network (HTN) planning is closely related to the context-free presentation we apply. Generalized HTN planning can simulate context-free grammars [62] if tasks may self-recurse. We consider the opposite case, using automata and grammars to represent a policy. In practical applications of HTN, it is typically finite-state equivalent [117, 124].

Automatic inference of hieararchies is also explored with the Alpine [107] and Highpoint [12] algorithms. In contrast to this work, we take an automata-based view in order find hierarchical abstractions while also handling alternative outcomes and faults.

In the assembly domain, Tellex, Knepper, et. al. use a STRIPS-style planner to sequence assembly actions, and when failure occurs, consider how to generate useful requests for human help [174]. In contrast, we focus on how the robot can recover from faults on its own.

2.4 Policy Specification Approaches

Domain specific *robot programming languages* are widely used. An early approach which remains in common use is the G-code of CNC machines [61]. The Forth programming language was originally developed to control radio telescopes [152]. [121] presents a domain-specific language for robot manipulation tasks. [104] describes a type-safe, Turing-complete robot programming language. Other approaches develop robot programming frameworks in general programming languages, such as OpenRTM [10], Orocos [22], and ROS [149]. The goal of robot programming languages and frameworks is generally to provide a convenient software environment to specify robot behavior. In contrast, we are concerned with formally modeling and verifying robot behavior, whiling maintaining the convenience of automatically generating executable software.

There are numerous other approaches to learning from demonstration for robotic systems [11, 17]. Many approaches focus on learning continuous trajectories [156], while in this work, we focus on a symbolic abstraction of a specific task. Other symbolic learning approaches include [27] which learns goal configurations for sets of objects and [60] which learns a logical model for a STRIPS planner from multiple human demonstrations. Our work differs from these other methods by producing a syntactic task model which, combined with the semantics for a robot, represents a hybrid dynamical control policy that is formally verifiable and efficiently executable.

CHAPTER III

SPECIFYING LANGUAGE MODELS

Formal language provides a unifying basis for a variety of specification approaches. First, we introduce a linguistic model, the Motion Grammar, based on context-free grammars, and give two examples of hierarchically specified applications. Then, we consider alternate specification approaches based on human demonstrations and logical planning domains.

3.1 The Motion Grammar

The Motion Grammar (MG) is a Syntax-Directed Definition expressing the language of interaction between agents and real-world uncertain environments. MG tokens are system states or discretized sensor readings. MG strings are histories of these states and readings over the system execution. Like SDDs for programming languages, the MG must have two components: *syntax* and *semantics*. The syntax represents the ordering in which system events and states may occur. The semantics defines the response to those events. The MG uses its syntax to decide from the set of system behavior and semantics to interpret the state and select continuous control decisions.

The Motion Grammar represents the operation of a robotic system as a Context-Free

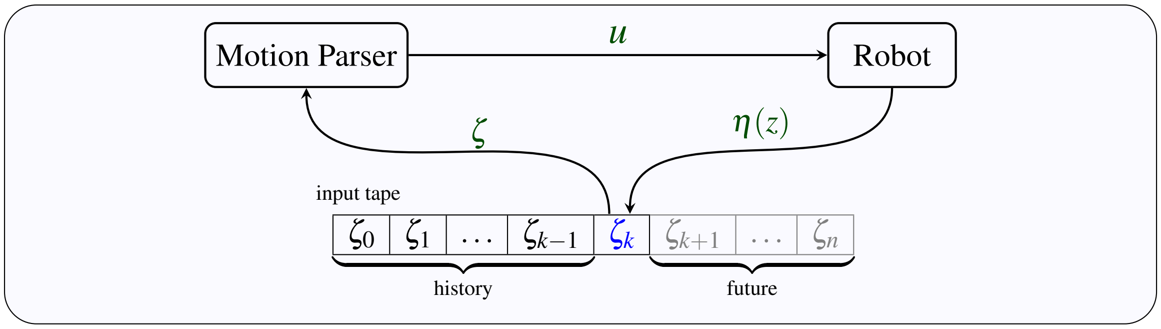


Figure 4: Operation of the Motion Grammar.

language. The grammar is used to generate the *Motion Parser* which drives the robot as shown in Fig. 4.

Definition 1 (Motion Grammar)**.** *The tuple* $\mathcal{G}_M = (Z, V, P, S, \mathcal{X}, \mathcal{Z}, \mathcal{U}, \eta, K)$ *where,*

Z	*set of events, or tokens*
V	*set of nonterminals*
$P \subset V \times (Z \cup V \cup K)^*$	*set of productions*
$S \in V$	*start symbol*
$\mathcal{X} \subseteq \mathbb{R}^m$	*continuous state space*
$\mathcal{Z} \subseteq \mathbb{R}^n$	*continuous observation space*
$\mathcal{U} \subseteq \mathbb{R}^p$	*continuous input space*
$\eta : \mathcal{Z} \times P \times \mathbb{N} \mapsto Z$	*tokenizing function*
$K \subset \mathcal{X} \times \mathcal{U} \times \mathcal{Z} \mapsto \mathcal{X} \times \mathcal{U} \times \mathcal{Z}$	*set of semantic rules*

Definition 2 (Motion Parser)**.** *The Motion Parser is a program that recognizes the language specified by the Motion Grammar and executes the corresponding semantic rules for each production. It is the control program for the robot.*

From Def. 1, the Motion Grammar is a CFG augmented with additional variables to handle the continuous dynamics. Variables Z, V, P, and S are the CFG component. Spaces $\mathcal{X}$, $\mathcal{Z}$, and $\mathcal{U}$ are for the continuous state, measurement, and input. The tokenizing function η produces the next input symbol for the parser according to the sensor reading and the position within the currently active production. The semantic rules K describe the continuous dynamics of the system and are contained with the productions P of the CFG. Using these discrete and continuous elements, the combined Motion Grammar $\mathcal{G}_M$ explicitly defines the *Hybrid System Path*.

Definition 3 (Hybrid System Path)**.** *The* path *of a system defined by Motion Grammar $\mathscr{G}_M$ is the tuple $\Psi = (x, \sigma)$ where,*

$x : t \mapsto \mathscr{X}$ *continuous trajectory through $\mathscr{X}$*

$\sigma \in \mathfrak{L}\{\mathscr{G}_M\}$ *discrete string over Z*

3.1.1 Application of the Motion Grammar

We use the Motion Grammar in as a model for both *offline reasoning*, in system specification and analysis, and for *online parsing* during system execution. The properties of Context-Free languages provide guarantees for each of these phases. Offline, we can always verify correctness of the language (subsection 4.1.2) and there are numerous algorithms [4, 59, 142, 142] for automatically transforming the grammar into a parser for online control. Online, the parser controls the robot. The structure of CFLs guarantees that online parsing is $O(n^3)$ in the length of the string [59], and with some restrictions on the grammar [4, p.222], parsing is $O(n)$ – constant at each time step, a useful property for real-time control.

Online parsing is illustrated in Fig. 4. The output of the robot z is discretized into a stream of tokens ζ for the parser to read. The history of tokens is represented in the parser's internal state, i.e. the stack and control state of a PDA. Based on this internal state and the next token seen, the parser decides upon a control action u to send to the robot. The token type ζ is used to pick the correct production to expand at that particular step, and the semantic rule for that production uses the continuous value z to generate the input u. Thus, the Motion Grammar represents the language of robot sensor readings and translates this into the language of controllers or actuator inputs.

3.1.2 Languages, Systems, and Specifications

The Motion Grammar models and controls a robotic *system*. Often during controller design, there is a rigid distinction between what is the plant and what is the controller, and analogously, Fig. 4 shows the Robot and the Motion Parser as separate blocks. However, these are arbitrary distinctions. Consider the case of feedback linearization where we introduce some additional computed dynamics so that we can apply a linear controller. While these additional dynamics may physically exist as software on a CPU, for the purpose of designing the linear controller, they are part of the plant. With the Motion Grammar, we have the same freedom to designate components between the plant and controller in whatever way is most convenient to the design of the overall system.

For linguistic control approaches, there is one critical distinction to make between the language of the *system* and the language for the *model*. The system is the physical entity with which we are concerned: the controller and the robot. The model is the description of how the controller and robot respond; it is a set of mathematical symbols on paper or in a computer program. Both the system and the model can be described by formal languages.

Definition 4. *The* System Language*, $\mathfrak{L}_g$, is the set of strings generated by the robot and parsed by the controller during operation.*

Definition 5. *The* Modeling Language*, $\mathfrak{L}_s$, is the set of strings that describe the operation of controllers and robots.*

These languages are related. Each string in the modeling language describes a particular system: a robot and controller. This specification is parsed offline to generate the control program. The system language is parsed online *by the control program*. The Motion Grammar is a modeling language that describes a Context-Free *system*.

We emphasize that the Motion Grammar is not simply a Domain Specific Language or Robot Programming Language [37, p.339] but rather the direct application of linguistic theory to robot control in order to formally verify performance. The language described by the Motion Grammar is that of the robotic system itself.

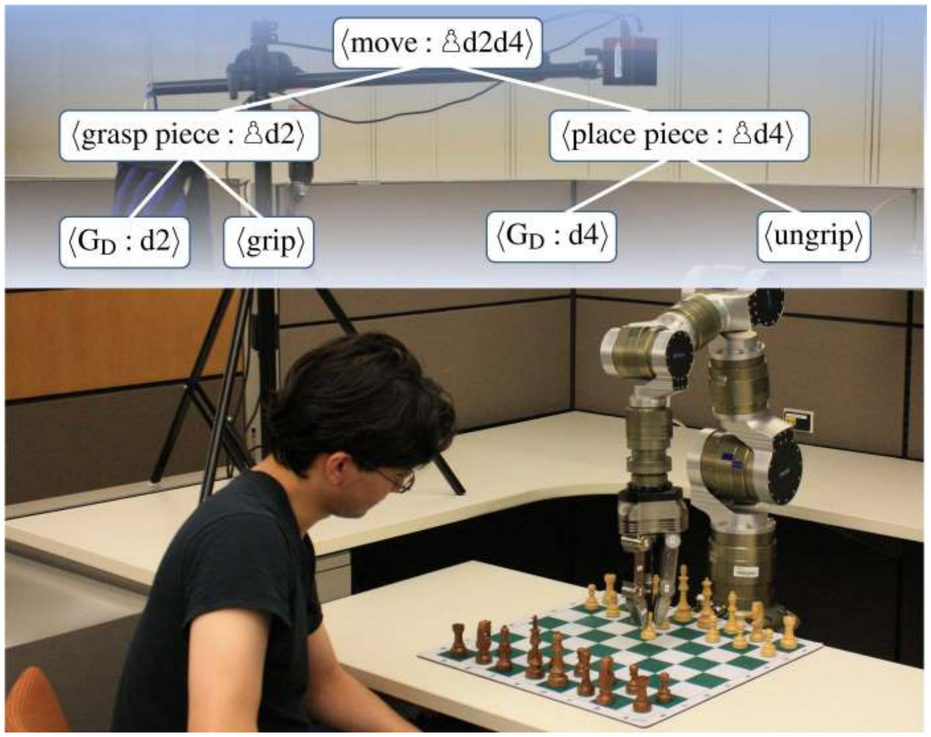


Figure 5: Our experimental setup for human-robot chess and a partial parse-tree indicating the robot's plan to perform a chess move.

3.2 Hierarchical Task Specification: Physical Human-Robot Games

The Motion Grammar is a useful model for controlling physical robots. In this section, we discuss how to apply grammars to robots and illustrate the points with our sample applications of Yamakuzushi and human-robot chess.

We performed these experiments using a Schunk LWA3 7-DOF robot arm with a Schunk SDH 7-DOF, 3-fingered hand as shown in Fig. 5. A wrist mounted 6-axis force-torque sensor and finger-tip pressure distribution sensors provided force control feedback. The robot manipulated pieces in standard yamakuzushi and chess sets, and a Mesa SwissRanger 4000 mounted overhead allowed it to locate the individual pieces. We used a Kalman filter on the force-torque sensor and both median and Kalman filters on the Swiss Ranger to handle sensor uncertainty. The robot used a speaker and text-to-speech program to communicate with its human opponent. Domain-specific planning of chess moves was done with the Crafty chess engine [91]. The perception, motion planning, and control software was implemented primarily in C/C++ and Common Lisp using the Ach (chapter 6 message-passing IPC running on Ubuntu Linux 10.04. The lowest-levels of our grammatical controller operate at a 1 kHz rate.

3.2.1 Tokenizing

The tokens in the Motion Grammar for are based on both the sensor readings and game state. A summary of token types for Yamakuzushi and Chess are given in and Table 1 Table 3. Position thresholds, velocity thresholds, and timeouts indicate when the robot has reached the end of a trajectory. Force thresholds and position thresholds indicate when the robot is in a safe operating range.

While formal language defines tokens as atomic symbols, these tokens are in fact abstractions of underlying phenomena. Consider the tokens of natural language: words may exist as vibrations in air, ink on paper, or magnetic transitions on a metal disk, yet all these representations define the same symbol. In formal grammars, this hierarchy is made explicit through the relationship between nonterminal and terminal symbols. Terminal symbols are atomic. Nonterminals represent a set of strings of symbols, in essence a language of their own. Whenever it is necessary to deepen the abstraction for some terminal symbol, α, we can convert α to a nonterminal and define a new set of strings that α may expand to. We have used this approach for the manual construction of MG since it facilitates hierarchical task decomposition. For automatic grammar generation, we can again use this hierarchy of symbols to translate the task-appropriate symbols from humans to robots even though human and robot actions are quite different at the atomic level.

3.2.2 Parsing

Once the Motion Grammar for the task is developed, it must be transformed into the Motion Parser. For our Yamakuzushi and chess applications, we used a hand-written recursive descent parser, an approach also employed by GCC [70]. A recursive descent parser is written as a set of mutually-recursive procedures, one for each nonterminal in the grammar, algorithm 1. Each procedure will fully expand its nonterminals via a top-down, left-to-right derivation. This approach is a good match for the Motion Grammar's top-down task decomposition and its left-to-right temporal progression.

Algorithm 1: parse-recursive-descent-A

```
1 Choose a production for A, $A \to X_1 \dots X_n$;
2 for $i = 1 \dots n$ do
3     if nonterminal? $X_i$ then
4         call $X_i$;
5     else if $X_i = \eta\,(z\,(t))$ then
6         continue;
7     else
8         syntax error
9 Execute semantic rule for $A \to X_1 \dots X_n$;
```

3.2.3 Syntax and Semantics

The syntax of the Motion Grammar represents the discrete system dynamics while the *semantic rules* in the grammar compute the continuous dynamics and control inputs. Within the Motion Parser, semantic rules are procedures that are executed when the parser expands a production. For our application, these rules store updated sensor readings, determine new targets for the controller, and send control inputs. These values are stored in the attributes of tokens and nonterminals. Attributes for a nonterminal node in the parse tree are *synthesized* from child nodes and *inherited* from both the parent nodes and the left-siblings of that nonterminal. Here, we give a key example of robot control through semantic rules.

The Syntax-Directed Definition presented in Fig. 6 illustrates a simple grammar for implementing trapezoidal velocity profiles. Expanding $\langle \mathrm{A_i} \rangle$ will carry the system through the phases of the trajectory. While $[0 \leq \mathrm{t} < \mathrm{t_1}]$, the system will constantly accelerate according to $\langle \mathrm{A_1} \rangle$. While $[\mathrm{t_1} \leq \mathrm{t} < \mathrm{t_2}]$, the system will move with constant velocity according to $\langle \mathrm{A_2} \rangle$. While $[\mathrm{t_2} \leq \mathrm{t} < \mathrm{t_3}]$, the system will constantly decelerate according to $\langle \mathrm{A_3} \rangle$. Finally,

PRODUCTION	SEMANTIC RULES
$\langle \mathrm{T} \rangle \rightarrow \langle \mathrm{T}_1 \rangle \langle \mathrm{T}_2 \rangle$	
$\langle \mathrm{T}_1 \rangle \rightarrow \langle \mathrm{A}_1 \rangle \langle \mathrm{A}_2 \rangle$	
$\langle \mathrm{T}_2 \rangle \rightarrow \langle \mathrm{A}_3 \rangle \langle \mathrm{A}_4 \rangle$	
$\langle \mathrm{A}_1 \rangle \rightarrow [0 \leq \mathrm{t} < \mathrm{t}_1]$	$x_r = x_0 + \frac{1}{2}\ddot{x}_m t^2, \dot{x}_r = t\ddot{x}_m$
$\langle \mathrm{A}_2 \rangle \rightarrow [\mathrm{t}_1 \leq \mathrm{t} < \mathrm{t}_2]$	$x_r = x_0 + \frac{1}{2}\ddot{x}_m t_1^2 + \dot{x}_m(t - t_1), \dot{x}_r = \dot{x}_m$
$\langle \mathrm{A}_3 \rangle \rightarrow [\mathrm{t}_2 \leq \mathrm{t} < \mathrm{t}_3]$	$x_r = x_n - \frac{1}{2}\ddot{x}_m(t_3 - t)^2, \dot{x}_r = \dot{x}_m + \ddot{x}_m(t_2 - t)$
$\langle \mathrm{A}_4 \rangle \rightarrow [\mathrm{t}_3 \leq \mathrm{t}]$	$u = 0$

Figure 6: Syntax-Directed Definition that encodes impedance control over trapezoidal velocity profiles. For each A_i, the input is computed according to $u = \dot{x}_r - K_p(x - x_r) - K_f(f - f_r)$.

the system will stop according to $\langle \mathrm{A}_4 \rangle$. Each segment of the piecewise smooth trajectory is given by the semantic rule of one of the productions. This is an example of how the continuous domain control of physical systems can be encoded in the semantics of a discrete grammar.

3.2.4 Yamakuzushi

We implemented and evaluated the performance of the Motion Grammar on the Japanese game Yamakuzushi (yama). This game is similar to Jenga. In yama, a mountain of Shogi pieces is randomly piled in the middle of a table as shown in Fig. 7. Each of the two players tries to clear the pieces from the table. Each player is only allowed to use one finger to move pieces. If a player causes the pieces to make a sound, it becomes the other players turn. The winner is the player who removes the most pieces.

3.2.4.1 Touching Pieces

The Finite State Machine in 8(a) could be used to make the robot touch a Shogi piece. This state machine is equivalent to the grammar in 8(b). In the grammar, the tokenizing function η applies a threshold to the force-torque sensors and produces [contact] if the end-effector forces exceed the threshold or [no contact] otherwise. To expand the $\langle$touch$\rangle$ nonterminal, the parser consumes a [contact] and returns, or it consumes a [no contact], moves down a small increment using the trapezoidal velocity profile in $\langle$touch$'\rangle$ and $\langle$g$\rangle$, and recurses on

Table 1: Yamakuzushi Tokens

Token	Description
$[\alpha \leq \mathrm{t} < \beta]$	Within time Range
[contact]	E.E. touching piece
[no contact]	not touching piece
[destination]	at traj. end
[human piece]	removed by human
[robot piece]	removed by robot
[clear]	board cleared
[sound]	noise removing piece
[quiet]	no noise made
[in space]	human in workspace
[¬in space]	not in workspace
[point]	element of point cloud

Table 2: Attributes for the Yamakuzushi Motion Grammar

Attr.	Description
Sensor Driven	
t	Current Time
x	Act. Robot/Point Pos.
f	Act. E.E. Force
Inherited/Synthesized	
t_α	Duration or Timeout
x_r	Ref. Robot Pos.
$\dot{x}_r$	Ref. Robot Vel.
x_0	Traj. Start Pos.
x_n	Traj. End Pos.
f_r	Ref. E.E. Force

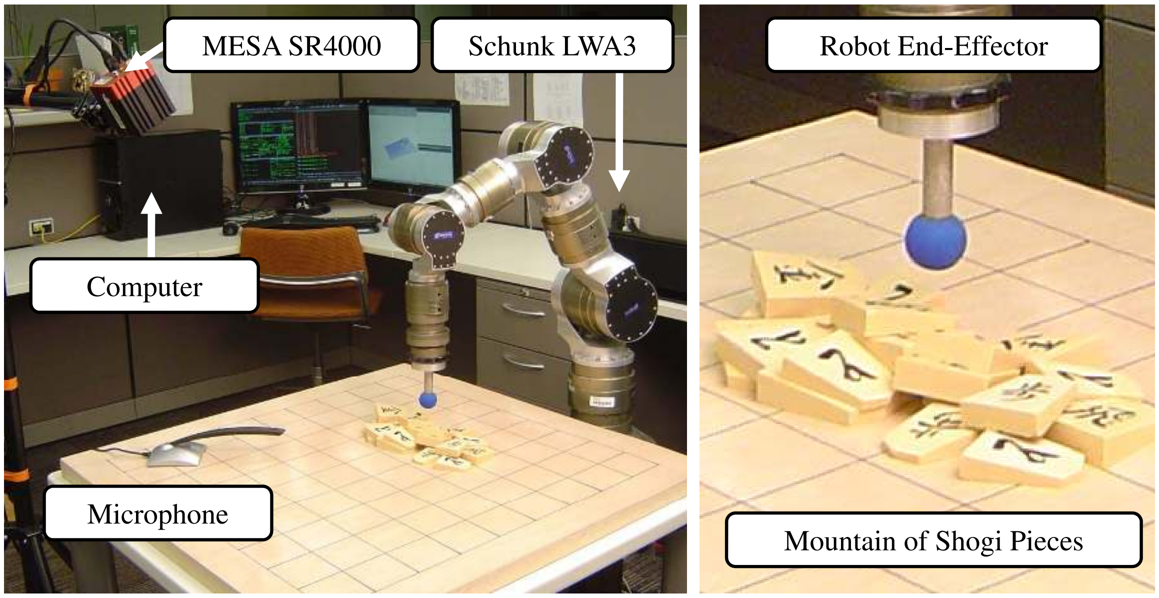
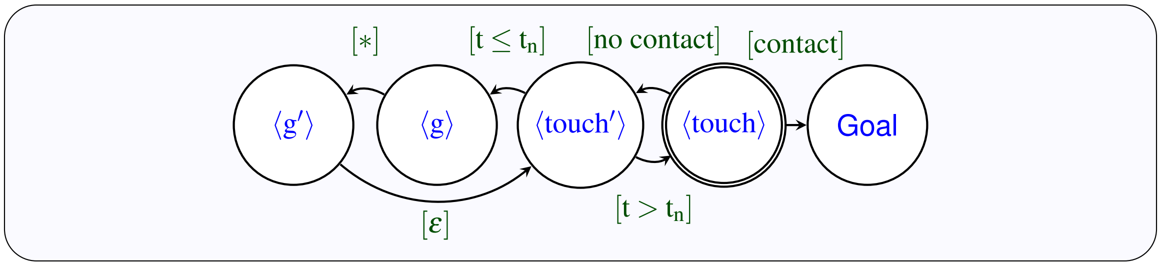


Figure 7: Our experimental setup for physical human-robot games of Yamakuzushi.

$\langle$touch$\rangle$. This behavior is mirrored by the state transitions in 8(a).

We implemented this grammatical controller for touching Shogi pieces on the LWA3 and compared it to a pure continuous-domain impedance controller. Due to the large physical constants of the LWA3, we implemented our impedance controller on top of a velocity controller. This approach has the potential for oscillation, especially when gains are large, yet even under these circumstances, the grammatical controller achieved superior performance. The impedance controller in 9(a) with an appropriate gain is able to make contact with the piece, but it does suffer from some oscillation and overshoot. An impedance controller with high gains in 9(b) has severe oscillation and very poor performance. The grammatical controller in 9(c) has both less overshoot and less oscillation than the purely continuous impedance controller. Additionally, we also observed the grammatical controller to be much more robust to sensing errors. If we estimated the height of a piece incorrectly, the impedance controller would often completely fail to make contact due to limited ability to increase gains; however, the grammatical controller would still be able to find the piece.



(a) Finite State Machine Representation

$$\begin{array}{rcl}
\langle \text{touch} \rangle & \rightarrow & [\text{contact}] \\
 & | & [\text{no contact}]\, \langle \text{touch}' \rangle \langle \text{touch} \rangle \\
\langle \text{touch}' \rangle & \rightarrow & [\text{t} > \text{t}_\text{n}] \mid [\text{t} \leq \text{t}_\text{n}]\, \langle \text{g} \rangle \langle \text{touch}' \rangle
\end{array}$$

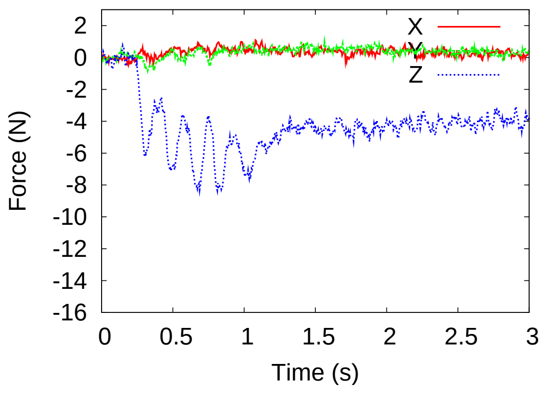
(b) Equivalent MG Fragment

Figure 8: Illustration of control for piece touching.

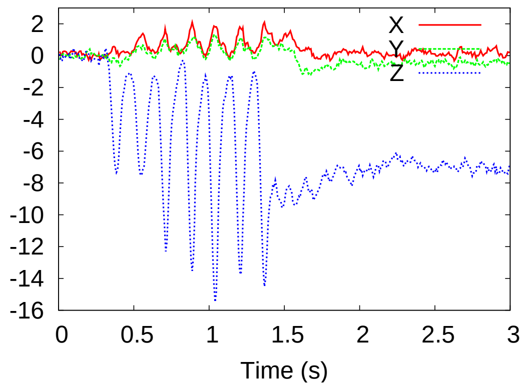
3.2.4.2 Sliding and Reacquiring Lost Pieces

The grammar in Fig. 10 describes how the robot slides pieces and how it can reacquire pieces it has lost. This grammar again uses the trapezoidal velocity profile ⟨g⟩. The tokenizer η supplies [destination] when robot has moved the piece to the desired location. If the robot momentarily loses contact with the piece, it will continue expanding ⟨slide⟩; however, when the contact loss is long enough for the robot to move past the piece, the robot must backtrack to the last contact position to reacquire the piece. This action is performed by ⟨reacquire⟩. Following this grammar allows the robot to move pieces across the table and recover any pieces that it loses.

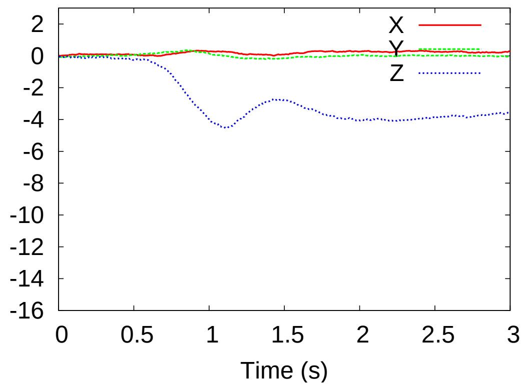
Our implementation of this grammar on the LWA3 slides pieces and recovers them after any contact loss. As the robot moves through the sequence in Fig. 11, it uses the end-effector forces shown in 11(d) to make its decisions regarding piece contact. At 6 s, the robot begins moving down to touch the piece. It acquires the pieces at 6.8 s, 11(a) and begins sliding. At 10.8 s, 11(b), it loses contact with the piece. Recognizing this, the robot backtracks, and again makes contact with the piece at 18.5 s. It then continues sliding the piece, reaching the destination at 28.3 s, 11(c).



(a) Impedance



(b) High Gain Impedance

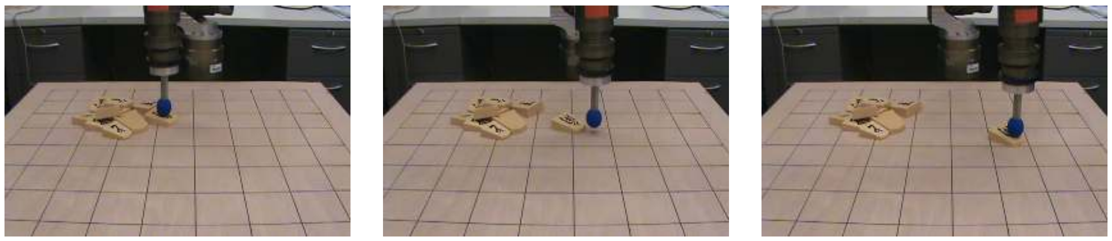


(c) Grammatical

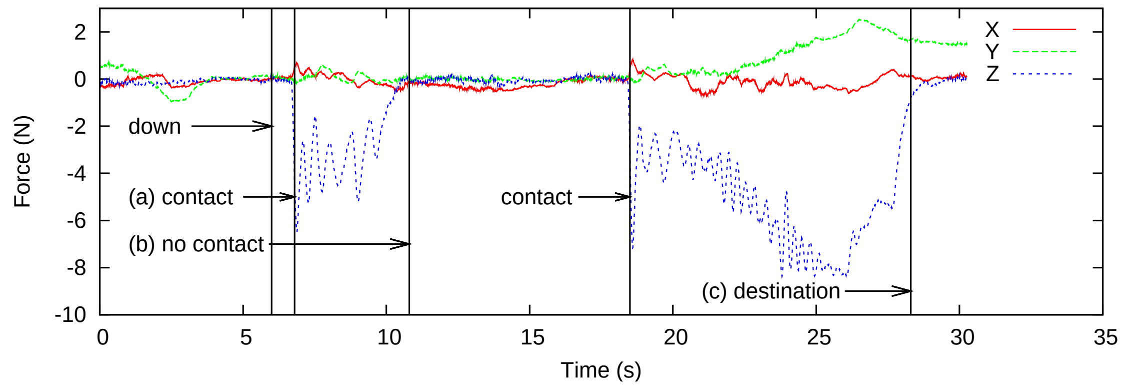
Figure 9: Yamakuzushi Piece Touching with Impedance and Discrete Control Strategies

$$
\begin{array}{rcl}
\langle\text{slide}\rangle & \rightarrow & [\text{contact}]\ \langle\text{g}\rangle\langle\text{slide}\rangle \\
 & | & [\text{no contact}]\ \langle\text{g}\rangle\langle\text{slide}\rangle \\
 & | & [\text{destination}] \\
 & | & \langle\text{reacquire}\rangle\ [\text{contact}]\ \langle\text{g}\rangle\langle\text{slide}\rangle \\
\langle\text{reacquire}\rangle & \rightarrow & [\text{no contact}]\ \langle\text{touch}\rangle
\end{array}
$$

Figure 10: Grammar fragment to reacquire lost Yamakuzushi pieces.



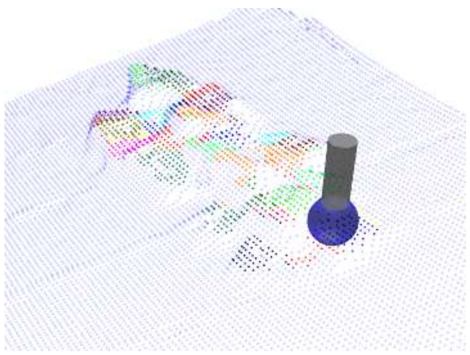
(a) contact (b) no contact (c) destination



(d) Forces at the robot end-effector during grammar execution.

Figure 11: Application of sliding grammar in Fig. 10 when contact is lost.

(a) Pieces


(b) Planes

$$
\begin{array}{rcl}
\langle\text{act}\rangle & \rightarrow & \langle\text{target}\rangle\langle\text{touch}\rangle\langle\text{slide}\rangle \\
\langle\text{target}\rangle & \rightarrow & \langle\text{plane}\rangle_0 \mid \ldots \mid \langle\text{plane}\rangle_\text{n} \\
\langle\text{plane}_i\rangle & \rightarrow & [\text{point}]_j \mid [\text{point}]_j \langle\text{plane}_i\rangle
\end{array}
$$

(c) MG Fragment

Figure 12: Deciding target Yamakuzushi piece and direction.

3.2.4.3 Selecting Target Pieces

The grammar in 3.2.4.3 describes how the robot chooses a target piece to move. The Swiss Ranger provides a point cloud from the stack of pieces, 12(a). From this point cloud, a set of planes is progressively built based on the distance between the test point and the plane and on the angle between the plane normal and the normal of a plane in the region of the test point. Using only these identified planes, the robot selects a target piece. The precedence of the target plane is based on height above the ground, a clear path to the edge of the table, and whether the piece may be supporting stacked neighboring pieces. The parser will select the highest precedence plane as the target to move, 12(b).

3.2.4.4 Deciding the Winner

An example of a Context-Free system language is deciding the winner of the game. The grammar fragment for this task is shown in Fig. 13. This grammar will count the number of pieces removed by the human [human piece] and the robot [robot piece]. The ⟨draw⟩ nonterminal serves to match up a piece removed by the human and a piece removed by the

$$
\begin{array}{rcl}
\langle\text{winner}\rangle & \rightarrow & \langle\text{draw}\rangle \mid \langle\text{robot}\rangle \mid \langle\text{human}\rangle \\
\langle\text{draw}\rangle & \rightarrow & \varepsilon \\
 & \mid & [\text{robot piece}]\,\langle\text{draw}\rangle\,[\text{human piece}]\,\langle\text{draw}\rangle \\
 & \mid & [\text{human piece}]\,\langle\text{draw}\rangle\,[\text{robot piece}]\,\langle\text{draw}\rangle \\
\langle\text{robot}\rangle & \rightarrow & \langle\text{draw}\rangle\,[\text{robot piece}]\,\langle\text{draw}\rangle \\
 & \mid & \langle\text{draw}\rangle\,[\text{robot piece}]\,\langle\text{robot}\rangle \\
\langle\text{human}\rangle & \rightarrow & \langle\text{draw}\rangle\,[\text{human piece}]\,\langle\text{draw}\rangle \\
 & \mid & \langle\text{draw}\rangle\,[\text{human piece}]\,\langle\text{human}\rangle
\end{array}
$$

Figure 13: Grammar fragment to decide winner

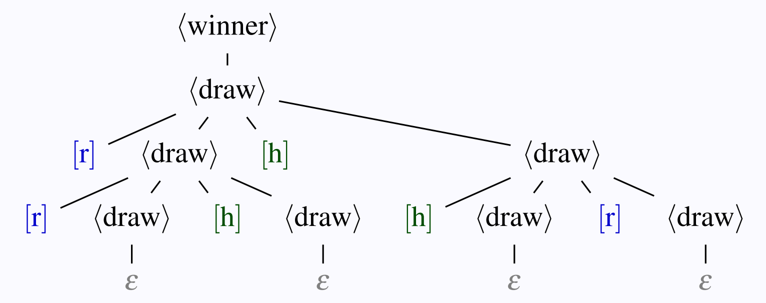


Figure 14: Parse tree for winner decision problem in draw case:
[h] ≡ [human piece], [r] ≡ [robot piece]

$$
\begin{array}{rcl}
\langle \text{game} \rangle & \rightarrow & \langle \text{robot turn} \rangle \, [\text{clear}] \, \langle \text{winner} \rangle \\
 & | & \langle \text{robot turn} \rangle \langle \text{human turn} \rangle \, [\text{clear}] \, \langle \text{winner} \rangle \\
 & | & \langle \text{robot turn} \rangle \langle \text{human turn} \rangle \langle \text{game} \rangle \\
\langle \text{robot turn} \rangle & \rightarrow & \langle \text{act} \rangle \, [\text{quiet}] \, \langle \text{robot turn} \rangle \\
 & | & \langle \text{act} \rangle \, [\text{sound}] \\
 & | & [\text{clear}] \\
\langle \text{human turn} \rangle & \rightarrow & \langle \text{wait sound} \rangle \langle \text{wait safe} \rangle \\
\langle \text{wait sound} \rangle & \rightarrow & [\text{sound}] \\
 & | & [\text{clear}] \\
 & | & [\text{quiet}] \, \langle \text{wait sound} \rangle \\
\langle \text{wait safe} \rangle & \rightarrow & [\text{in space}] \, \langle \text{wait safe} \rangle \\
 & | & [\neg \text{in space}]
\end{array}
$$

Figure 15: Complete Yama Grammar. This is the remaining set of productions used in the game.

robot. The ⟨robot⟩ and ⟨human⟩ nonterminals consume the extra tokens for pieces removed by the robot or the human, indicating that player is the winner. An example parse tree for a draw condition is given in Fig. 14. This parse tree demonstrates how each ⟨draw⟩ matches one [r] and one [h] token which requires a CFL [86, p125]. This solution to the counting problem for deciding the winner demonstrates the advantage of using a Context-Free model for the MG.

3.2.4.5 Complete Game

The remaining productions to implement a full game of Yamakuzushi are given in Fig. 15. A game consists of alternating robot and human turns until the board is clear. During the ⟨robot turn⟩, it will repeatedly ⟨act⟩ to remove pieces until it causes [sound] by making a noise exceeding the preset threshold or until it clears the board. During the ⟨human turn⟩, the robot will simply wait until it detects a [sound] or sees that the board has been cleared. After the human makes a [sound], the robot will wait until the human is out of the workspace before beginning its turn.

Table 3: Chess Grammar Tokens

Sensor Tokens		
Token	$\eta(z)$	**Description**
[0]	$t < t_1 \vee \|\|\mathbf{x} - \mathbf{x}_1\|\| > \varepsilon_x \vee \|\|\dot{\mathbf{q}}\|\| > \varepsilon_{\dot{q}}$	Not at Traj. End
[1]	$\neg$ [0]	At Traj. End
[limit]	$\|\|\mathbf{F}\|\| > F_{\max}$	Force Limit
[grasped]	$\int \rho dA > \varepsilon_{\int \rho}$	Pressure sum limit
[ungrasped]	$\neg$ [grasped]	Pressure sum limit

Chessboard Tokens	
Token	**Description**
[set]	board is properly set
[moved]	opponent has completed move
[checkmate]	checkmate on board
[resign]	a player has resigned
[draw]	players have agreed to draw
[cycle(x)]	x is in a cycle of visited during

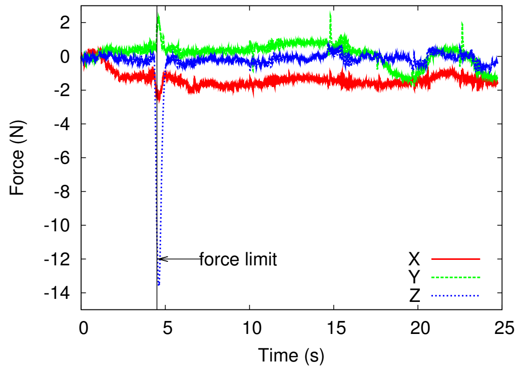
Perception Tokens		
Token	$\eta(z)$	**Description**
[obstacle]	$w(\mathbf{C}) < w_k$	Robot workspace occupied
[occupied(x)]	$w(x) > w_m in$	Piece is present in x
[clear(x)]	$\neg$ [occupied(x)]	No piece in x
[fallen(x)]	$\text{height}(x) < h_{\min}$	Piece is fallen
[offset(x)]	$\text{mean}(x) - \text{pos}(x) > \varepsilon$	Piece is not centered
[moved]	$C_r \neq C_c$	Boardstate is different
[misplaced(x)]	$C_r(x) \neq C_c(x)$	Piece is missing

3.2.5 Chess

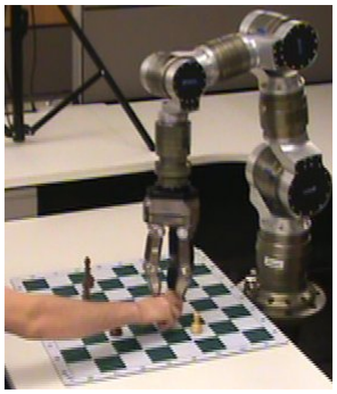
Next, we demonstrate a Motion Grammar to perform the real-time motion control in human-robot chess.

3.2.5.1 Guarded Moves

Our implementation of guarded moves using the Motion Grammar allows the human and robot to safely operate in the same workspace. A [limit] token is generated when the wrist force-torque sensor encounters forces above a preset limit. The limit is large enough so that



(a) Forces



(b) Contact

Figure 16: Grammatical guarded moves safely protecting the human player.

$$
\begin{aligned}
\langle \mathrm{G} \rangle &\to \langle \mathrm{G_D} \rangle \mid \langle \mathrm{G_L} \rangle & (1)\\
\langle \mathrm{G_D} \rangle &\to [1] \mid \langle \kappa \rangle \langle \mathrm{G_D} \rangle & (2)\\
\langle \mathrm{G_L} \rangle &\to [\mathrm{limit}] \mid \langle \kappa \rangle \langle \mathrm{G_L} \rangle & (3)\\
\langle \kappa \rangle &\to [0]\ \left\{ \dot{\mathbf{q}} = \mathbf{J}^* \left(\dot{\mathbf{x}} - \mathbf{K}_p \left(\mathbf{x} - \mathbf{x}_r \right) - \mathbf{K}_f \left(\mathbf{F} - \mathbf{F}_r \right) \right) \right\} & (4)
\end{aligned}
$$

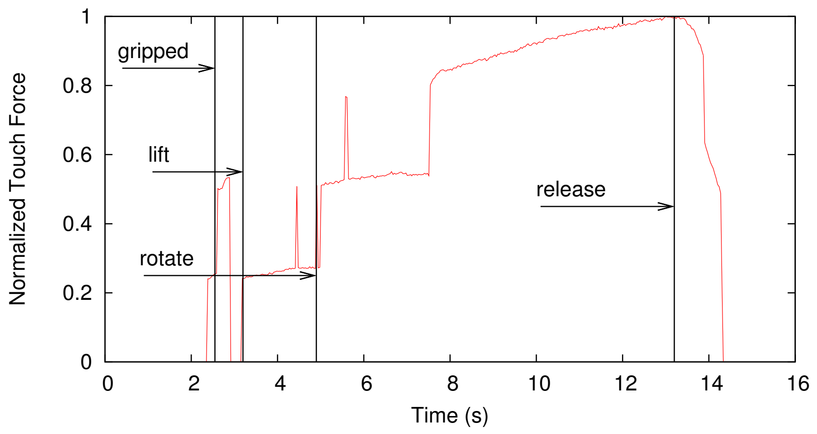
Figure 17: Grammar fragment for guarded moves

the robot can perform its task and small enough to not injure the human or damage itself. When the parser detects [limit], it stops and backs off, preventing damage or injury. The plot in 16(a) shows the forces encountered by the robot in this situation. The large spike at 4.7 s occurs when the robot's end-effector makes contact with the human's hand pictured in 16(b). The grammar in Fig. 17 *guarantees* that when this situation occurs, the robot will stop. After the human removes his hand from the piece, the robot can then safely reattempt its move

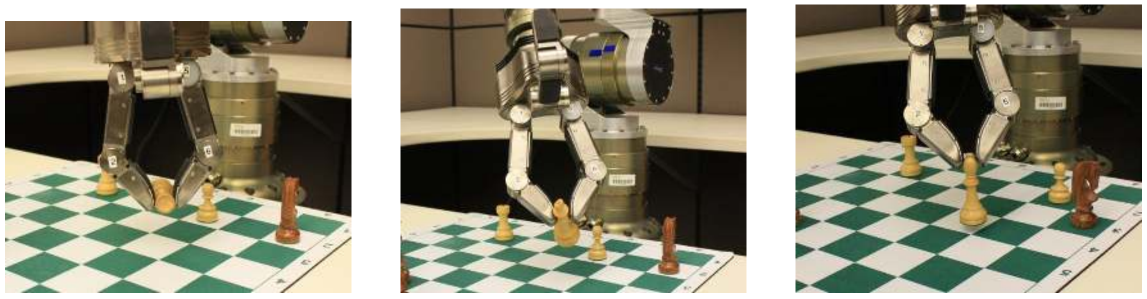
This example shows the importance of both response to unpredictable events – the human entering the workspace – and fast online control possible with the Motion Grammar. The robot must respond immediately to the dangerous situation of impact with the human. The polynomial runtime performance of Context-Free parsers means that the grammatical controller can respond quickly enough, and the syntax of Fig. 17 guarantees that the robot will stop moving according to the kinematic model. For guarded moves with a dynamic

$$\begin{aligned}\langle \mathrm{recover} : \mathbf{x}, \mathrm{z}\rangle &\rightarrow \langle \mathrm{G_D} : \mathbf{x}\rangle\langle \mathrm{pinch}\rangle\langle \mathrm{G_D} : \mathbf{x} + \mathrm{h}(\mathrm{z})\hat{k}, \frac{\pi}{6}\rangle\langle \mathrm{release}\rangle \\ \langle \mathrm{pinch}\rangle &\rightarrow [\mathrm{grasped}] \mid [\mathrm{ungrasped}]\,\langle \mathrm{pinch}\rangle\end{aligned}$$

Figure 18: Chess Grammar fragment for recovering fallen pieces



(a) Touch Force: Knight



(b) Grasped, Rook (c) Rotated, Queen (d) Finished, Bishop

Figure 19: Robot recovering fallen chess pieces

model, the method from [52] could be incorporated in place of the kinematic model here.

3.2.5.2 Fallen Pieces

The grammar to set fallen pieces upright has a fairly simple structure but builds upon the previous grammars to perform a more complicated task, demonstrating the advantages of a hierarchical decomposition for manipulation. This grammar is shown in Fig. 18, and Fig. 19 shows a plot of the finger tip forces and pictures for this process. The production $\langle \mathrm{recover} : \mathbf{x}, \mathrm{z}\rangle$ will pick up fallen piece z at location $\mathbf{x}$. The nonterminal $\langle \mathrm{G_D} : \mathbf{x}\rangle$ moves the

$$\begin{array}{rcl} \langle\text{reset board}\rangle & \rightarrow & [\text{set}] \mid [\text{misplaced}(x)]\,\langle\text{reset}: x, \text{home}(x)\rangle \\ \langle\text{reset}: x_0, x_1\rangle & \rightarrow & [\text{clear}(x_1)]\,\langle\text{move}: x_0, x_1\rangle \\ & \mid & [\text{occupied}(x_1)]\,\langle\text{reset}: x_1, \text{home}(x_1)\rangle\langle\text{move}: x_0, x_1\rangle \\ & \mid & [\text{cycle}(x_1)]\,\langle\text{move}: x_1, \text{rand}()\rangle \end{array}$$

Figure 20: Grammar fragment to reset chessboard

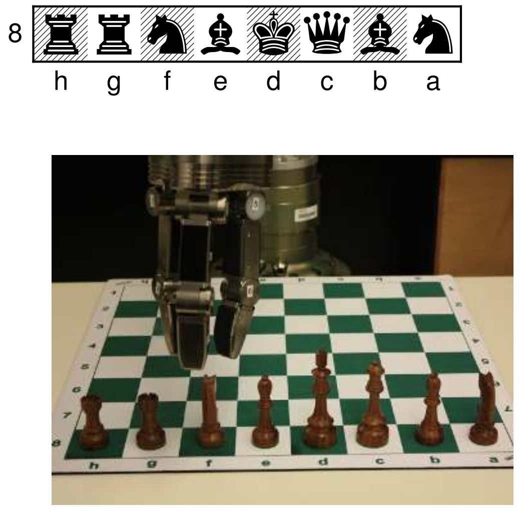
arm to location $\mathbf{x}$. The production $\langle\text{pinch}\rangle$ will grasp the piece by squeezing tighter until the fingertip pressure sensors indicate a sufficient force. The production $\langle G_D : \mathbf{x} + h(z)\hat{k}, \frac{\pi}{6}\rangle$ will lift the piece sufficiently high above the ground and rotate it so that it can be replaced upright. Finally the nonterminal $\langle\text{release}\rangle$ will release the grasp on the piece setting it upright.

3.2.5.3 Board Resetting

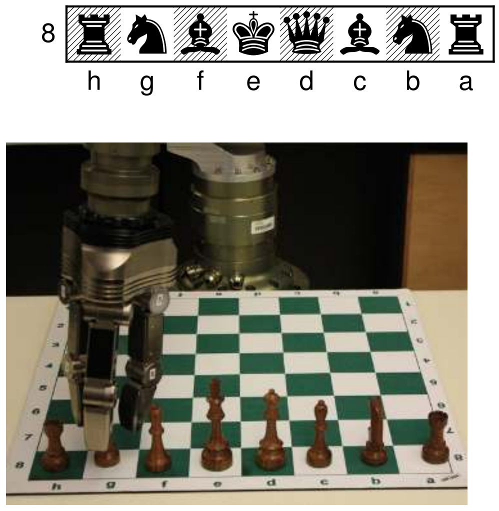
The problem of resetting the chess board presents an interesting grammatical structure. If the home square of some piece is occupied, that square must first be cleared before the piece can be reset. Additionally, if a cycle is discovered among the home squares of several pieces, the cycle must be broken before any piece can be properly placed. The grammatical productions to perform these actions a given in Fig. 20.

An example of this problem is shown in 21(a) where all of Blacks's Row 8 pieces have been shifted right by one square. The parse tree for this example is shown in 21(c), rooted at $\langle\text{reset board}\rangle$. As the robot recurses through the grammar in Fig. 20, chaining an additional $\langle\text{reset}\rangle$ for each occupied cell, it eventually discovers that a cycle exists between the pieces to move. To break the cycle, one piece, ♘c1, is moved to a random free square, χ. With the cycle broken, all the other pieces can be moved to their home squares. Finally, ♘χ can be moved back to its home square. This sequence of board state tokens and $\langle\text{move}\rangle$ actions can be seen by tracing the leaves of the parse tree, shown also beginning from PLAN in 21(c).

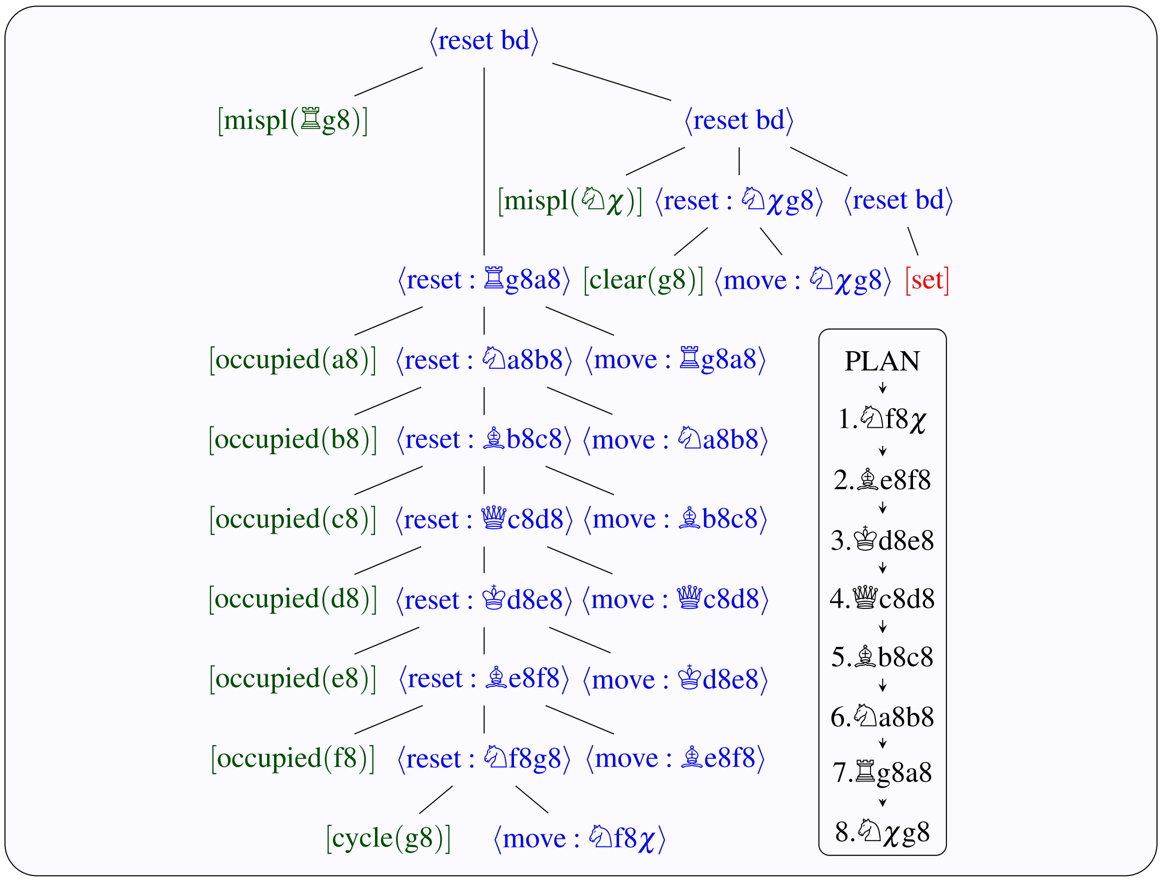
Observe that as the parser searches through the chain of pieces that occupy each other's



(a) Board position - Initial

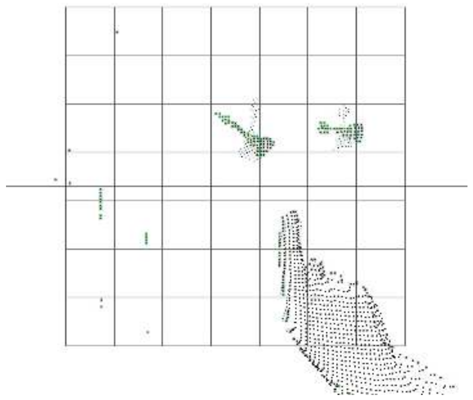


(b) Board position - Final



(c) Motion grammar parse tree and plan for resetting the board.

Figure 21: Example of board resetting


(a) Detecting obstacle (Black points are obstacles. Red/Green points indicate the orientation of each fallen piece.)

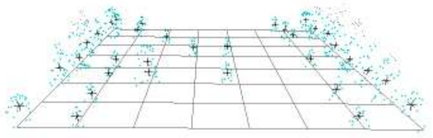

(b) Finding offsets for all pieces

Figure 22: Perception with point cloud is discretized into tokens.

home squares, it is effectively building up a stack of the moves to make. This demonstrates the benefits of the increased power of Context Free Languages over the Regular languages commonly used in other hybrid control systems. Regular languages, equivalent to finite state machines, lack the power to represent this arbitrary depth search.

Claim 1. *Let n be the number of misplaced pieces on the board. The grammar in Fig. 20 will reset the board with at most $1.5n$ moves.*

Proof. Every misplaced piece not in a cycle takes one move to reset to its proper square. Every cycle causes one additional move in order to break the cycle. A cycle requires two or more pieces, so there can be at most $0.5n$ cycles. Thus one move for every piece and one move for $0.5n$ cycles give a maximum of $1.5n$ moves. □

3.2.5.4 Perception and Board Tokens

To play the game of chess, we integrated our controller with the Crafty [91] chess engine. The Crafty *boardstate* serves as the model of the position of the chessboard. The MESA SR4000 point cloud is discretized by clustering to generate the tokens given in Table 3. We use a finite moving average filter over the point cloud to remove sensor noise.

Obstacles are found in the point cloud $\mathbf{C}$ by a weighting function $w(\mathbf{C})$ which finds out if the workspace above the chessboard is occupied or not. An example of an obstacle is shown in 22(a). We give the following attributes to each cluster in the point cloud: the weight of the cluster, the height of the cluster from the chessboard, the maximum area occupied by a cross section parallel to the chessboard, and the mean of the cluster. Here, the weight is denoted by $w(.)$, and it counts the number of points in that cluster. The height of the cluster is the highest point in the cluster. The maximum area occupied by the chess piece is expressed as a ratio of its width and length. If the ratio is above a certain threshold, we can easily conclude that a chess piece is fallen. The longest side of the chess piece gives its orientation. The mean point gives the center of the chess piece. Fig. 22 shows these attributes in the point cloud.

If an obstacle is found, it is denoted by [obstacle]. Nearest Neighbor over the entire chessboard determines all squares x with [occupied(x)]. If a piece is not placed exactly in the center of the square, an offset is computed and denoted by [offset(x)]. The boardstate retrieved from perception is termed C_r and the one from the Crafty engine is C_c. C_r is with C_c reported by Crafty to find whether a move has been made. If a move has been made, then [clear(x)] and [misplaced(x)] are determined. Our perception algorithm also finds out the height, orientation, and the area occupied by a horizontal cross-section for each piece. Using this and a recursive nearest neighbor algorithm for clustering, we can find all [fallen(x)] as shown in Table 3.

3.2.5.5 Full Game

The entire motion planning and control policy is specified in the grammar in Fig. 23. This grammar describes the game, $\langle$game$\rangle$, as consisting of an alternating sequence of the robot moving, $\langle$act$\rangle$, followed by the human moving, $\langle$wait$\rangle$, until the game has ended, $\langle$end$\rangle$, via checkmate, resignation, or draw. When it is the robot's turn, it will correct any fallen pieces, $\langle$fix$\rangle$, make its move, and then again correct any pieces that may have fallen while it was

$$
\begin{array}{rcl}
\langle \text{game} \rangle & \rightarrow & \langle \text{act} \rangle \langle \text{end} \rangle \mid \langle \text{act} \rangle \langle \text{game}' \rangle \\
\langle \text{game}' \rangle & \rightarrow & \langle \text{wait} \rangle \langle \text{end} \rangle \mid \langle \text{wait} \rangle \langle \text{game} \rangle \\
\langle \text{end} \rangle & \rightarrow & [\text{checkmate}] \mid [\text{resign}] \mid [\text{draw}] \\
\langle \text{act} \rangle & \rightarrow & \langle \text{fix} \rangle \langle \text{turn} \rangle \langle \text{fix} \rangle \\
\langle \text{fix} \rangle & \rightarrow & \langle \text{end} \rangle \mid [\text{fallen} : \text{x}, \text{z}] \, \langle \text{recover} : \text{x}, \text{z} \rangle \langle \text{fix} \rangle \mid \varepsilon \\
\langle \text{turn} \rangle & \rightarrow & \langle \text{move} : \text{x}_0, \text{x}_1 \rangle \mid \langle \text{capture} : \text{x}_0, \text{x}_1 \rangle \\
 & \mid & \langle \text{castle} \rangle \mid \langle \text{castle queen} \rangle \mid \langle \text{en passent} \rangle \\
 & \mid & \langle \text{resign} \rangle \mid \langle \text{draw} \rangle \\
\langle \text{wait} \rangle & \rightarrow & [\text{moved}] \mid \langle \text{wait} \rangle \\
\langle \text{move} : \text{x}_0, \text{x}_1 \rangle & \rightarrow & \langle \text{grasp piece} : \text{x}_0 \rangle \langle \text{place piece} : \text{x}_1 \rangle \\
\langle \text{grasp piece} : \text{x} \rangle & \rightarrow & \langle \text{G}_\text{L} : \text{x} \rangle \langle \text{grasp piece} : \text{x} \rangle \mid \langle \text{G}_\text{D} : \text{x} \rangle \langle \text{grip} \rangle \\
\langle \text{place piece} : \text{x} \rangle & \rightarrow & \langle \text{G}_\text{L} : \text{x} \rangle \langle \text{place piece} : \text{x} \rangle \mid \langle \text{G}_\text{D} : \text{x} \rangle \langle \text{ungrip} \rangle \\
\langle \text{grip} \rangle & \rightarrow & [\text{grasped}] \mid [\text{ungrasped}] \, \langle \text{grip} \rangle \\
\langle \text{capture} : \text{x}_0, \text{x}_1 \rangle & \rightarrow & \langle \text{take} : \text{x}_1 \rangle \langle \text{move} : \text{x}_0, \text{x}_1 \rangle \\
\langle \text{take} : \text{x} \rangle & \rightarrow & \langle \text{move} : \text{x}, \text{offboard} \rangle \\
\langle \text{castle} \rangle & \rightarrow & \langle \text{move} : \text{♔e1g1} \rangle \langle \text{♖h1f1} \rangle \\
\langle \text{castle queen} \rangle & \rightarrow & \langle \text{move} : \text{♔e1c1} \rangle \langle \text{♖a1d1} \rangle \\
\langle \text{en passent} : \text{x} \rangle & \rightarrow & \langle \text{take} : \text{x} - 1 \rangle \langle \text{move} : \text{♙x} \rangle \\
\langle \text{resign} \rangle & \rightarrow & \langle \text{G}_\text{L} : \text{♔} + 1 \rangle \langle \text{resign} \rangle \mid \langle \text{G}_\text{D} : \text{♔} + 1 \rangle \langle \text{resign}' \rangle \\
\langle \text{resign}' \rangle & \rightarrow & \langle \text{G}_\text{L} : \text{♔} - 1 \rangle \langle \text{resign}' \rangle \mid \langle \text{G}_\text{D} : \text{♔} - 1 \rangle
\end{array}
$$

Figure 23: Grammar Productions for Chess Game

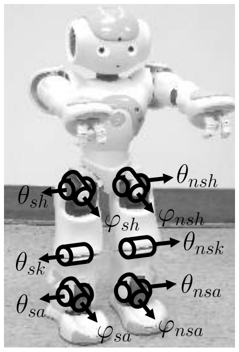


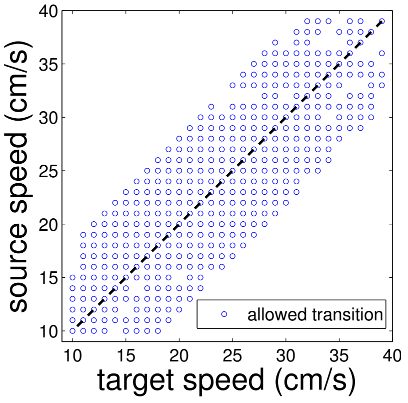
Figure 24: Aldebran Nao

making the move. Making a move, ⟨turn⟩, can be either a simple move between squares, a capture, a castle, en passent, or a draw or resignation. A simple piece move, ⟨move⟩, requires first grasping the piece, then placing it on the correct square. To grasp the piece, the robot will move its hand around then piece then tighten its grip, ⟨grip⟩, until there is sufficient pressure registered on the touch sensors. To capture a piece, the robot will remove the captured piece from the board, ⟨take⟩, and then move the capturing piece onto that square. A ⟨castle⟩ requires the robot to move both the rook and the king. For ⟨en passent⟩, the robot will ⟨take⟩ the captured pawn and then move its own pawn to the destination square. Finally, to resign – indicating a failure in chess strategy, not motion planning – the robot moves its end-effector through the square occupied by the king, knocking it over. By following the rules of this grammar, our system will play chess with the human opponent.

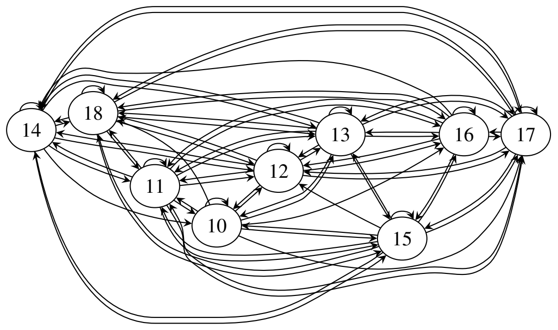
3.3 Walking Speed Graphs

Next, we apply this linguistic framework to the task of speed-controlled walking for the bipedal Nao robot. In this application we build on the method of Human-Inspired Control [8] aims for more human-like walking on bipedal robots [9, 145], generating stable control laws from human demonstrations. Then, we connect these control laws into a grammar representing the set of speed controlled actions available to the Nao robot.

Bipedal walking exhibits both continuous and discrete dynamics throughout the course of a step – the continuous behavior occurs when the non-stance leg swings freely and the



(a)



(b)

Figure 25: Graph Γ_s of permissible speed transitions. (a) full graph, 10-39cm/s. (b) partial graph, 10-18cm/s.

discrete behavior occurs when the non-stance foot strikes the ground [9]. It is, therefore, natural to model bipedal robots as hybrid systems.

The Aldebaran NAO robot, Fig. 24, is a 0.5 m, 5 kg bipedal robot with 25 degrees-of-freedom (DOF). We focus on controlling the NAO's legs, each of which has five DOF, and also control the shoulder pitch joints to better balance of upper body of the robot. The robot contains an on-board Intel Atom PC running GNU/Linux with the NAOqi software framework. This setup permits control of the robot's motors at 100 Hz. Additionally, Force Sensitive Resistors (FSR) located on the bottom of the feet detect the reactive force when the non-stance foot hits the ground (see Fig. 29).

Applying Human-Inspired control for the Nao approach give a graph of stable fixed speeds and transition speeds, Fig. 25 [41].

Based on the graph Γ_s of permissible speed transitions in Fig. 25, we proceed to construct the Motion Grammar for the system, which we will use to automatically generate the control software. First, we convert the speed graph (Fig. 25) to a Finite Automaton A_s (Fig. 26). Then we add symbols for transitions steps between different speeds to produce automaton A_T. Next, we replace each individual step symbol with symbols to set the appropriate parameter matrix and to take a step based on that matrix, producing grammar G_p.

Finally, we extend G_p with a grammar for discrete-time control of the individual steps. The result is a grammar $\hat{G}$ describing all sequences of walking speeds.

We first convert the graph of permissible speed transitions into a Finite Automaton (FA) for the language of permissible speed transitions. This means moving the important symbols – speeds for this walking domain – from the nodes in the graph to the edges in the FA. Fig. 26 shows the FA for transitions between 10 and 18 cm/s. The corresponding FA for the full system with transitions between 10 and 39 cm/s has 31 states, 30 terminals, and 401 edges. Algorithm 2 performs this transformation.

Rewriting the graph as an FA has a few benefits. First, we can apply many existing algorithms for Finite Automata such as Hopcroft's Algorithm for state minimization [85]. Crucially, abstracting the graph to an FA provides the automaton state as a computational *memory*, enabling more detailed decision making than simply stating which speeds may follow which other speeds. This will be necessary as we introduce the additional language symbols used for online parsing and supervisory control.

Algorithm 2: Graph to Finite Automaton

```
  Input: $\Gamma = (Q, E)$ ;                              // Graph
  Input: $w \in Q$ ;                         // Graph Initial Vertex
  Output: $A = (Q', Z', E', S')$ ;             // Finite Automaton
1 $Z' \leftarrow Q$;
2 $Q' \leftarrow Q$;
3 $S' \leftarrow w$;
4 $E' \leftarrow \{p \times p \times q \ : \ (p \times q) \in E\}$;
```

Definition 6 (Graph Traces)**.** *For graph $\Gamma = (V, E)$ and vertex $w \in V$, let Traces(w,Γ) be the set of sequences of symbols over V such that $\sigma \in$ Traces(w,Γ) if and only if $\sigma_0 = w$ and for every σ_i and σ_{i+1}, $\sigma_i \times \sigma_{i+1} \in E$.*

Lemma 1. *In algorithm 2 with $\Gamma = (Q, E)$, Traces(w, Γ) is equal to the language of Finite Automaton $A = (Q', Z', E', S')$.*

Proof. We prove by induction. Define σ^Γ and σ^A as strings in Traces(w, Γ) and the language of A, respectively. For the inductive case, consider some equal prefix of σ^Γ and σ^A

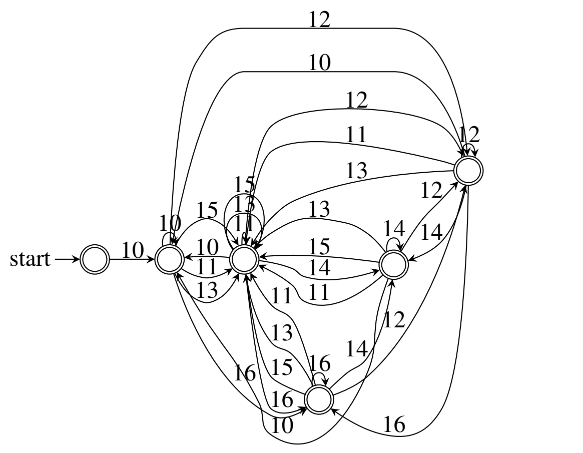


Figure 26: Minimum state FA A_s of speed transitions. Edge labels are speed in cm/s. Shown only for 10-16cm/s.

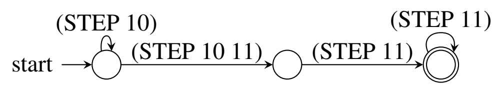


Figure 27: Finite Automaton fragment A_T showing transition step between 10 and 11 cm/s

of length i, $\forall j \le i$, $\sigma_j^\Gamma = \sigma_j^A$. σ_i^Γ is given by visiting state $\sigma_i \in Q$, and σ_{i+1}^Γ must be in the set $\{q\,:\,(\sigma_i \times q) \in E\}$. Each state p in A has outgoing edges which only contain terminal p, and at terminal σ_i^A, we will have reached some successor state of state σ_i which is in $\{q\,:\,(\sigma_i \times q) \in E\}$. Therefore, σ_{i+1}^A must be in $\{q\,:\,(\sigma_i \times q) \in Q\}$ as well. For the base case, $\sigma_0^\Gamma = w$. σ_0^A must be in the set $\{p\,:\,(S' \times p \times q) \in E'\}$, which is the set $\{S'\} = \{w\}$; therefore, $\sigma_0^A = w$ as well. $\square$

Now, we add to Fig. 26 the transition steps to go between different fixed walking speeds. Fig. 27 shows a fragment of the resulting FA. The full transition-step FA for 10-39 cm/s has 60 states, 400 terminals, and 430 edges.

Next, we replace each of the unique [(step x)] and [(step y z)] symbols with a semantic rule to apply the appropriate parameter matrix to walk at fixed speed x or transition from speed y to z followed by a symbol for the actual step. We also add transitions to terminate upon a special [HALT] symbol. This is shown as a grammar G_P in Fig. 28 for speeds of 10 and 11 cm/s. The full grammar for 10-39 cm/s has 490 productions.

Finally, we take each $\langle$step$\rangle$ symbol and decompose it with a discrete-time hybrid controller to take a single step, G_{step}. This step controller is shown as the grammar in Fig. 29.

$$\begin{array}{rcl}
\langle 0\rangle & \rightarrow & [\mathrm{HALT}] \\
 & | & \{\texttt{setparam}\ 10\}\,\langle\mathrm{step}\rangle\langle 0\rangle \\
 & | & \{\texttt{setparam}\ 10\ 11\}\,\langle\mathrm{step}\rangle\langle 1\rangle \\
\langle 1\rangle & \rightarrow & \{\texttt{setparam}\ 11\}\,\langle\mathrm{step}\rangle\langle 2\rangle \\
\langle 2\rangle & \rightarrow & \{\texttt{setparam}\ 11\}\,\langle\mathrm{step}\rangle\langle 2\rangle \\
 & | & [\mathrm{HALT}]
\end{array}$$

Figure 28: Parameter Grammar G_P

$$\begin{array}{rcl}
\langle\mathrm{step}\rangle & \rightarrow & \{\texttt{time} \leftarrow 0\}\,\langle \mathrm{S}_1\rangle \\
\langle \mathrm{S}_1\rangle & \rightarrow & \{\texttt{count} \leftarrow 0\}\,\{\kappa\}\,\langle \mathrm{S}_1'\rangle \\
\langle \mathrm{S}_1'\rangle & \rightarrow & [\texttt{weight} < \varepsilon_{\mathrm{w}}]\,\langle \mathrm{S}_1\rangle \\
 & | & [\texttt{weight} \geq \varepsilon_{\mathrm{w}}]\,\langle \mathrm{S}_1''\rangle \\
\langle \mathrm{S}_1''\rangle & \rightarrow & [\tau < \varepsilon_\tau]\,\langle \mathrm{S}_1\rangle \\
 & | & [\tau \geq \varepsilon_\tau]\,\langle \mathrm{S}_2\rangle \\
\langle \mathrm{S}_2\rangle & \rightarrow & \{\texttt{count} \leftarrow \texttt{count}+1\}\,\{\kappa\}\,\langle \mathrm{S}_2'\rangle \\
\langle \mathrm{S}_2'\rangle & \rightarrow & [\texttt{weight} < \varepsilon_{\mathrm{w}}]\,\langle \mathrm{S}_1\rangle \\
 & | & [\texttt{weight} \geq \varepsilon_{\mathrm{w}}]\,\langle \mathrm{S}_2''\rangle \\
\langle \mathrm{S}_2''\rangle & \rightarrow & [\texttt{count} < \varepsilon_{\mathrm{c}}]\,\langle \mathrm{S}_2\rangle \\
 & | & [\texttt{count} \geq \varepsilon_{\mathrm{c}}]\,\{\texttt{reset stance}\}
\end{array}$$

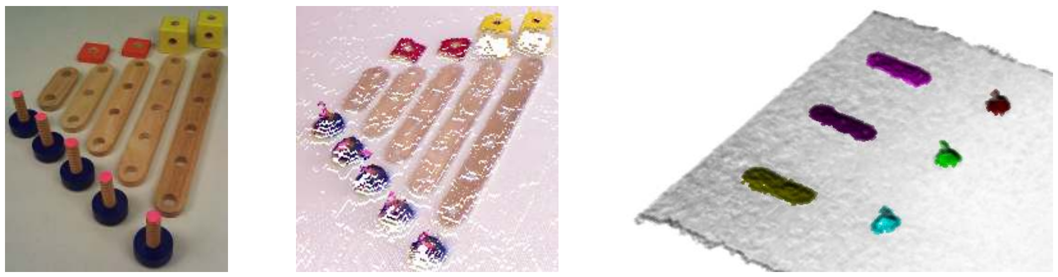
Figure 29: Step Grammar G_{step}

The $\{\kappa\}$ production in this grammar computes the inputs for the robot based on the current parameter matrix for the current control cycle, described in [146]. The productions of the resulting grammar $\hat{G}$ are the union of productions of G_P and G_{step}. Grammar $\hat{G}$ has 412 terminals, 70 nonterminals, and 504 productions, which represent a control policy for all sequences through speeds of 10-39 cm/s that the robot may take.

3.4 Inferring Grammars for Small Object Assembly

We demonstrate the automatic transfer of an assembly task from human to robot. This work extends performs real visual analysis of human demonstrations to automatically extract a policy for the task. We tokenize each human demonstration into a sequence of object connection symbols, then transforms the set of sequences from all demonstrations into an automaton, which represents the task-language for assembling a desired object. Finally, we combine this assembly automaton with a kinematic model of a robot arm to reproduce the demonstrated task.

Our experimental setup consists of an assembly kit of wooden pieces, a Kinect RGBD camera, and a simulated Schunk LWA3 7-DOF robot arm with Schunk SDH 7-DOF dexterous hand. From a physical human demonstration, we infer the control policy for the



(a) Construction Kit (b) Point Cloud (c) Segmentation/Clustering

Figure 30: Experimental Setup and Kinect Data

task and then implement that policy on the simulated Schunk robot. To capture the demonstration, the Kinect sensor is mounted above a table to observe a human performing the assembly task. The assembly pieces, shown in 30(a), come from a Melissa & Doug brand wooden construction set. The only modification we make to the pieces is to attach a brightly colored dot to the end of screws. This simplifies distinguishing the screw from an attached bar in the Kinect image, which has a limited resolution of 640 $\times$ 480 pixels. To illustrate our inference pipeline, we will show each of the steps required to build the simple assembly in 32(a). After inferring the policy from human demonstration, we simulate this policy with a kinematic model of the Schunk robot and then display the results, shown in Fig. 31, with the Peekabot visualization tool.

3.4.1 Assembly Language

In an object assembly, the connections between objects form a graph. In the simple case, objects are the graph nodes and connections between objects are the edges. However, we can make this model more precise by accounting for the multiple connection points on objects. To do this, we introduce additional nodes for these connection points. Each object node has edges to each of its connection point nodes. Each connection in the assembly is represented by an edge between the two graph nodes for the connection. This type of graph is shown in Fig. 32.

From the representation in Fig. 32, we can produce an appropriate set of event symbols.

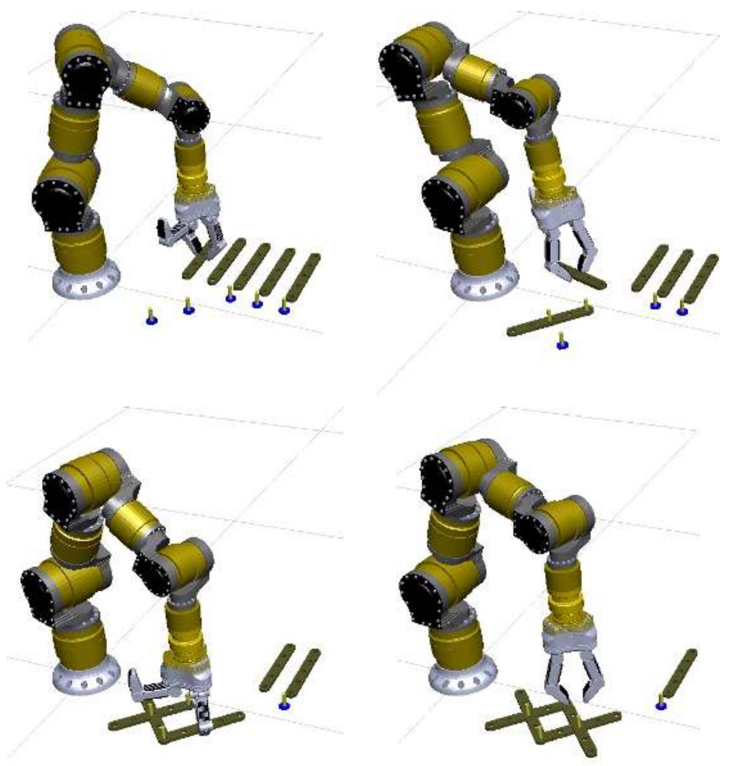
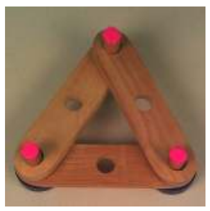
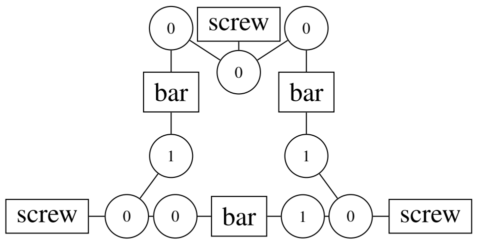


Figure 31: Robot Assembly



(a) Assembly



(b) Connection Graph

Figure 32: Connection Graph for an object assembly.

The meaningful events of the assembly domain are when the connection graph is modified by connecting a new object to the assembly or when creating an additional connection between objects already part of the assembly. This event is represented as the tuple $o_i \times c_j \times o_k \times c_\ell$, where o_α is some object and c_β is the connection point on that object. In our figures, we write these symbols as "p.q-r→x.y-z" where p is the type of object o_i, q is the object number of o_i, $p.q$ is then o_i, r is c_j, x is the object type of o_k, y is the object number of o_k, $x.y$ is then o_k, and z is c_ℓ. A sequence of these connection symbols represents the full construction of the assembly.

Definition 7 (Assembly Symbols)**.** *Let O be the finite set of objects. Let $C \subset \mathbb{N}$ be the finite set of connection points for any given object. Then the alphabet of assembly symbols is $Z = O \times C \times O \times C$.*

The language over these assembly symbols is a *syntactic* model of the assembly task policy. Each string in the language is a plan to assemble the desired object. In this language, the task is abstracted to the level where we can transfer it from human to robot. We then produce a Motion Grammar representing the hybrid dynamical control policy for the robot assembly task by combining this assembly language with the continuous *semantics* and lower level abstractions for our robot.

3.4.2 Human Activity to Event String

The first step in our system for automatically generating motion grammars is converting a human demonstration of the desired task, assembling an object, into a string of the connection events that the human performs. Given multiple example strings, we can then infer the Motion Grammar for the robot.

3.4.2.1 Image Segmentation and Clustering

First, we segment the RGBD image to identify the clusters representing objects and partial assemblies. Since the table is the largest feature in the image, we can robustly fit a plane to the table using RANSAC. For large objects, the height of each point above the table segments the object from the table. However, because some of our objects are within the depth sensing error of the Kinect, we cannot use the depth information alone. Instead, we combine the depth and color information to perform the segmentation.

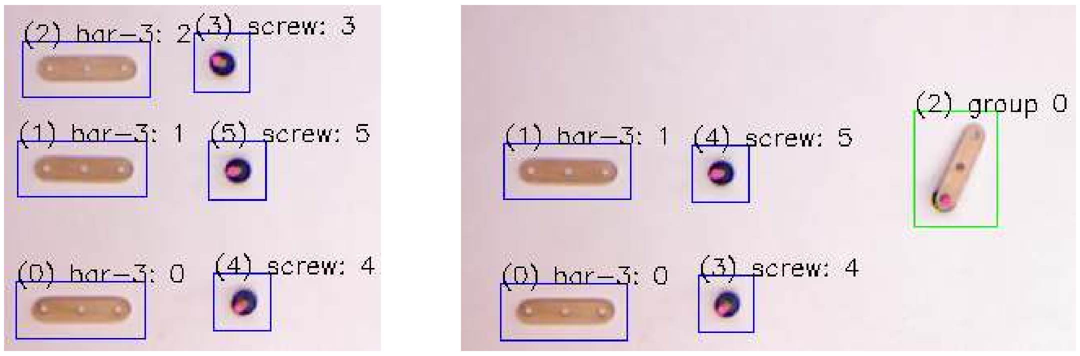
We perform segmentation by computing the Mahalanobis distance D_m of each point in the space of height above table z, hue h, and saturation s. This approach assumes a uniformly colored table, which is appropriate in our setup. To approximate the mean and variance of h and s, we iteratively compute these values for points on the table according to z, then reject outliers. Then, with the resulting mean and variance for the space, we compute the Mahalanobis distance for each point in the image using $D_m = \sqrt{(x-\mu)E^{-1}(x-\mu)}$, where $x = [z\, h\, s]^T$, μ is the mean of x, and E is a weight matrix.

All points with both distance D_m above a threshold and with z above the table are taken as part of objects or assemblies. These points are then clustered according to Euclidean distance (30(c)).

3.4.2.2 Object Recognition and Tracking

The next step is to recognize the objects that form each cluster and to track the objects across subsequent images. Since we have a small, fixed set of objects, we can recognize these objects by template matching in the RGB (sans D) image, 33(a). However, the tracking problem is complicated by two factors. First, many of our objects look identical so we cannot independently track them across subsequent frames. Second, because a human is moving the objects with his or her hands, tracking is only relevant when objects are occluded. We handle these issues by assuming that most of the objects in the frame are stationary, which is appropriate given that the human has only two hands to move objects. This assumption allows us to convert object tracking to the Assignment Problem.

The Assignment Problem is an optimization problem that consists of finding the minimum cost matching between two sets, A and B, where the distances between members of A



(a) First Image (b) Second Image

Figure 33: Recognized and labeled objects. Specific objects are tracked across subsequent images and objects combined into one cluster are grouped.

and B are known. Several subtasks in our inference pipeline are instances of this problem.

Definition 8 (Assignment Problem)**.** *Given sets A and B and distance function $d : A \times B \mapsto \mathbb{R}$, find the bijection $f : A \mapsto B$ such that the cost $J = \sum_{a \in A} d(a, f(a))$ is minimized.*

To convert object tracking to the assignment problem, we represent the point clusters in the initial frame as set A and in the subsequent frame as set B. The distance $d(a,b)$ is then the Euclidean distance between the centroids of the two clusters in each frame, $d(a,b) = \sqrt{x_a^T x_b}$. We can then solve this Assignment Problem using the Hungarian Algorithm, enabling us to track motion when multiple identical objects are moved without crossing.

By recognizing objects and tracking them across frames, we can determine when an object is added to an assembly, 33(b). In the event tuple, (o_i, c_j, o_k, c_ℓ), this gives us object o_i. The next step is to determine which other object o_i is connected to and how these two objects are connected.

3.4.2.3 *Structure Recognition*

To identify the precise connections between objects in an assembly, we first locate the individual objects within that assembly. Our system locates the bars and screws and then infers the connections between them.

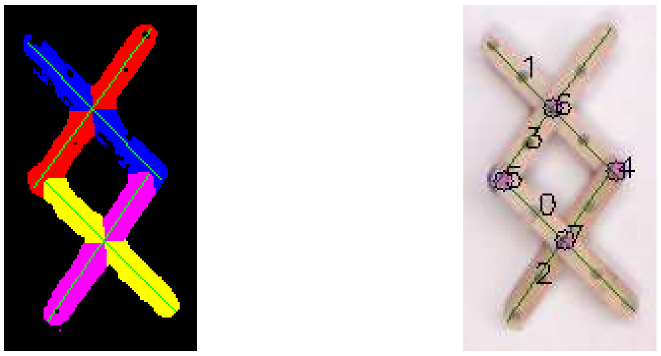

(a) Lines (b) Structure

Figure 34: Inferred structure of the object assembly. Segmented points are iteratively clustered and line fit.

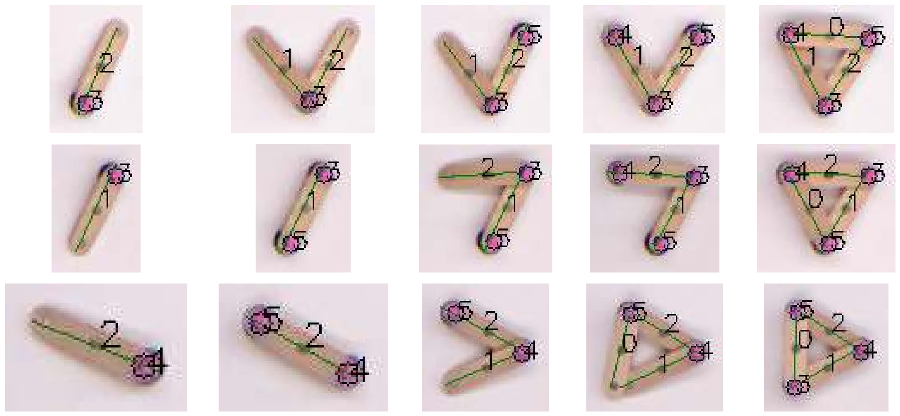


Figure 35: Each row is a demonstration sequence for the example object.

We locate the screws in the assembly using template matching. The bars are located by iteratively fitting lines and clustering points to the closest lines. The resulting lines are shown in 34(a).

Now that we have the locations of a number of identical bars and screws, we track the specific object for each located element. By assuming that the elements of the assembly are mostly stationary between frames, this becomes another instance of the assignment problem. The first set A is the located objects from the previous frame and the second set B is the elements in the current frame. Distance between sets is Euclidean distance between object positions, $d = \sqrt{x^T x}$. Solving this assignment problem gives the specific object for each located element.

“1.0-0→0.0-0” “1.0-0→0.1-0” “1.1-0→0.0-1” “1.2-0→0.1-1” “1.1-0→0.2-0” “1.2-0→0.2-1”

“1.0-0→0.0-0” “1.1-0→0.0-1” “1.2-0→0.1-0” “1.0-0→0.1-1” “1.1-0→0.2-0” “1.2-0→0.2-1”

“1.0-0→0.0-0” “1.1-0→0.0-1” “1.0-0→0.1-0” “1.1-0→0.2-0” “1.2-0→0.1-1” “1.2-0→0.2-1”

Figure 36: Three generated strings from demonstrations, one per row. Each string indicates the sequence of object connections. A connection between screw *i* and bar *k* at bar location ℓ is “1.i-0→ 0.k-ℓ.”

Having located the specific objects in the assembly, we now infer the connections between them. To do this, we assume that screws and bars can only connect at fixed locations on the bar, which is true for our construction set. Then we identify the connections with another assignment problem. The first set is the screws in the assembly. The second set is the connection points on the bars. Solving this gives us a connection for each screw and a single bar. To identify which screws connect multiple bars, we first identify the intersections between the lines for each bar. If that intersection goes through a screw, then that screw must connect the intersecting bars. Thus, we identify all connections between screws and bars in the assembly.

3.4.2.4 Symbol Generation

Given the connection graph at each frame, we can now abstract the demonstration to a sequence of symbols. Whenever the connection graph changes between subsequent frames, we add a new symbol representing that change to the sequence. Since our assemblies contain many identical objects, we also renumber the objects in the order they are added to the assembly. Some assembly strings are shown in Fig. 36. Thus, we abstract the human demonstration of assembly construction to a sequence of object connections which we use to infer a motion grammar for the robot to repeat the task.

3.4.3 Event Strings to Robot Grammar

Now that we have reduced the human demonstrations to an initial symbolic abstraction, we can transform this abstraction into a controller for the robot. First, we use the example

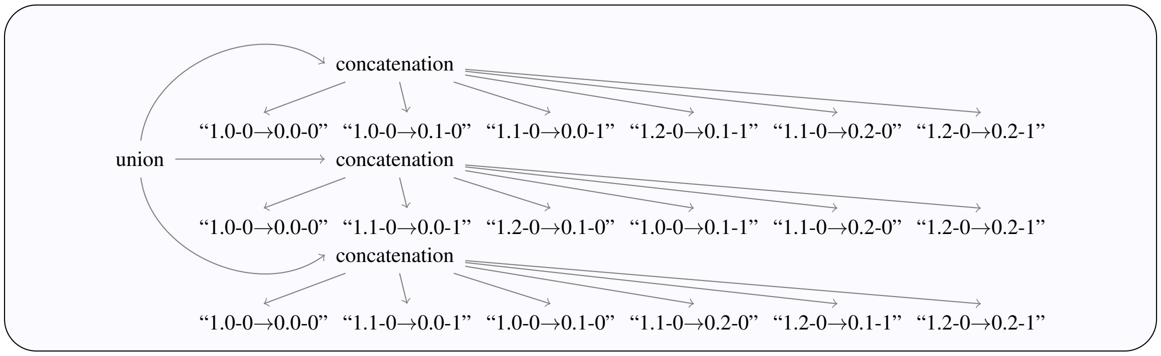


Figure 37: Regular Expression parse tree for Assembly Task.

strings to infer a syntactic model of the assembly task. Then, we combine the syntax of this assembly language with the semantic model of our robot to produce an MG for the demonstrated task.

3.4.3.1 Strings to Regular Expression

First, we convert the set of demonstration strings S to a regular expression R. This is directly accomplished by taking the union over all demonstration strings S. Thus, $R = \bigcup_{\sigma \in S} \sigma$. The language of this regular expression $L(R)$ is now a syntactic abstraction of all given demonstrations. For our example assembly, we transform the strings in Fig. 36 to the regular expression in Fig. 37. Notice that we introduce one union operator and k concatenation operators where k is the number of demonstrations, so the size of the regular expression is $O(1+k+pk) = O(pk)$.

3.4.3.2 Regular Expression to Nondeterministic Finite Automaton

Next, we convert the regular expression R to a Nondeterministic Finite Automaton (NFA) N using the McNaughton-Yamada-Thompson (MYT) algorithm [4, p159]. This transformation is always possible because Regular Expressions and NFA are equivalent representations. The MYT algorithm recursively walks the parse tree for the regular expression, producing an NFA which represents the same language. The resulting NFA for our example is shown in Fig. 38. Note that this NFA follows the conventional form of language

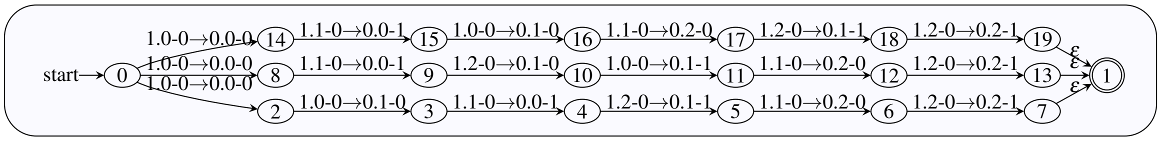


Figure 38: Inferred NFA for Assembly Task. Language symbols are on edges; state labels are arbitrary.

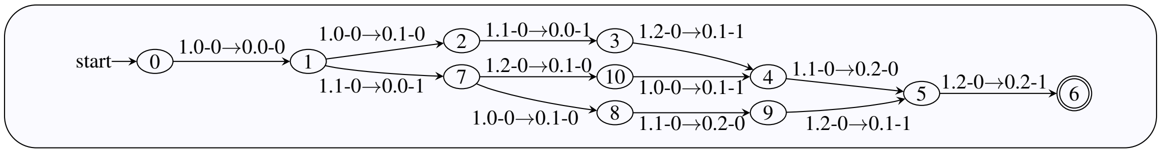


Figure 39: Minimum State DFA for Assembly Task.

symbols on edges and arbitrary state labels [4, 86].

The MYT algorithm visits each symbol of the Regular expression once and adds a constant bounded number of states and edges to the NFA for each regular expression symbol. Thus, for a regular expression of size n, the MYT algorithm runs in linear $O(n)$ time. Since our regular expression is size $O(pk)$, the MYT algorithm will run in time $O(pk)$ and produce an NFA of size $O(pk)$.

3.4.3.3 Nondeterministic Finite Automaton to Minimum Deterministic Finite Automaton

We now convert the assembly NFA to a minimum-state DFA using Brzozowski's algorithm [23]. Because NFA and DFA are equivalent representations, this transformation is always possible. Brzozowski's algorithm produces a minimum state DFA by reversing all connections in the FA and converting the result to a DFA, then repeating that procedure once more. The resulting DFA for assembly is shown in Fig. 39.

Analyzing the runtime of Brzozowski's Algorithm is more complicated than in the previous case. The NFA to DFA conversion in this algorithm has a worst-case exponential time, though typical performance is much better [4, p.153]. Brzozowski's Algorithm often outperforms Hopcroft's Algorithm for DFA minimization [85] which has $O(n \log n)$ runtime [26]. We provide implementations for both Brzozowski's and Hopcroft's Algorithms.

$$\begin{array}{rcl}
\langle \mathrm{o_i, c_j, o_k, c_\ell} \rangle & \rightarrow & [\neg \mathrm{placed(o_i)}]\, \langle \mathrm{P(x_i, x_{ws})} \rangle \langle \mathrm{o_i, c_j, o_k, c_\ell} \rangle \\
 & | & [\mathrm{placed(o_i)}]\, \langle \mathrm{P(x_k, x_i)} \rangle \\
\langle \mathrm{P} \rangle & \rightarrow & \langle \mathrm{pick} \rangle \langle \mathrm{place} \rangle \\
\langle \mathrm{pick} \rangle & \rightarrow & \langle \mathrm{move} \rangle \langle \mathrm{grasp} \rangle \\
\langle \mathrm{place} \rangle & \rightarrow & \langle \mathrm{move} \rangle \langle \mathrm{ungrasp} \rangle \\
\langle \mathrm{move} \rangle & \rightarrow & \langle \mathrm{T_1} \rangle\, [|\mathrm{x} - \mathrm{x_r}| < \varepsilon] \\
\langle \mathrm{grasp} \rangle & \rightarrow & \{\mathrm{close}\}\, [|\mathrm{x} - \mathrm{x_r}| < \varepsilon] \\
\langle \mathrm{ungrasp} \rangle & \rightarrow & \{\mathrm{open}\}\, [|\mathrm{x} - \mathrm{x_r}| < \varepsilon]
\end{array}$$

Figure 40: Pick and Place Grammar for Schunk LWA3 and SDH.

3.4.4 Manipulation Grammar

Next, we employ the grammar of Fig. 6 to hierarchically decompose the connection symbols from Fig. 39. This hierarchical decomposition exploits the power of Context-Free languages to compactly represent the task policy. The resulting grammar, shown in Fig. 40, will expand the connection symbols $\langle \mathrm{o_i, c_j, o_k, c_\ell} \rangle$ to sequences of robot trajectories necessary to perform the connection. Notice the $[\mathrm{placed}(\sigma)]$ and $[\neg\mathrm{placed}(\sigma)]$ symbols, which are an example of using sensors for memory in order to maintain an efficient model.

Note that to implement the assembly task, we must satisfy the geometric constraints in addition to the ordering constraints expressed by the DFA. One could naïvely handle these geometric constraints by initially placing objects arbitrarily and then later repositioning – or dragging – the object to satisfy the constraint. However, if we account for the geometry in our language, we can minimize this repositioning. Thus, we consider distances between pairwise connections in our language as follows. Given two symbols (o_i, c_j, o_k, c_ℓ) and (o_m, c_n, o_k, c_p), we observe that o_i and o_m are both connected to o_k and thus their positions are constrained by the distance between c_ℓ and c_p, given as $x_j - x_n = x_\ell - x_p$. When o_i is already placed, we select an x_n to satisfy this constraint,

$$x_n = (x_p - x_\ell) - x_j \tag{5}$$

Now, we expand the grammar of Fig. 40 to make the connection. First, if object o_n has not been placed, we place it at position x_{ws} calculated according to the constraint. Then, we pick o_k and place it. The picking and placing follow the trajectories of Fig. 6, and the robots grasps by pinching the object between two fingers of the SDH. Through the combination of this manipulation grammar and the inferred assembly automaton of Fig. 39, the robot reenacts the human demonstration, shown in Fig. 31.

3.5 Logical Planning Domains

Logical domains correspond to formal languages over propositions and actions [38, 50]. The formal language view provides a set of techniques for checking properties such as reachabilitity and safety over sets of states and in the presence of action uncertainty and unpredictable events [151]. However, because the worst-case size of finite automata representations is exponential in the number of propositions, it is a challenge to produce practically compact forms. To address this representational challenge in logical domains, we introduce (1) an algorithm to compute the minimum state deterministic finite automaton for a logical domain directly from the set of logical actions, (2) an algorithm to compute compact, context-free forms for hierarchical domains, and (3) independence conditions to enable separate solutions to subgoals. Inducing hierarchies on policies has several advantages. First, repeated subtasks need only be stored once, thereby compacting the representation. Second, inferred sub-policies can be reused as *high-level actions* when generating new policies, reducing computational demands while at the same time transferring existing knowledge to new tasks. In addition, hierarchies also present a task structure, allowing easier human inspection and analysis of the generated policies.

Applying Motion Grammars for logical domains enable compact representation of reactive robot controllers and directly connects logical domains with guarantees on performance and correctness in discrete event and hybrid systems. In this section, we generate a Motion Grammar representing a policy for a logical domain. In contrast to planning approaches, such as STRIPS [65], which generate single plans for meeting goals, we focus on generating robust control *policies* that will achieve the goals by specifying the robot's response to any occurring event. In contrast to single plans, policies encode different alternative executions of the task depending on environmental conditions. We show how the presented policy generation algorithm can be used to produce compact controllers for a robot manipulation and assembly task. Given a planning domain representing furniture components, we derive a hierarchical controller for assembling specific furniture pieces.

3.5.1 Minimum Finite-State Regular Policies

We first consider the connections between logical domains and the regular (finite-state) set. A planning domain defines interleaved sequences of state assignments and action symbols. Starting from some initial state S, we can select actions which lead to subsequent states. The domain defines a set of these sequences which are the potential plans. This corresponds to the definition of a formal language, which is also a set of sequences of symbols. *A planning domain is a formal language,* whose strings are the permissible plans.

Because the planning domains in 12 have a finite set of propositions, their state space is also finite, i.e., it is at most the space of boolean words over the propositions. The language of states and actions is therefore regular. algorithm 3 defines the naïve construction of a finite automaton for this language, by enumerating all states. The reverse translation, from a finite automaton to a planning domain, is also possible and defined in algorithm 4, which also minimizes the number of propositions in the planning domain (see 2) by compactly encoding the automata states.

Proposition 1. *Planning domains over finite propositions are equivalent to the regular set.*

Algorithm 3: Planning Domain to Regular Automataton

```
Input: $(\Phi, K, S_0, \Gamma)$ : Planning Domain
Output: $(Q, Z, E, q_0, F)$: Language
1 $Q \leftarrow 2^{\Phi}$ ;                /* all boolean words over propositions */
2 $E \leftarrow \{q_i \xrightarrow{k} q_j \ : \ k \in K \wedge \mathrm{pre}(k) \models q_i \wedge \mathrm{post}(k) \models q_j\}$;
3 return $(Q, K, E, S_0, \Gamma)$;
```

Proof. Construction using algorithm 3 and algorithm 4. □

Note that running algorithm 3 followed by algorithm 4 may not yield the original set of propositions but rather an equivalent representation. However, in the automata representation, we can always recover the canonical form [86].

While the state space resulting from algorithm 3 is exponential in the number of propositions, we can always find minimum-state forms of finite automata. Minimizing state removes transitions to states from which the goal is unreachable, yielding an automaton containing all paths to the goal and no paths not leading to the goal. This represents a policy for the domain; following any string in the automaton will reach the goal.

Hopcroft's algorithm finds the minimum state form by iteratively refining partitions of the state space until reaching a fixpoint [85]. However, the algorithm normally operates on an initial automaton, which in our case could be exponentially large. Instead, we modify this algorithm to operate not on an initial automaton, but directly on the logical domain. algorithm 5 shows this modification of Hopcroft's algorithm to directly produce the minimum state finite automaton from the logical domain.

In algorithm 5, we start with an initial partitioning Q of the states into the goal and non-goal states, line 2. Then, for each of the partitions q, we refine q into subgroups where for some action k, all states in the same subgroup transition to the same partition in Q. The key

Algorithm 4: Finite Automaton to Minimal Planning Domain

```
Input: $(Q, Z, E, q_0, F)$ : Language
Output: $(\Phi, K, S_0, \Gamma)$ : Planning Domain
1 $n \leftarrow |Q|$;
2 Renumber states $Q$ as $0, 1, \dots, n$;
3 $\Phi \leftarrow \phi_1, \dots, \phi_{\log_2 n}$;                /* binary encoding of states */
4 $K \leftarrow \emptyset$;
5 foreach $(q_i \xrightarrow{\kappa} q_j) \in E$ do
6     Add to $K$ a new action whose precondition is the binary encoding of $q_i$, action symbol is
      $\kappa$, and effect is the binary encoding of $q_j$.
7 $S_0 \leftarrow q_0$;
8 $\Gamma \leftarrow F$;
```

insight is to compute predecessor and successor states directly from the action precondition and effects, line 7, instead of requiring an initial, potentially much larger, automaton. Then, we refine sets of states using symbolic logical operations, line 10. This is repeated until we reach a fixpoint of Q.

Proposition 2. *To represent minimum state DFA $A = (Q, Z, E, q_0, F)$ as a planning domain, the minimum number of propositions necessary is* $\log_2 |Q|$.

Algorithm 5: Minimized Regular Automatazation

```
   Input: $(\Phi, K)$ : Planning Domain
   Input: $\Theta$ : Start Set
   Input: $\Gamma$ : Goal
   Output: $\mathfrak{L}$: Language
1  $Z \leftarrow \{\kappa : \kappa \in K\}$ ;                       /* all action symbols */
   /* Partition State                                          */
2  $Q \leftarrow \{\Gamma, \neg\Gamma\}$; // Initial Partitioning
3  $T \leftarrow \{\Gamma\}$;
4  while $T$ do
5      $q' \leftarrow \mathrm{pop}(T)$;
6      forall the $k \in K$ do
7          if $\mathrm{post}(k) \wedge q' \neq \mathrm{false}$ then
8              $Q^* = \emptyset$;
9              forall the $y \in Q$ do
10                 $i = y \wedge \mathrm{pre}(k)$;          // Subset of $y$ transitioning on $k$
11                 $j = y \wedge \neg\mathrm{pre}(k)$;    // Subset of $y$ not transitioning on $k$
12                 if $i \neq \mathrm{false} \wedge j \neq \mathrm{false}$ then
13                     $Q^* \leftarrow Q^* \cup \{i, j\}$; // Replace partition $y$ with $i$ and $j$
14                     if $y \in T$ then
15                         $T \leftarrow (T - y) \cup \{i, j\}$;
16                     else if $|i| < |j|$ then
17                         $T \leftarrow T \cup \{i\}$;
18                     else
19                         $T \leftarrow T \cup \{j\}$;
20                 else
21                     $Q^* \leftarrow Q^* \cup \{y\}$;                  // Don't split $y$
22             $Q \leftarrow Q^*$;
   /* Create transitions                                       */
23 $E \bigcup_{k \in K} \left\{ q_i \xrightarrow{k} q_j \right\} (q_i \wedge \mathrm{pre}(k) \neq \mathrm{false}) \wedge (q_j \wedge \mathrm{post}(k) \neq \mathrm{false})$;
24 return $(Q, Z, E, \Theta, \Gamma)$;
```

Proof. Assume we can represent A using k propositions. This will represent at most 2^k states. To represent $|Q|$ distinct states, it is necessary that $2^k \geq |Q|$, so $k \geq \log_2 |Q|$. $\square$

Representing logical policies as regular automata works for planning domains with sufficiently small state spaces to fit into memory. For planning domains with larger state spaces, the approach does not yield reasonably compact automata to be used as controllers on a robot system. Even for tasks of moderate complexity the size of generated automata can exceed the available memory. In the next section we show how CFGs can be used to create more compact, hierarchical representations of policies.

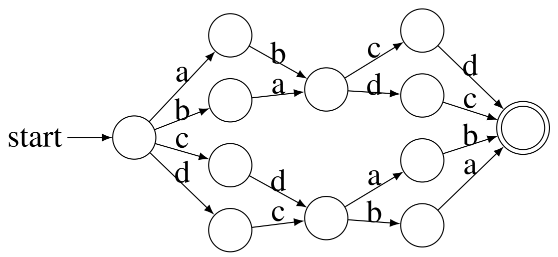
3.5.2 Hierarchically Compacted Context-Free Policies

To address the potentially large state spaces produced by regular automatization, we can instead use a context-free representation. Even though the *language* is regular, a context-free *representation* can reduce the necessary storage requirements by representing repeated subgoals in a hierarchy. Rather than duplicating a policy fragment to achieve the subgoal in a finite-state automaton, we store the fragment once and reference it though a grammar nonterminal symbol.

3.5.2.1 Automatic Hierarchization

The recursive structure of context-free languages compactly represents repeated subtasks. This closely corresponds to the *High Level Actions* (HLA) used in Hierarchical Task Networks (HTN) [62]. HTN generally focuses on efficiently finding a single plan given a pre-determined hierarchy. Our goal is to compactly represent a policy without requiring manual hierarchical decomposition.

We induce a hierarchization of the domain by identifying repeated *submachines* in an initial automaton, see Fig. 41. A submachine is an automaton containing a subset Q' of overall states Q and the edges corresponding only to Q'. The submachine defines a language to achieve a portion of the overall task.



(a)

$$
\begin{aligned}
\langle \mathrm{S} \rangle &\rightarrow \langle \mathrm{X} \rangle \langle \mathrm{Y} \rangle | \langle \mathrm{Y} \rangle \langle \mathrm{X} \rangle \\
\langle \mathrm{X} \rangle &\rightarrow ab|ba \\
\langle \mathrm{Y} \rangle &\rightarrow cd|dc
\end{aligned}
$$

(b)

Figure 41: Example of repeated submachines. (a), finite automaton. (b), context-free grammar for the same language. The repeated elements of the automaton are represented by grammar nonterminals $\langle \mathrm{X} \rangle$ and $\langle \mathrm{Y} \rangle$. Duplicate states for repeated submachines can be removed by referencing the nonterminal for the submachine.

Definition 9 (Submachine)**.** *Given an automaton* $A = (Q, Z, E, q_0, F)$*, a submachine is an automaton* $A' = (Q', Z', E', q'_0, F')$ *where* $Q' \subset Q$*,* $Z' = Z$*,* $E' = \left\{ q_i \xrightarrow{z} q_j \in E \ : \ q_i \in Q' \wedge q_j \in Q' \right\}$*,* $q'_0 \in Q'$*, and*
$F' = \{ q_i \in Q' \ : \ q_i \xrightarrow{z} q_j \in E \wedge q_j \notin Q' \}$.

To hierarchically compact the automaton, we find duplicated submachines to combine. Finite automata have canonical representations [85] which we can order lexigraphically. Thus, we count the number of occurrences of submachines in the original automaton using a tree of submachines. A submachine appearing more than once is replaced at each occurrence in the initial automaton with a nonterminal symbol reference to the submachine. This compacts the representation by storing equivalent submachines only once.

We describe the procedure to hierarchize an automaton A in algorithm 6. For a target submachine size, we first collect sets of connected states in A, line 2. Then, we count occurrences of the canonical automata represented by those components, line 6. Finally, we return the set of repeated canonical automata, which are the submachines to compact.

To improve the efficiency of algorithm 6, we apply a branch-and-bound-like approach to quickly discard subcomponents of state which have multiple exit nodes that transition to states outside the subcomponent. Components with a single exit node will have only one non-start with an edge to a state not in the component:

$$\left|\{q_i \in Q' - \{q'_0\} \;:\; q_i \xrightarrow{z} q_j \in E \wedge q_j \notin Q'\}\right| = 1 \tag{6}$$

We can check whether this condition will be violated before building the full component by considering the number of successors of states in the component. If the current component size, plus the successors of all but a candidate exit node exceed k, then a full component of size k cannot have a single exit:

$$f(q_i) = \left|\{q_k \notin Q' \;:\; q_i \xrightarrow{z} q_k \in E \wedge \neg\exists q_j \in Q',\, \left(q_j \xrightarrow{z} q_k \in E \wedge q_i \neq q_j\right)\}\right| \tag{7}$$

$$\left|Q'\right| + \left|\{q_k \notin Q' \;:\; q_i \xrightarrow{z} q_k \in E \wedge q_j \xrightarrow{z} q_k \in E \wedge q_i, q_j \in Q' \wedge q_i \neq q_j\}\right| + \sum_{q_i \in Q'} f(q_i) - \max_{q_i \in Q'} f(q_i) \leq k \tag{8}$$

Hierarchically compacting repeated submachines will reduce the representational size by the submachine size times the number of eliminated repetitions.

Proposition 3. *Given a submachine $A' = (Q', Z', E', q'_0, F')$ which appears k times in original automaton A, removing A' from A will reduce the number of states by $(k-1)|Q'| - 2k$.*

Algorithm 6: FA-Hierarchization

```
Input: $A = (Q, Z, E, q_o, F)$ : Finite Automaton
Input: $k$ : Submachine Size
Output: $H$ : Submachines
/* Identify K-Components                                    */
1 foreach $q \in Q$ do
2     $K_q \leftarrow \{p \;:\; p \subset Q \wedge |p| = k \wedge p$ has at most one exit node$\}$ ;
/* Count Occurrences of Canonical Automata                  */
3 foreach $K_q \in K$ do
4     foreach $p \in K_q$ do
5         $A \leftarrow$ cannonicalize($p$);
6         $C_A \leftarrow C_A + 1$ ;
/* Find Repeated Automata                                   */
7 $H \leftarrow \{A \;:\; C_A > 1\}$;
```

Proof. Initially, all occurrences of A' in A require $k|Q'|$ states. When A' is removed, it must be stored once, and each site expanding A' must have a predecessor and successor state to A', totalling $2k+|Q'|$ after removal. Then, $k|Q'|-(2k+|Q'|) = (k-1)|Q'|-2k$. □

3.5.2.2 Goal Independence

To further compact the automata state size, we consider the necessary conditions to produce independent solutions to different subgoals. Informally, subgoals are independent if we can develop plans and policies for one subgoal without negating other subgoals. Independent subgoals can thus be solved by concatenating plans and policies, turning a potential product of automata states into merely a sum.

Definition 10 (Weakly Independent Goals)**.** *Goals ϕ_1 is weakly independent of ϕ_2 over set of states Θ if for all θ in Θ, there exists a plan p_1 from θ to ϕ_1 and* there exists *plan p_2 from the postconditions of p_1 to ϕ_2 such that no states along p_2 entail $\neg\phi_1$.*

Definition 11 (Strongly Independent Goals)**.** *Goals ϕ_1 is strongly independent of ϕ_2 over set of states Θ if for all θ in Θ, there exists a plan p_1 from θ to ϕ_1 and* for all *plans p_2 from the postconditions of p_1 to ϕ_2, no states along p_2 entail $\neg\phi_1$.*

Weak Independence states that it is possible to solve goal ϕ_1, then solve ϕ_2 without violating ϕ_1. Strong Independence states that solving ϕ_2 never violates ϕ_1.

We can modify algorithm 5 to incorporate the check for independence. First, generate the automaton A_1 from Θ to ϕ_1. Then, generate the automaton A_2 from accept(A_1) to ϕ_2 while constraining ϕ_1 to hold in all states. We can express the constraint by changing the initial state partitioning to be $Q \leftarrow \{\phi_2 \wedge \phi_1, \neg\phi_2 \wedge \phi_1\}$.

Proposition 4. *If ϕ_1 is weakly independent of ϕ_2 over Θ, A_1 is the automaton to achieve ϕ_1, and A_2 is the automaton to achieve ϕ_2, then the size of the automaton A_{12} to achieve $\phi_1 \wedge \phi_2$ is $|A_{12}| = |A_1| + |A_2|$.*

Algorithm 7: Hierarchical Policy Grammar Generation

```
  Input: $D$ : Planning Domain
  Input: $\Theta$ : State set
  Input: $\Gamma$ : Goals
  Input: $\varsigma$ : Constraints
  Input: $H$ : High-level action library
  Output: $G$ : Planning Grammar
  /* Check for high-level action                                    */
1 if $\langle \Gamma, \Theta, \varsigma \rangle \in \mathrm{H}$ then
2     return $\langle \gamma, \Theta, \varsigma \rangle$;
  /* Check if we can independently solve a subgoal                  */
3 if $\exists \gamma \in \Gamma$, indep$(\gamma, \Gamma \backslash \gamma)$ then
4     $G_1 \leftarrow$ recurse$(D, \Theta, \gamma, \varsigma, H)$ ;          /* Recurse on subgoal */
5     $H \leftarrow H \cup G_1$ ;              /* Add subgoal solution to library */
6     return $G_1 \cup$ recurse$(D, \text{accept}(G_1), \Gamma \backslash \gamma, \varsigma \wedge \gamma, H)$;
7 else
      /* Dependent subgoal, cannot hierarchically decompose        */
8     Return the automaton to solve goal $\Gamma$, subject to constraint $\varsigma$;
```

Combining the independence definitions and induced hierarhical actions from algorithm 6, we introduce algorithm 7 to produce hierarhical grammars for a logical domain. This algorithm recursively constructs the hierarchical context-free grammar by applying the inferred high-level actions when possible, solving for independent goals when possible, and otherwise construction the minimized finite automaton solution. grammar for hierarchical planning. Initially, we check for an existing grammar fragment for the current goals, line 1. Otherwise, we check if a subgoal can be solved for independently, line 3. If so, then we recurse on only that subgoal and recurse on the remaining subgoals. If no subgoal is independent, we compute the automaton to simultaneously solve all subgoals, line 7.

3.5.3 Alternative Outcomes and Faults

In real-world systems, failures happen. Reliably executing manipulation tasks depends on handling errors and faults. Typical approaches for logical task planning identify a single execution path, without considering faults. This eases computation, but presents a challenge when the robot must respond to unexpected conditions. The principal challenge in explicitly representing policies for logical domains is handling the state space that is potentially in the number of propositions. We can address this challenge by using a linguistic, hierarchical representation for manipulation task policies. Using a language-based policy representation, we can compactly encode desired execution, potential faults, and the appropriate response.

We consider two types of errors and how they may be handled. First, we consider subtask or action failure, such as a missed grasp, where a chosen action does not produce the desired effect. Second, we consider uncontrollable events, such a force limits, where some transition occurs unpredictable or unavoidably. In both of these cases, we can use a language-based approach to continue and recover.

3.5.3.1 Subtask Failure

Manipulation subtasks may not always be executed correctly. For example, grasps may miss, objects may be dropped, and parts may not align. To reliably execute an overall task, a robot should gracefully handle these errors. We consider the issue of subtask failure by extending the logical planning domain to include alternative effects of actions.

Definition 12 (Alternative Propositional Planning Domain)**.** $D = (\Phi, K, S_0, \Gamma)$*, where* Φ *is the finite set of propositions, and* K *is the set of actions. Each* $K_i = (\alpha, \beta, \kappa, E)$*, where* $\alpha \subseteq \Phi$ *is the set of propositions which must be true before execution,* $\beta \subseteq \Phi$ *is the set of propositions which must be false before execution,* κ *is the action symbol, and* E *is the set of potential effects of the action* (a, d) *where* $a \subseteq \Phi$ *is the set of propositions made true by execution, and* $d \subseteq \Phi$ *is the set of propositions made false by execution. A state* S *is an assignment of propositions* Φ*. A set of states* Θ *corresponds to a boolean formula over* Φ*.* S_0 *is the initial state. The goal condition* Γ *is a list of subgoals in* $\Phi \times \{\text{true}, \text{false}\}$.

This variation on planning domains can be readily used with the techniques introduced in subsection 3.5.1. The key difference is to consider the alternative effects when using the variation of Hopcroft's algorithm to generate the initial minimum state automaton. When computing the predecessor states of some set, we must consider whether any alternative effect of each action leads to the current set. If so, then that action and effect denotes a valid predecessor. Once we have this initial automaton, then we can directly apply the previous methods for computing hierarchical decompositions and solving for independent subgoals.

3.5.3.2 Uncontrollable Events

While a robot can choose which discrete action to attempt, some discrete events may be uncontrollable. These events may represent system faults, hardware limits, or actions of other agents or humans. Controllable events may be blocked by a supervisor, while uncontrollable events may not. Control in the presense of uncontrollable events is well studied in the context of discrete event systems. Now, we relate this approach to logical task planning.

In general, we can test if system G is controllable with regard to specification S by considering prefixes on the controlled system G' which may be followed by an uncontrolled event. The prefixes of X are given as $\widetilde{X}$. All prefixes of the controlled system G' followed by an uncontrollable event which are prefixes of the original system G must also exist in G':

$$G' = G \cap S \qquad \widetilde{G'} Z_{uc} \cap \widetilde{G} \subseteq \widetilde{G'} \tag{9}$$

By representing logical planning domains using language, we can directly apply this result to uncontrollable events within those domains. The planning domain itself corresponds to system G and the goal corresponds to a specification S which is the set of all strings leading to the desired state. Then, we can apply (9) to check if this goal is achievable given the uncontrollable events.

```
(define (domain bimanual-move)
  (:action grasp-left
    (:precondition (not holding-left))
    (:effect (or (not holding-left) holding-left)))
  (:action grasp-right
    (:precondition (not holding-right))
    (:effect (or (not holding-right) holding-right)))
  (:action lift-left
    (:precondition (not heavy)
                   (not holding-right) holding-left)
    (:effect (or lifted heavy)))
  (:action lift-left-right
    (:precondition holding-left) (:effect lifted))
  (:action move
    (:precondition lifted) (:effect moving))
  (:action limit
    (:precondition moving) (:effect limit))
  (:action destination
    (:precondition moving) (:effect destination))
  (:action retract
    (:precondition limit) (:effect (not limit))))
```

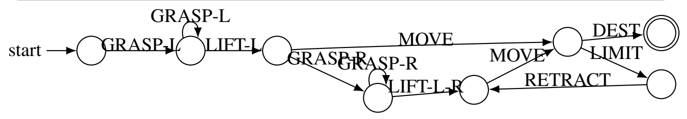


Figure 42: Example grasping domain for bimanual manipulation and the corresponding policy automaton. Faults include failed grasps, heavy objects require lifting with two hands, and limits which require stopping the motion.

3.5.4 Logical Domain Examples

Now we demonstrate this approach on two example domains. First, we show the automata for the classic Sussman Anomaly example. Then, we consider a furniture assembly task which induces many hierarchical subtasks.

3.5.4.1 Sussman Anomaly Revisited

The Sussman Anomaly is a minimal example domain demonstrating nonserializable goals [135], see Fig. 43. The goals in the Sussman anomaly domain are not strongly independent (see 11). Treating the goals independently may result in a suboptimal plan or even a cycle. We can, however, construct the minimum state automaton.

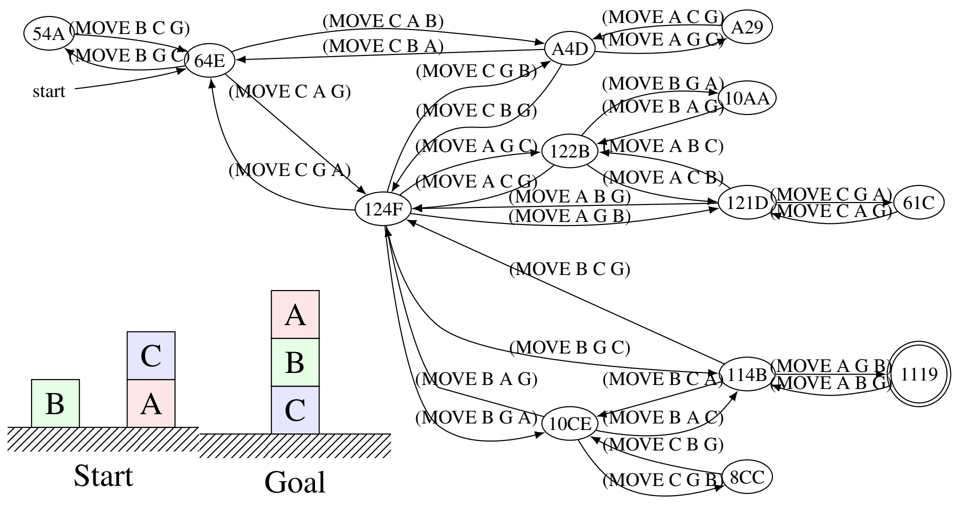


Figure 43: Sussman Anomaly and Finite Automaton. State labels are hexadecimal binary encoding of propositions: [(CLEAR A), (CLEAR B), (CLEAR C), (CLEAR G), (ON A B), (ON A C), (ON A G), (ON B A), (ON B C), (ON B G), (ON C A), (ON C B), (ON C G)].

3.5.4.2 Robot Assembly Tasks

Next, we consider a logical domain for assembling furniture pieces. Furniture assembly is particularly well suited for demonstrating the introduced policy generation algorithms because this domain presents large number of different combinations of actions, while at the same time containing a strong hierarchy of actions. A brute force approach enumerating all possible plans would be too costly, but there is potential for a compact task representation of identification of hierarchies and dependencies among actions.

Fig. 3.5.4.2 depicts the assembly domain used in the following experiments. A set of furniture elements, namely rods, brackets and sheets can be connected together through screws to assemble different furniture pieces. Each of the elements has a set of connections, represented in the figure by blue spheres. To connect rods to each other, a single screw is needed. To connect a rod to a sheet, a metal bracket and five screws are needed. We define the actions to align parts and connect screws in Fig. 45.

Within this assembly domain, we consider the task of assembling a table with four legs. Each leg is connected to table surface using a bracket. Four screws connect the bracket

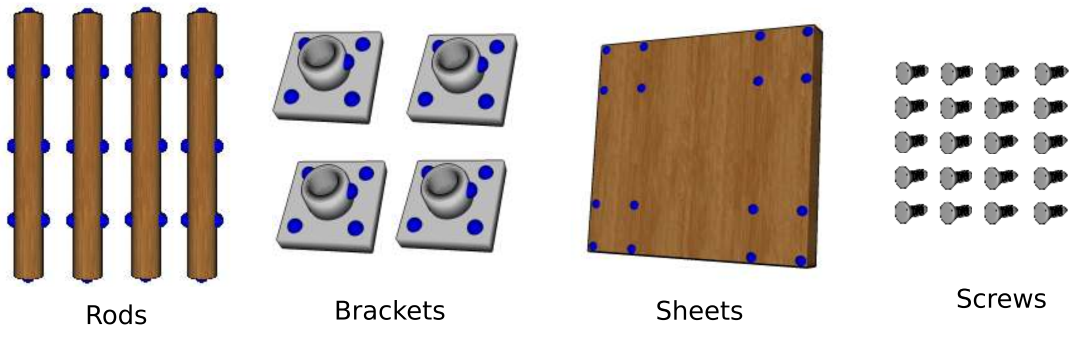


Figure 44: The assembly domain used in our experiments. The goal of this task is to connect furniture elements together in order to assemble a complete piece of furniture, e.g., a table. Each furniture element has a set of connection points (blue spheres) through which it can be connected to other elements. Rods can be connected to each other using a single screw. In order to connect a rod to a sheet, a metal bracket and five screws are needed.

Table 4: State Space Reduction via State Minimization and Hierarchization.

Representation	Size
Sussman Enumerated States	$2^{13} = 8192$
Sussman Minimum State Regular Form	13
Table Enumerated States	$2^{89} = 618970019642690137449562112$
Table Minimum State Regular Form	3321
Table Hierarchical Context-Free Form	183

to the surface and one screw connects the leg to the bracket. From the actions operating on 89 propositions, we produce the minimum state regular automaton and a hierarchical context-free representations. Fig. 46 shows an n inferred high-level action to align the screw holes on the bracket and table surface. Building this, Fig. 47 aligns and screws the bracket. Similarly, Fig. 48 aligns and screws a rod. The top-level of the induced hierarchy is shown in Fig. 49. From the initial 89 propositions, we infer a final hierchical representation containing 183 states across all levels. The sizes of these representations are summared in Table 4.

3.6 Relationship of Grammars and Other Representations

The Motion Grammar builds on a number of advances in linguistic control. This subsection relates our approach to several similar methods: Petri Nets, Hybrid Automata, MDLe,

```
(define (domain assembly)
  (:action (screw x x-i y y-j)
    (:precondition
     (free x) (clear x-i) (clear y-j) (aligned x-i y-j))
    (:effect (not (free x)) (not (free y))
             (not (clear x-i)) (not (clear y-j))
             (screwed x-i y-j)))
  (:action (align x x-i y-j)
    (:precondition
     (free x) (clear x-i) (clear y-j))
    (:effect (aligned x-i y-j))))
```

Figure 45: Logical actions for the assembly domain. Two pieces are assembled by screwing together their respective connection points. Before screwing together, the connections points must be aligned. To align points, the object itself must not be screwed down, i.e., it is free to move.

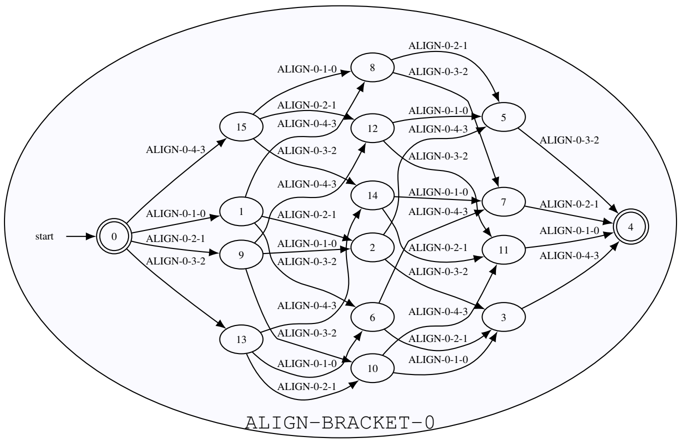
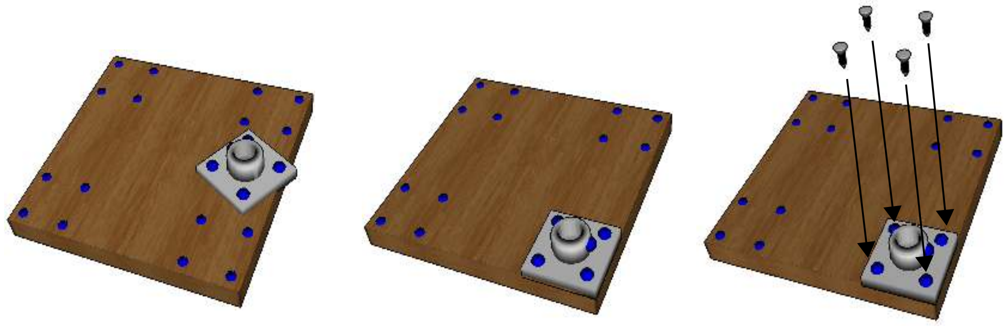
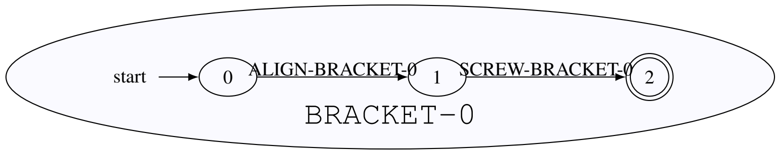


Figure 46: A high-level action to align all screw holes on a bracket.


(a)



(b)

Figure 47: A high-level action to align and screw and bracket to the table surface. Note that the edge labels in (b) are high-levels actions as well.

Maneuver Automata, Linear Temporal Logic, and the C Programming Language.

3.6.1 Petri Nets

Petri Nets are a modeling technique for discrete event systems based on a bipartite graph that represents the structure and dependencies of event firing. They are often used to model concurrent systems while CFGs generally represent a sequential structure. The languages that can be represented by a Petri Net are distinct from the Context-Free set. The language of some string followed by its reverse, $\left\{ww^R | w \in Z^*\right\}$, is Context-Free, but it is not a Petri Net language. The language of sequences of equal numbers of a, b, and c, $\{a^n b^n c^n\}$, is not Context-Free but can be represented by a Petri Net. However, the Petri Net languages are a strict superset of the Regular set and a strict subset of the Context-Sensitive set [143]. In consequence, the syntactic class of systems which can be modeled by a Petri Net is distinct from those modeled by the Context-Free Motion Grammar.

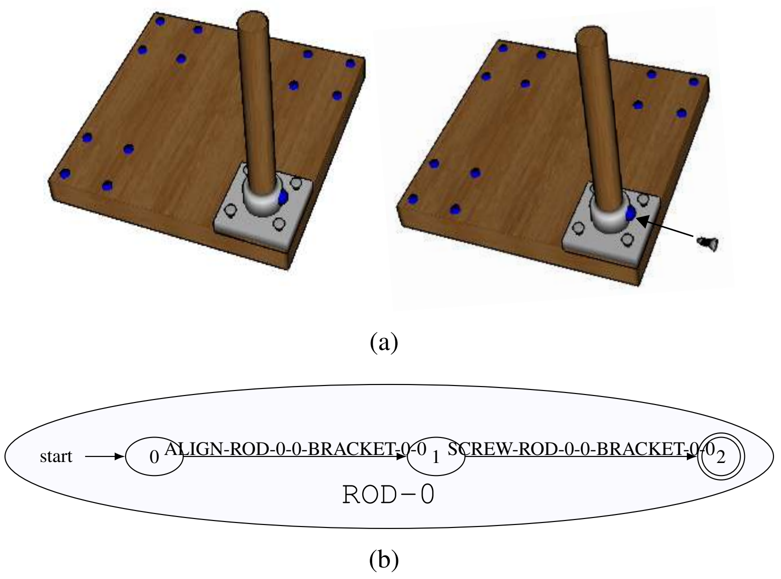


Figure 48: A high-level action to align and screw a rod to a bracket.

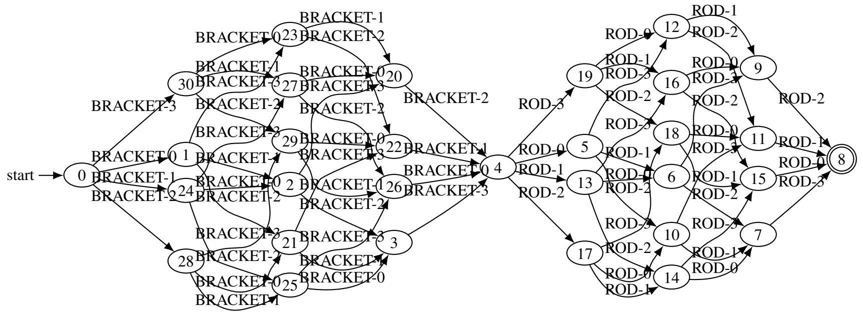
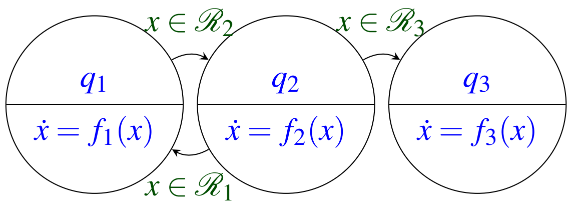


Figure 49: Top-Level Automaton for table assembly, incorporating High Level Actions from Fig. 46.

3.6.2 Hybrid Automata

Hybrid Automata represent a system with both event and time-driven dynamics. The system has a number of modes $q \in Q$. Each mode q_i is governed by some differential equation f_i. Transitions between modes occur in response to discrete events. The modes Q are generally finite [6, 81], so we can represent these transitions with a Finite Automaton. Many descriptions of Hybrid Automata also define *jump sets* or *reset conditions* which discontinuously change state x; this is not a feature we consider in this analysis.

A Hybrid Automaton with finite control states or modes Q can be transformed into an



(a) Hybrid Automaton

PRODUCTION	SEMANTIC RULES
$\langle \mathrm{Q}_1 \rangle \to [\mathrm{x} \in \mathscr{R}_2]\, Q_2$	$\dot{x} = f_1(x)$
$\langle \mathrm{Q}_2 \rangle \to [\mathrm{x} \in \mathscr{R}_1]\, Q_1$	$\dot{x} = f_2(x)$
$\mid\ [\mathrm{x} \in \mathscr{R}_3]\, Q_3$	$\dot{x} = f_2(x)$
$\langle \mathrm{Q}_3 \rangle \to \varepsilon$	$\dot{x} = f_3(x)$

(b) Motion Grammar

Figure 50: Example of Hybrid Automata to Motion Grammar Conversion

equivalent Motion Grammar. This is possible because every Finite Automaton is equivalent to a Regular Grammar, and Regular Grammars are a subset of Context-Free Grammars. An example of this process for a three-state system is shown in Fig. 50. The algorithm to perform this transformation is given by Algorithm 8. Because the Motion Grammar is Context-Free, the reverse is not always possible, and there are Motion Grammars, such as Fig. 20, with no equivalent finite mode Hybrid Automaton.

Algorithm 8: HA-to-$\mathscr{G}_M(Q, \Sigma, E, F)$

```
Input: $Q$ : set of discrete states
Input: $\Sigma$ : alphabet of tokens
Input: $E$ : set of edges, $Q \times Q$
Input: $F$ : set of continuous dynamics functions associated with each state in $Q$
1 foreach $q_i \in Q$ do
2     Create nonterminal $\langle \mathrm{Q_i} \rangle$;
3 foreach $\sigma_i \in \Sigma$ do
4     Create token $[\sigma_\mathrm{i}]$;
5 foreach $e_j \in E,\ e_j : q_i \times \sigma_j \mapsto q_k$ do
6     Create production $Q_i \to [\sigma_\mathrm{j}]\ Q_k$ with semantic rule $\dot{x} = f_i(x)$;
```

3.6.3 MDLe

The MDLe is a Modeling Language with a Context-Free grammar [87]. Each string in the MDLe represents some control program. While the *modeling* (subsection 3.1.2) language MDLe is Context-Free, each of MDLe control programs can parse only a Regular Language *system* language. This is in contrast to the Motion Grammar which describes the *System*

Language for a *Context-Free System*.

Theorem 1. *The System Language recognized by an MDLe string is Regular.*

Proof. Given that an MDLe controller is represented by a string in the MDLe language, we prove that the resulting System Language is regular by providing an algorithm to transform any MDLe string, Σ, into a Finite Automaton, $A = (S, E, d)$ that accepts the System Language $\mathfrak{L}_g$. MDLe string Σ is composed of tokens [(], [)], [,], controllers $u \in U$, and interrupts $\xi \in B'$. Algorithm 9 creates the automaton A corresponding to Σ. Notice that any u or ξ which appears multiple times in Σ results in multiple states in the FA.

The resulting Finite Automaton encodes the evaluation rules for the MDLe string. Since we can transform Σ to a Finite Automaton, Σ must recognize a Regular System Language.

□

Algorithm 9: MDLe-to-FA(Σ, U, B')

```
   Input: $\Sigma$ : MDLe specification string
   Input: $U$ : set of controllers
   Input: $B'$ : set of interrupts
   /* Create States                                        */
1  S = $\Sigma - \{[(], [)], [,]\}$;
   /* Create Transitions                                   */
2  foreach $s \in S$ do
3      if $s \in U$ then
4          foreach $\xi_i$ enclosing $s$ in $\Sigma$ do
5              Create a transition $\left(s \xrightarrow{\xi_i=1} \xi_i\right)$;
6      if $s \in B'$ then
7          Create a transition $\left(s \xrightarrow{\varepsilon} r\right)$, where $r$ is the next $\sigma_i$ following s in $\Sigma$ such that $r \in S$;
```

Two examples of this conversion procedure are shown in Fig. 51, one simple case and one more complicated case. Unlike the transformation to Hybrid Automata in [87], we do not restrict repeated controllers in Σ to a single state in our system language Finite Automata. Notice also that there is ambiguity in the case of simultaneously active interrupt functions. [87] specifies that this is resolved via precedence among the different interrupts.

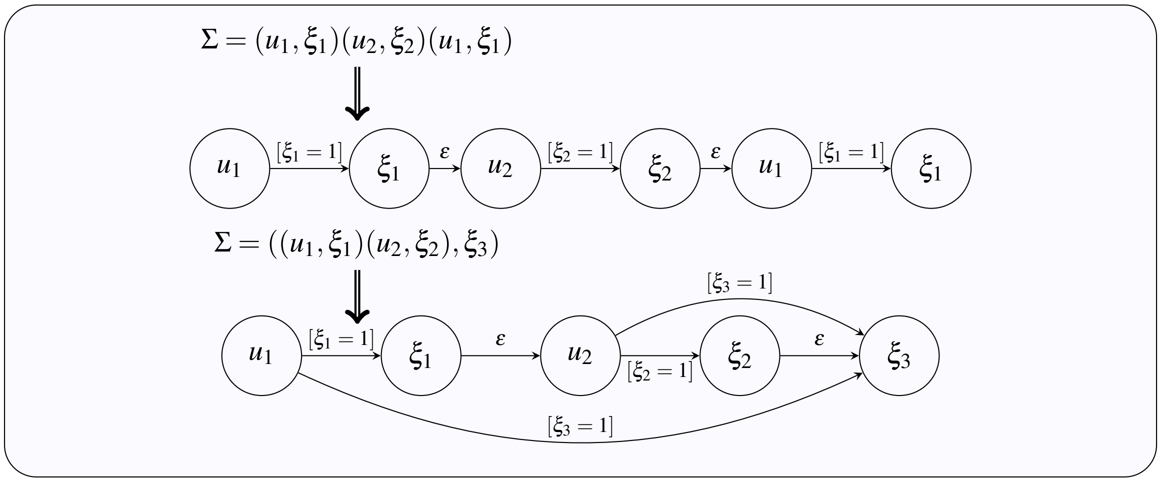


Figure 51: Example Transform: MDLe to Finite Automata

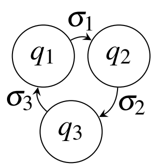
Corollary 1. *Every MDLe string can be translated to a Motion Grammar.*

Proof. The Motion Grammar is a Context-Free grammar for the System Language, and we can translate every MDLe string to a Finite Automaton accepting the System Language. Finite Automata are equivalent to Regular Grammars. Regular Grammars are a subset of Context-Free Grammars. □

From Corollary 1, we also observe that the Motion Grammar can control a broader class of systems than the MDLe. MDLe controllers accept only Regular Languages while the Motion Grammar accepts Context-Free languages with LL(1) semantics, which include all Regular Languages. Thus, the Motion Grammar can describe systems that the MDLe cannot.

3.6.4 Maneuver Automata

There are some important similarities between the Maneuver Automaton and the Motion Grammar. The Maneuver Automaton represents a hybrid system moving between a set of *trim* trajectories $q \in Q$ using a motion library of *maneuvers* $\sigma \in \Sigma$ [68]. This system is represented as a Finite Automaton with states Q and tokens Σ. It is possible to transform this representation into a grammar suitable for online control of the system. An example of this process is shown in Fig. 52. First, the Maneuver Automaton, 52(a) is rewritten



$$\begin{array}{lcl}
\langle \mathrm{q_1} \rangle & \rightarrow & [\sigma_1]\,\langle \mathrm{q_2} \rangle \\
\langle \mathrm{q_2} \rangle & \rightarrow & [\sigma_2]\,\langle \mathrm{q_3} \rangle \\
\langle \mathrm{q_3} \rangle & \rightarrow & [\sigma_3]\,\langle \mathrm{q_1} \rangle
\end{array}$$

(a) Offline Grammar

$$\begin{array}{lcl}
\langle \mathrm{q_1'} \rangle & \rightarrow & [\mathrm{x} \in \mathscr{R}_1] \mid [\mathrm{x} \in \mathscr{R}_1]\,\langle \sigma_1' \rangle \\
\langle \sigma_1' \rangle & \rightarrow & \kappa_1\,\langle \mathrm{q_2'} \rangle \\
\langle \mathrm{q_2'} \rangle & \rightarrow & [\mathrm{x} \in \mathscr{R}_2] \mid [\mathrm{x} \in \mathscr{R}_2]\,\langle \sigma_2' \rangle \\
\langle \sigma_2' \rangle & \rightarrow & \kappa_2\,\langle \mathrm{q_3} \rangle \\
\langle \mathrm{q_3'} \rangle & \rightarrow & [\mathrm{x} \in \mathscr{R}_3] \mid [\mathrm{x} \in \mathscr{R}_3]\,\langle \sigma_3' \rangle \\
\langle \sigma_3' \rangle & \rightarrow & \kappa_3\,\langle \mathrm{q_1'} \rangle
\end{array}$$

(b) Online Grammar

Figure 52: Maneuver Automaton $\rightarrow$ Online Grammar.

as a Regular Grammar, G_o in 52(a), with one production of the form $\langle \mathrm{q_i} \rangle \rightarrow \left[\sigma_\mathrm{j}\right] \langle \mathrm{q_k} \rangle$ to indicate each transition in the automaton. We then transform this offline grammar into an online grammar G_n according to Algorithm 10. Entry into a trim state is marked by a region of the continuous state space $x \in \mathscr{R}$. The controller for some maneuver σ is given by a semantic rule κ_σ.

Algorithm 10: G_o-to-$G_n(G_o)$

```
  /* Productions from states                                   */
1 foreach $\langle \mathrm{q_i} \rangle$ in $\mathrm{G_o}$ do
2     Create production $\langle \mathrm{q_i'} \rangle \rightarrow \left[\mathrm{x} \in \mathscr{R}_{\mathrm{q_i}}\right]$;
  /* Productions from transitions                              */
3 foreach $\langle \mathrm{q_i} \rangle \rightarrow \left[\sigma_\mathrm{j}\right] \langle \mathrm{q_k} \rangle$ in $\mathrm{G_o}$ do
4     Create production $\langle \mathrm{q_i'} \rangle \rightarrow \left[\mathrm{x} \in \mathscr{R}_{\mathrm{q_i}}\right] \langle \sigma_\mathrm{j}' \rangle$ ;
5     Create production $\langle \sigma_\mathrm{j}' \rangle \rightarrow \kappa_{\sigma_\mathrm{j}} \langle \mathrm{q_k'} \rangle$;
```

We also note that an arbitrary Maneuver Automaton cannot be directly transformed into a Motion Grammar. The Maneuver Automaton does not include information about how long to hold in trim states q or when to begin maneuvers σ. Thus, it does not represent a policy and it can be transformed only to a grammar that is not Semantically LL(1). Thus, Claim 1 indicates that it cannot be a Motion Grammar.

Even though we cannot directly transform a Maneuver Automaton to a Motion Grammar, this transformation is possible by adding the additional information necessary for

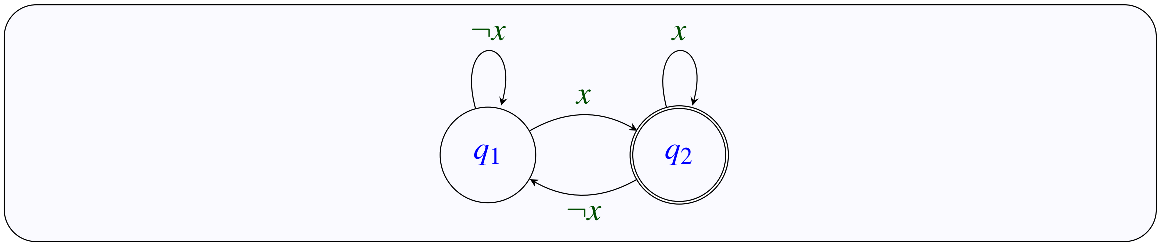

$$\begin{tikzcd} q_1 \arrow[loop above, "\neg x"] \arrow[r, bend left, "x"] & q_2 \arrow[loop above, "x"] \arrow[l, bend left, "\neg x"] \end{tikzcd}$$

Figure 53: Example of equivalence between Büchi Automata and Linear Temporal Logic formula $\square\Diamond x$.

LL(1) Semantics, such as by establishing precedence levels between conflicting productions or extending the representation to include tokens such as timeouts for coasting times. By augmenting the Maneuver Automaton with the additional information to achieve a policy, we can then derive a corresponding Motion Grammar.

3.6.5 Linear Temporal Logic

Linear Temporal Logic (LTL) is an extension to propositional logic that describes the behavior of discrete systems over an infinite time horizon. This is an often convenient notation to specify various system properties. Every statement in LTL can be represented as a Büchi automaton; an example is Fig. 53. Büchi automata are a variation on Regular automata that describe infinite length strings [13]. We can restate classical automata over finite length strings as a special case of automata over infinite length strings by looping through the accept state of a classical automaton [84, p.131].

Definition 13 (Stutter Extension)**.** *The stutter extension of finite string σ accepted by automaton A which halts with accept state q_n is the ω-run $\sigma,(q_n,\varepsilon,q_n)^{\omega}$ [84].*

Alternatively, we can specify that some LTL property α holds only until a particular terminating condition, \$, by replacing all $\square\alpha$ with $\alpha \cup \$$. Because of the correspondence between LTL and formal language, we may also use LTL formulas to describe correctness of the Motion Grammar. One algorithm for checking Context-Free systems with LTL is given by [63].

3.6.6 The C Programming Language

The C programming language is a Turing-Complete computational model while the Motion Grammar is Context-Free. Rice's theorem means that for an arbitrary C program, *we can guarantee nothing*, not even that it halts! Because the Motion Grammar is restricted to Context-Free computation, the Earley parser [59] means online parsing will have worst case polynomial runtime. Furthermore, Theorem 2 means that for an arbitrary Motion Grammar, *we can always verify it* against an arbitrary Regular specification. This makes clear the trade-off we have made: sacrifice computational power to guarantee runtime performance and verifiability. As a practical matter, though, any Motion Grammar may be transformed into a C program since all Context-Free languages are Turing-Recognizable.

CHAPTER IV

ANALYZING LANGUAGE MODELS

Using these specified linguistic models, we consider what kinds of reasoning about the system are possible. First, we discuss *completeness* and *correctness* of the model. Then, we develop a calculus of rewrite rules and apply these to the composition of semantic maps and manipulation rules. Finally, we discuss the impact of language class on system design.

4.1 Model Guarantees

4.1.1 Completeness

For a robot to be reliable, it must respond to any feasible situation. This requires a *policy*. For a Motion Grammar model $\mathcal{G}_M$ of system F to represent a policy, it must include the set of all *paths* that the system can take. This property is given by the simulation relation $F \preceq \mathcal{G}_M$, "$\mathcal{G}_M$ simulates F." The concrete definition of a path depends on type of system we are dealing with. For discrete systems, a path is the sequence of states and transitions the system takes. For continuous systems, a path is the trajectory though its state space [74]. For the hybrid systems we consider here, paths and simulation have both continuous and discrete components. Using 3 for path Ψ, we define simulation as follows,

Definition 14. *Given $\mathcal{G}_M$ and system F with $x(t), x'(t), u(t), u'(t) \in \mathcal{X}_F, \mathcal{X}_{\mathcal{G}_M}, \mathcal{U}_F, \mathcal{U}_{\mathcal{G}_M}$ for time t and initial conditions $x_0, x'_0 \in \mathcal{X}_F, \mathcal{X}_{\mathcal{G}_M}$.*
Then $F \preceq_c \mathcal{G}_M \equiv (x_0 = x_0' \wedge u(t) = u'(t) \implies x(t) = x'(t))$.

Definition 15. *Given $\mathcal{G}_M$ and system F then $F \preceq_d \mathcal{G}_M \equiv \mathfrak{L}(F) \subseteq \mathfrak{L}(\mathcal{G}_M)$*

Relation $F \preceq_c \mathcal{G}_M$ shows that F and $\mathcal{G}_M$ follow the same continuous trajectories. We match these trajectories exactly because a Motion Grammar must represent a policy and have LL(1) semantics – at each point along the path, $\mathcal{G}_M$ must specify a unique input u.

Thus, Def. 14 precludes grammars which specify infeasible trajectories of the physical system, such as moving to unreachable configurations, because such a grammar would not contain the true system trajectory. When the system F's $x(t)$ does not match the grammar $\mathscr{G}_M$'s $x(t)$ for the specified input u, this does not satisfy $\preceq_c$.

Relation $F \preceq_d \mathscr{G}_M$ shows that the language of the system is a subset of the language of Motion Grammar. Note that for events which represent region entry, $F \preceq_d \mathscr{G}_M$ is implied by $F \preceq_c \mathscr{G}_M$. We define $\preceq_d$ separately in order to model some events as purely discrete with no continuous-domain component.

Definition 16. *Given $\mathscr{G}_M$ and system F then* $\mathsf{complete}\{\mathscr{G}_M\} \equiv F \preceq \mathscr{G}_M \equiv F \preceq_c \mathscr{G}_M \wedge F \preceq_d \mathscr{G}_M$

Relation $F \preceq \mathscr{G}_M$ means that $\mathscr{G}_M$ is a faithful model of F which captures relevant system behavior, that all feasible paths are represented by $\mathscr{G}_M$. Proving simulation between arbitrary systems is a difficult problem. In the purely discrete Context-Free case, it is undecidable [86, p.203]. However, we can always disprove completeness with a counterexample: for x and y, a path of x not defined by y would prove $x \not\preceq y$. Our main concern, though, is not simulation between any two systems but that our model $\mathscr{G}_M$ simulate the physical system we wish to control. In this work, we approach simulation and completeness as a modeling problem. We match the productions of the model $\mathscr{G}_M$ to the operating modes and events of F, though we do have the freedom to specify input u and define new regions or switching points as is convenient.

In addition to providing a policy for the robot, a complete Motion Grammar has another important use: the grammar is a discrete abstraction for the entire system. We can use this abstraction to prove that the modeled system is *correct*.

4.1.2 Correctness

Given a policy for the robot, it is crucial to evaluate the correctness of that policy. We define the correctness of a language specified as a Motion Grammar, $\mathfrak{L}(\mathscr{G}_M)$, by relating it to a *constraint language*, $\mathfrak{L}_r$. While $\mathfrak{L}(\mathscr{G}_M)$ for a given problem integrates all problem subtasks, as shown in subsection 5.3.3, the constraint language targets correctness with respect to a specific criterion. Criteria can be formulated for general tasks, including safe operation, target acquisition, and the maintenance of desirable system attributes. By judiciously choosing the complexity of these languages, we can evaluate whether or not all strings generated by our model $\mathscr{G}_M$ are also part of language $\mathfrak{L}_r$.

Definition 17. *A Motion Grammar $\mathscr{G}_M$ is correct with respect to some constraint language $\mathfrak{L}_r$ when all strings in the language of $\mathscr{G}_M$ are also in $\mathfrak{L}_r$:* $\mathsf{correct}\{\mathscr{G}_M, \mathfrak{L}_\mathrm{r}\} \equiv \mathfrak{L}(\mathscr{G}_M) \subseteq \mathfrak{L}_r$.

This approach to verifying correctness provides a model-based guarantee on behavior, ensuring proper operation of the discrete abstraction represented by $\mathscr{G}_M$. This verification of the *model* $\mathscr{G}_M$ ensures correctness of the underlying physical system F to the extent that $\mathscr{G}_M$ is *complete*, 16. If we suppose system F contains some hybrid path ψ_bad with discrete component σ_bad and that ψ_bad is not in $\mathscr{G}_M$ – that is, $\mathscr{G}_M$ is not complete – then checking $\mathfrak{L}(\mathscr{G}_M) \subseteq \mathfrak{L}_r$ gives no information about whether $\sigma_\text{bad} \in \mathfrak{L}_r$. On the other hand, when $\mathscr{G}_M$ does contain the set of all feasible system paths, verifying $\mathscr{G}_M \subseteq \mathfrak{L}_r$ ensures correctness of all these paths. Thus, a complete model is necessary in order to meaningfully verify correctness.

The question of $\mathsf{correct}\{\mathscr{G}_M, \mathfrak{L}_\mathrm{r}\}$ is only decidable for certain language classes of $\mathfrak{L}(\mathscr{G}_M)$ and $\mathfrak{L}_r$. Hence, the formal guarantee on correctness is restricted to a limited range of complexity for both systems and constraints. We show decidability and undecidability for combinations of Regular, Deterministic Context-Free, and Context-Free Languages.

Lemma 2. *Let $\mathscr{L}_R$, $\mathscr{L}_D$, and $\mathscr{L}_C$ be the Regular, Deterministic Context-Free, and Context-Free sets, respectively, and let $R \in \mathscr{L}_R$, $D, D' \in \mathscr{L}_D$, and $C, C' \in \mathscr{L}_C$. Then,*

1. *$C \subseteq C'$ is undecidable. [86, p.203]*
2. *$R \subseteq C$ is undecidable. [86, p.203]*
3. *$C \subseteq R$ is decidable. [86, p.204]*

Table 5: Decidability of correct $\{\mathscr{G}_M, \mathfrak{L}_r\}$ by language class.

	$\mathfrak{L}_r \in \mathscr{L}_R$	$\mathfrak{L}_r \in \mathscr{L}_D$	$\mathfrak{L}_r \in \mathscr{L}_C$
$\mathfrak{L}(\mathscr{G}_M) \in \mathscr{L}_R$	yes	yes	no
$\mathfrak{L}(\mathscr{G}_M) \in \mathscr{L}_D$	yes	no	no
$\mathfrak{L}(\mathscr{G}_M) \in \mathscr{L}_C$	yes	no	no

4. *$R \subseteq D$ is decidable. [86, p.246]*
5. *$D \subseteq D'$ is undecidable. [86, p.247]*

Corollary 2. *Based on $\mathscr{L}_R \subset \mathscr{L}_D \subset \mathscr{L}_C$, the results from [86] extend to the following statements on decidability:*

1. *$D \subseteq R$ and $R \subseteq R$ are decidable.*
2. *$D \subseteq C$ is undecidable.*
3. *$C \subseteq D$ is undecidable.*

Combining these facts about language classes, the system designer can determine which types of languages can be used to define both the grammars for specific problems and general constraints.

Theorem 2. *The decidability of* correct $\{\mathscr{G}_M, \mathfrak{L}_r\}$ *for Regular, Deterministic Context-Free, and Context-Free Languages is specified by Table 5.*

Proof. Each entry in Table 5 combines a result from Lemma 2 or Corollary 2 with Definition 17. $\square$

Theorem 2 ensures that we can prove the correctness of a Motion Grammar with regard to any constraint languages in the permitted classes. We are limited to Regular constraint languages except in the case of a Regular system language which allows a Deterministic Context-Free constraint. Regular constraint languages may be specified as Finite Automata, Regular Grammars, or Regular Expressions since all are equivalent. We can also use Linear Temporal Logic as described in subsection 3.6.5.

To evaluate correct $\{\mathscr{G}_M, \mathfrak{L}_{\mathrm{r}}\}$, consider $\mathfrak{L}(\mathscr{G}_M) \subseteq \mathfrak{L}_r$ as, "Does $\mathfrak{L}(\mathscr{G}_M)$ contain any string *not* in $\mathfrak{L}_r$?" which gives equation (10) [13, p.163].

We can explicitly evaluate (10) by computing the Regular $\overline{\mathfrak{L}_r}$ [86, p.59], intersecting this with $\mathfrak{L}(\mathscr{G}_M)$ [86, p.135], then testing the Context-Free result for emptiness [66]. These algorithms are implemented in the Motion Grammar Kit.

$$\mathfrak{L}(\mathscr{G}_M) \cap \overline{\mathfrak{L}_r} \overset{?}{=} \emptyset \qquad (10)$$

4.2 *Discussion of Language Class*

We now discuss the impact of Formal Language classes on the system. The language class we use for our robot model determines our ability to execute, verify, and even represent our system. We discuss how these classes relate to assembly languages, and how we can design our language to operate within those classes which are suitably efficient and verifiable.

4.2.1 Chomsky Hierarchy of Languages

The formal languages classes form a hierarchy of increasing representative power and decreasing verifiability. The more general a language class, the less we can verify about the properties of languages within that class. The typical classes of this hierarchy are the *Regular*, *Context-Free*, *Context-Sensitive*, and *Recursively-Enumerable* Sets. Table 5 shows that for robotic systems, Regular and Context-Free sets permit both formal verification and efficient execution. The Context-Sensitive and Recursively-Enumerable sets do not. Thus, we have focused our efforts on producing models which are Context-Free or Regular.

4.2.2 Limits of Language Class

To understand the restrictions on language class and symbol selection, let us consider a simplified version of assembly, the task of connecting m independent screws, bars, and nuts, shown in 54(a). For each item, we must place a screw s, a bar b, and a nut n. We must insert the elements for any given item in order, and we must finish each item we start. Beyond that, we can construct items in any order. A few example strings in this language of Multiple Screw Assembly (MSA) we have just described are as follows,

- *sbnsbnsbn* (build one at a time)
- *sssbbbnnn* (insert all of a given symbol at a time)
- *ssbsbnnbn* (interleaved)

Now, we will show that MSA is not a Context-Free language using the pumping lemma. The pumping lemma is defined as follows [86],

Lemma 3 (Pumping Lemma)**.** *Let L be a CFL. There is a constant m called the pumping length depending only on L such that if z is in L, and $|z| \geq m$, we may write $z = uv^iwx^iy$ subject to*

1. $|vx| \geq 1$
2. $|vwx| \leq m$
3. $\forall i \geq 0,\ uv^iwx^iy \in L$

Theorem 3. *MSA is not Context Free.*

Proof. Proof by contradiction. Assume MSA is Context Free. Let m be the pumping length. Let $z = s^m b^m n^m$. Given $z = uvwxy$, vx must contain equal number of s, b and n. Otherwise, when we pump z to uv^2wx^2y, there would be incomplete items. Since $|vx| \geq 1$, vx must contain at least one of each symbol s, b, and n. For vx to contain at least one s and at least one n, vwx must straddle the b^m occurring in the middle of z. Thus, $vwx = s^a b^m n^a$ and $a \geq 1$. But if $vwx = s^a b^m n^a$, then $|vwx| > m$, violating condition 2 of the pumping lemma. This is a contradiction, so MSA is not Context-Free. $\square$

Theorem 4. *MSA is Context Sensitive.*

Proof. We prove by that MSA is context-sensitive by providing the following Linear Bounded Automaton (LBA) to recognize MSA. The input alphabet is the set $Z_i = \{b, s, n, \$\}$. The tape alphabet is the set $Z_t = \{s, b, n, e\}$. Initially, the tape contains e in the first cell. On each input s, scan the tape from left to right until e is found. Then replace the e with s and write an e in the next right cell. On each input b, and n, scan the tape from left to right looking for the matching s, or b respectively. Replace the tape symbol with the input symbol. If no matching symbol is found, reject. On input $ (end of input), scan the tape from left to right. If any symbol besides n or e is found, reject. Otherwise, accept. □

4.2.3 Language Classes and Symbol Design

Theorem 3 demonstrates the challenges when we restrict ourselves to policy representations which are efficient and fully decidable. The limited pushdown memory of a Context-Free systems permits decidability of a number of properties, yet it also limits the kind of systems we can represent. It may seem that even simple tasks with repeated actions cannot be modeled as Context-Free languages. However, this really illustrates the critical point of selecting appropriate language symbols. *By selecting the appropriate set of language symbols, we can still represent these systems and achieve both computational efficiency and formal verifiability.*

For automation systems with sensors, we can adopt a technique typical in behavior-based robotics that "the world is its own best model [20]." When we cannot store a necessary symbol in a verifiable automaton, we instead retrieve that symbol from a sensor, or at least from a source external to our formal language. Here, this approach is not merely a convenience or expedient, but mathematical necessity. The language classes with properties that are always decidable and efficiently parsable have limited representative power. If our desired dynamics are outside of that language class, part of our system description must necessarily come from outside of the language. By modifying the representation in this way, we can still verify the simplified linguistic model. It is only the non-Context-Free

(a) MSA Instance

$$\begin{array}{lcl} \langle \mathrm{A} \rangle & \to & [\mathrm{s}]\,\langle \mathrm{A} \rangle \\ & | & \left[\mathrm{s}'\right]\,[\mathrm{b}]\,\langle \mathrm{A} \rangle \\ & | & \left[\mathrm{b}'\right]\,[\mathrm{n}]\,\langle \mathrm{A} \rangle \\ & | & [\mathrm{e}] \end{array}$$

(b) Augmented Grammar

Figure 54: Multiple Screw Construction and a Context-Free Augmented Grammar.

dynamics which we cannot – and in the general case never can – verify.

We can apply this approach to the MSA problem by assuming a sensor which can detect the presence of partial items. Thus, we augment the token set to $Z = \{s,b,n\} \cup \{s',b',e\}$ where s' and b' indicate an uncovered s and b respectively, and e indicates no uncovered s or b. Then, we can model the system according to the grammar in 54(b).

We have also incorporated this idea into the symbol design for our object assembly language. Each connection symbol $o_i \times c_j \times o_k \times c_\ell$ in subsection 3.4.1 includes information about both the current world state and the next action to take. Thus, the assembly language is represented using only a Context-Free memory ensuring that we can both verify and efficiently execute the grammar.

4.3 Unpredictable Events

Robotic systems contain many sources of uncertainty. Linguistic approaches such as the Motion Grammar are well suited for addressing *unpredictable events* within the discrete dynamics. This occurs when at some point in time, the next token or discrete event is unknown. Other common sources of uncertainty include sensor noise, model error, and classification error.

A *complete* Motion Grammar (16) addresses unpredictable events by representing a linguistic *policy* over all feasible events. For example, in the human-robot chess match, the robot safely responds to the uncertain event of the human entering the workspace (subsubsection 3.2.5.1). Such a complete grammar defines a language which contains all strings of events which may occur, thus representing a policy to respond to those events.

Uncertainty due to sensor noise was an issue present in our human-robot chess implementation. To address this, we incorporated a Kalman Filter into the semantic rules K. This effectively attenuated the noise due to electromagnetic interference for the strain gauges in the wrist force-torque sensor. While Kalman Filters often operate well in practice, they do not guarantee robustness [58]. Additionally, error in state estimation may result in an event triggering due to estimated state which would not trigger due to actual state. When this is possible, additional grammar productions to handle the erroneous triggering are necessary. Thus, while our implementation was tolerant of the noise present in the system, further work is needed to formally address sensor noise.

One issue which we do not currently address in the Motion Grammar is multiple hypothesis state estimation such as that performed by a particle filter. This is important for applications such as visual tracking of humans. Extensions to the Motion Grammar such as stochastic or parallel parsing could address multiple hypothesis estimation. In addition, one could also preprocess the sensor data, though this will exist outside of the guaranteed model that the Motion Grammar provides. This type of uncertainty requiring multiple hypothesis estimation remains as another area for improvement.

4.4 Implementation Model Checking

A particular advantage of using formal language as an intermediate representation in a software synthesis pipeline is the ability to model check not just an abstracted model, but the actual structure used to generate the code. We apply this to the Nao speed control domain introduce in section 3.3. We verify that the *computation* defined by the derived grammar $\hat{G}$ properly represents the possible step sequences from the Motion Transition graph Fig. 25. Since the productions of $\hat{G}$ are the union of those of G_p and G_{step}, we verify that G_p represents a valid sequence of parameter settings and steps, knowing that each $\langle\text{step}\rangle$ in $\hat{G}$ will be properly decomposed by G_{step}.

First, we consider the overall structure of G_p, ensuring that each string in G_p is series of fixed speed steps, $\{\texttt{setparam}\ i\}\ \langle\text{step}\rangle$, connected by transition steps, $\{\texttt{setparam}\ i\ j\}\ \langle\text{step}\rangle$, and followed by [HALT]. This property is defined by the following regular expression S_a,

$$S_f = \bigcup_{i \in V_s} \left(\{\texttt{setparam}\ i\}\ \langle\text{step}\rangle\ (\{\texttt{setparam i}\}\ \langle\text{step}\rangle)^*\right)$$

$$S_t = \bigcup_{i \times j \in E_s} (\{\texttt{setparam}\ i\ j\}\ \langle\text{step}\rangle)$$

$$S_a = (S_t|\varepsilon)\left(S_f S_t\right)^*\left(S_f|\varepsilon\right)[\text{HALT}] \tag{11}$$

where $(E_s \times V_s)$ is the speed graph Γ_s and ε is the empty string. Note that S_f represents one or more steps at a single fixed speed and S_t represents a single transition step.

Second, we consider the speed transitions sequences in G_p to ensure that it contains no unstable paths according to the edges of Γ_s. For every $\{\texttt{setparam}\ i\}$ and $\{\texttt{setparam}\ i\ j\}$, we determine the valid predecessors and successors from the edges E_s of Γ_s,

$$\text{pred}(i,j) = \{\{\texttt{setparam}\ i\}\} \tag{12}$$

$$\text{pred}(j) = \{\{\texttt{setparam}\ i,j\}\ :\ i \times j \in E_s\} \tag{13}$$

$$\text{succ}(i,j) = \{\{\texttt{setparam}\ j\}\} \tag{14}$$

$$\text{succ}(i) = \{\{\texttt{setparam}\ i,j\}\ :\ i \times j \in E_s\} \tag{15}$$

$$\tag{16}$$

Then, we find the invalid predecessors and successors by removing those valid ones from the token set Z_p of grammar G_p,

$$S_{np}(i,j) = \bigcup_{z \in \left(Z_p - [\text{HALT}] - \langle\text{step}\rangle - \text{pred(i,j)}\right)} z \tag{17}$$

$$S_{ns}(i,j) = \bigcup_{z \in \left(Z_p - [\text{HALT}] - \langle\text{step}\rangle - \text{succ(i,j)}\right)} z \tag{18}$$

Next, we produce specifications to ensure that G_p contains no invalid transitions. For fixed speed i or transition $i \times j$, the specification is the complement of a string which includes and unstable transition to or from that i or $i \times j$.

$$S_p(i) = \overline{.^* S_{np}(i) \langle \text{step} \rangle \{\texttt{setparam i}\} .^*} \tag{19}$$

$$S_p(i,j) = \overline{.^* S_{np}(i,j) \langle \text{step} \rangle \{\texttt{setparam i j}\} .^*} \tag{20}$$

$$S_s(i) = \overline{.^* \{\texttt{setparam}\ i\} \langle \text{step} \rangle \mathrm{S_{ns}(i)}.^*} \tag{21}$$

$$S_s(i,j) = \overline{.^* \{\texttt{setparam}\ i\ j\} \langle \text{step} \rangle \mathrm{S_{ns}(i,i)}.^*} \tag{22}$$

From these specifications, we now formally verify that G_p is structurally valid and contains no unstable paths.

Proposition 5. *G_p is structurally valid and contains no unstable paths, that is:*
$G_p \subseteq S_a$ and
$\forall i \in V_s,\ (G_p \subseteq S_p(i) \wedge G_p \subseteq S_s(i))$ and
$\forall i \times j \in V_s \times V_s,\ (G_p \subseteq S_p(i,j) \wedge G_P \subseteq S_s(i,j))$

Proof. We verify this claim mechanically based on the model checking equation (10). Each specification S_a, $_p(i)$, $S_s(i)$, $S_p(i,j)$, and $S_s(i,j)$ is converted to a finite automaton A. Then, $\overline{A}$ is intersected with G_p. In all cases, the result is the $\emptyset$. $\square$

The software to perform this verification is implemented in the Motion Grammar Kit.

A critical point about this check is that it verifies not just an abstract model of the system, but the concrete representation – the data structure containing $\hat{G}$ – of computation to physically control the system. We use this same representation to synthesize the control software.

4.5 Motion Grammar Calculus

We introduce a calculus of rewrite rules for Motion Grammars [44].

4.5.1 Tokenization and Reachability

Tokens are instantaneous events which drive the discrete system dynamics. In this section, we focus on a particular type of event: entry into some region of interest within the continuous state space $\mathscr{X}$. Thus, the string of tokens in Z represents an abstracted path of the system through $\mathscr{X}$. Additionally, we will assume a fully observable system such that $x(t) \in \mathscr{X}$ is known for all t.

4.5.1.1 Region Tokens

There are different types of regions in $\mathscr{X}$ that may be relevant. In [159], position and velocity regions are used to produce robot trajectories. Here, we consider the general case of any region of interest in state space. The region for an event may be an area where the underlying dynamics of the system change, such as at a contact or impact. Regions may also be areas where we want our input to the system to abruptly change, such as a mobile robot reaching a way-point and switching to a new trajectory. A new token or event is generated when the system enters into the region.

Definition 18. *The token set Z is a set of regions representing a complete partition of the state space $\mathscr{X}$. For $[\mathrm{x} \in \mathscr{R}_\mathrm{i}] \in Z$,*

- $\mathscr{R}_i \cap \mathscr{R}_j = \emptyset$, $i \neq j$, *regions are nonoverlapping.*
- $\bigcup_{i=1}^{|Z|} \mathscr{R}_i = \mathscr{X}$*, regions cover the entire state space.*

Note that it is trivial to relax this condition by splitting overlapping regions. We use the non-overlapping formulation to simplify our analysis.

Tokenization is the process of breaking up the unstructured observation into a stream of tokens or events. This is the first step our controller must take to parse the observation. Because tokenization is implemented via digital logic or software, we use a discrete-time formulation. Based on 18, we define tokenization as follows,

Definition 19. *A new token is generated when the system crosses the boundary between two regions. In discrete time, when $x_{k-1} \in \mathscr{R}_i \wedge x_k \in \mathscr{R}_j \wedge i \neq j$, token $\left[\mathrm{x} \in \mathscr{R}_\mathrm{j}\right]$ is appended to the input tape.*

At each time step t_k, a discrete-time controller must compute which region it is in. If the region has changed since the previous step t_{k-1}, then the controller parses the token associated with the new region. One way to perform this tokenization is to express a region as bounded by codimension-1 manifolds $\mathscr{M}$ given as the level-set for some scalar function,

$$\mathscr{M} = \{x \, : \, s(x) = 0\} \tag{23}$$

Since the manifold is composed of all points where scalar $s(x) = 0$, the $\mathsf{sign}\left(s(x)\right)$ indicates the side of the manifold, and consequently the region, of any system state.

4.5.1.2 Conservative Reachability with Barrier Certificates

By defining tokens as regions, we can use the continuous dynamics to predict the discrete dynamics. This will be used in subsection 4.5.4, to transform the grammar. Observe that since tokens are regions, the set of discrete tokens which may be generated is equivalent to the set of reachable regions of continuous state. This problem has previously been addressed by others such [130] using Hamilton-Jacobi-Isaacs Partial Differential Equations (HJI PDE) to compute the backwards reachable set, and this method is indeed directly applicable to the Motion Grammar. However, it can often be very difficult to solve these HJI PDEs. If there are no known analytic solutions for the particular PDE of interest, it is often necessary to resort to numerical methods. The method of barrier certificates [148] instead considers behavior only along a specific boundary within the state space. This approach should be easier to evaluate and is directly applicable to our chosen method of tokenizing the state space.

We apply barrier certificates to the Motion Grammar using the Lie derivative along region boundaries. Consider the autonomous system dynamics given by smooth function

$\dot{x} = f(x)$. Let the boundary between $\mathscr{R}_i$ and $\mathscr{R}_j$ be given by the codimension-1 manifold $\mathscr{M}$, $c = s(x)$. The normal vector to $\mathscr{M}$ at point x is $\nabla s(x)$. To determine if the system will cross $\mathscr{M}$ at point x, we relate the direction of $\nabla s(x)$ and $f(x)$ using the sign of the Lie derivative, $\mathscr{L}_f s$,

Theorem 5. *Let $\mathscr{M} = \{x : s(x) = 0\}$. If $\mathscr{L}_f s(x) < 0 \; \forall x \in \mathscr{M}$, the system will never cross $\mathscr{M}$. If $\exists x \in \mathscr{M}$, $\mathscr{L}_f s(x) > 0$, the system* may *cross $\mathscr{M}$.*

Proof. Consider some $p \in \mathscr{M}$. Then $\mathscr{L}_f s(p) = \nabla s(p) \cdot f(p) = \|\nabla s(p)\| \, \|f(p)\| \cos\theta$, where θ is the angle between $\nabla s(p)$ and $f(p)$. When $\cos\theta > 0$, the system moves off $\mathscr{M}$ in the direction of increasing $s(x)$. When $\cos\theta < 0$, the system moves off $\mathscr{M}$ in the direction of decreasing $s(x)$. Since $\mathsf{sign}\,(\cos\theta) = \mathsf{sign}\left(\mathscr{L}_f s\right)$ we can use $\mathsf{sign}\left(\mathscr{L}_f s\right)$ to test which side of $\mathscr{M}$ the system will move to from p. If there is no p for which $\mathscr{L}_f s > 0$, then the system cannot move off the manifold to cross it. If there is any p for which $\mathscr{L}_f s > 0$, then from that p, the system will move off the manifold and thus cross it. □

Theorem 5 thus shows whether one region is directly reachable from another based on system dynamics $\dot{x} = f(x)$. It is conservative because it only says whether a boundary crossing occurs based on the local condition. The crossing occurs when both $\mathscr{L}_f s > 0$, and the global dynamics brings x to the local neighborhood of the boundary.

We can express $\mathscr{L}_f s$ only along $\mathscr{M}$ by parameterizing $\mathscr{M}$ by some $v \in \mathbb{R}^{n-1}$, where $\mathscr{X} \in \mathbb{R}^n$.

$$\mathscr{M} = \{x : x = \phi(v)\} \tag{24}$$

For example, if $\mathscr{M}$ is a hyperplane, then $\phi(v) = Mv$, where M is a $n \times (n-1)$ matrix. If $\mathscr{M}$ is a hypersphere, the ϕ can be defined to transform spherical coordinate vector v to Cartesian coordinates x. Then, we consider $\mathsf{sign}\left(\mathscr{L}_f s\left[\phi\left(v\right)\right]\right)$ to determine if the boundary may ever be crossed.

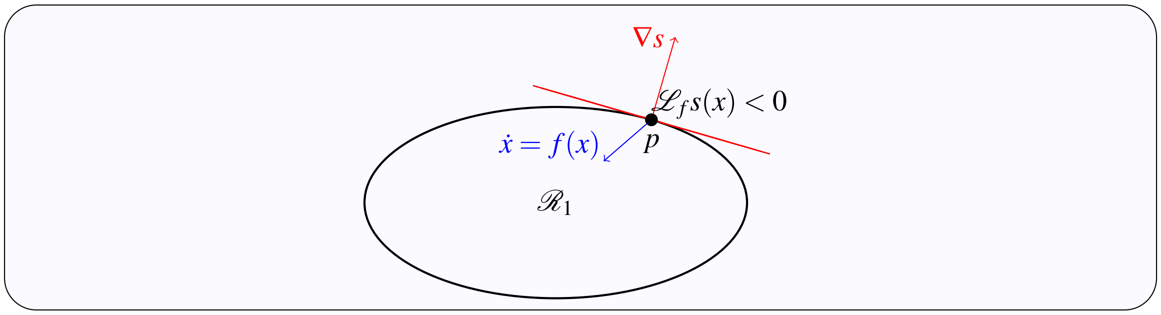


Figure 55: A region given by $1 = \frac{x^2}{2^2} + y^2$ and tangent planes by $1 + \frac{x_0{}^2}{2^2} + y_0{}^2 = \frac{2x_0}{2^2}x + 2y_0 y$. The system evolves by $\dot{x} = -x$, indicating that it stays within $\mathscr{R}_1$ at the boundary.

4.5.1.3 *Reachability Example*

Consider a region bounded by an ellipse centered on the origin as shown in Fig. 55.

$$M \equiv c = \frac{x_1{}^2}{a_1{}^2} + \frac{x_2{}^2}{a_2{}^2} \tag{25}$$

$$\nabla s(x) = \begin{bmatrix} \frac{x_1}{a_1{}^2} & \frac{x_2}{a_2{}^2} \end{bmatrix} \tag{26}$$

4.5.1.4 *System 1*

Consider time-driven dynamics $\dot{x} = -x$. This gives us $\mathscr{L}_f s = -\frac{x_1{}^2}{a_1{}^2} - \frac{x_2{}^2}{a_2{}^2}$. Since this is always negative, the system will not cross the boundary.

4.5.1.5 *System 2*

Consider time-driven dynamics $\dot{x} = \begin{bmatrix} y - x & -x - y \end{bmatrix}^T$. This gives us $\mathscr{L}_f s = \frac{x_1 x_2}{a_1{}^2} - \frac{x_1 x_2}{a_2{}^2} - \frac{x_1{}^2}{a_1{}^2} - \frac{x_2{}^2}{a_2{}^2}$. We can use a parameterization of the manifold by v, $x = \begin{bmatrix} a_1 \cos v & a_2 \sin v \end{bmatrix}$ to rewrite $\mathscr{L}_f s = \frac{a_2}{a_1} c_v s_v - \frac{a_1}{a_2} c_v s_v - 1$. From this, we see that if $\frac{a_1}{a_2} > 1 + \sqrt{2}$ or $\frac{a_2}{a_1} > 1 + \sqrt{2}$, the Lie derivative will be positive for some values of v meaning that the system will cross the boundary from some, but not all v.

4.5.1.6 *Additional Event Types*

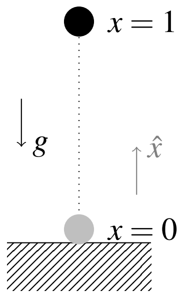
In addition to the region-based events of Def. 18-19 which result from the controllable continuous dynamics, we can model events resulting from uncontrollable continuous dynamics or purely-discrete dynamics as well. Such events may include faults, limit conditions, and even human actions or spoken words. All can be included as tokens in the grammar. In this paper however, we focus on the controllable region-entry events because it is by appropriately controlling these events that we can transform the grammar to achieve correctness.

4.5.2 Process to Derive Correct Grammars

To transform a *grammar which describes* the system into a *grammar that correctly controls* the system, we must consider which transformations are possible. Only certain transformations of the grammar are valid. We are particularly concerned with ensuring that our transformations maintain the property that the derived grammar $\mathscr{G}_M{}'$ is complete – that $\mathscr{G}_M{}'$ describes all *paths* that were possible in the original grammar $\mathscr{G}_M$, because $\mathscr{G}_M$ is an accurate representation of the hybrid system by definition. We show examples of transformations that lead to complete and incomplete $\mathscr{G}_M{}'$. Third, we define completeness and present a method for determining whether a class of transformations results in complete derived grammars, $\mathscr{G}_M{}'$. Finally, in subsection 4.5.4, we use this method to prove that certain transformations are valid for all Motion Grammars, forming a calculus that can be used to transform a Motion Grammar to a *correct* Motion Grammar.

4.5.2.1 Example of Complete and Incomplete Derivations

To illustrate the types of transformations which are and are not possible, consider the trivial system in 56(a). Here, a ball is dropped from height $x_0 = 1$. Its collision with the ground is perfectly inelastic, so it will stop as soon as it reaches $x = 0$. An initial grammar describing this system is shown in 56(b). From inspection, we can remove the expansion for $\langle \mathrm{H} \rangle$ and still have the valid grammar $\mathscr{G}_a'$ in 56(c). This is because the initial region for $\langle \mathrm{H} \rangle$, $x \geq 2$, is unreachable from the production for $\langle \mathrm{S} \rangle$. However, if we remove the production for $\langle \mathrm{G} \rangle$ as well, 56(d), then we no longer have a valid grammar. This is because $\mathscr{G}$ and $\mathscr{G}_b'$ are describing different sets of paths. Namely, $\mathscr{G}_b'$ takes a single path – tunneling to the center of the Earth – while $\mathscr{G}$ stops at $x = 0$. Thus, we see that there must be a specific relation



(a) System

$$\begin{aligned}\langle \mathrm{S}\rangle &\rightarrow [\mathrm{x}=1]\,\{\dot{x}=-mg\}\,\langle \mathrm{A}\rangle\\ \langle \mathrm{A}\rangle &\rightarrow \langle \mathrm{G}\rangle|\langle \mathrm{H}\rangle\\ \langle \mathrm{G}\rangle &\rightarrow [\mathrm{x}=0]\,\{\dot{x}=0\}\\ \langle \mathrm{H}\rangle &\rightarrow [\mathrm{x}\geq 2]\,\{\dot{x}=-mg\}\end{aligned}$$

(b) Initial Grammar, $\mathscr{G}$

$$\begin{aligned}\langle \mathrm{S}\rangle &\rightarrow [\mathrm{x}=1]\,\{\dot{x}=-mg\}\,\langle \mathrm{G}\rangle\\ \langle \mathrm{G}\rangle &\rightarrow [\mathrm{x}=0]\,\{\dot{x}=0\}\end{aligned}$$

(c) Simulating Grammar, $\mathscr{G}'_a$

$$\langle \mathrm{S}\rangle \rightarrow [\mathrm{x}=1]\,\{\dot{x}=-mg\}$$

(d) Non-Simulating Grammar, $\mathscr{G}'_b$

Figure 56: A dropped ball with a perfectly-inelastic collision. Examples of simulating and nonsimulating grammars. $\mathscr{G} \preceq \mathscr{G}'_a, \mathscr{G} \npreceq \mathscr{G}'_b$

between initial and derived grammars in order to maintain a faithful representation of the system. This specific relation is the simulation, $\mathscr{G} \preceq \mathscr{G}'$.

4.5.3 Completeness and a Simulation Lemma

A *complete* derived system model G' represents everything the initial system G could do. Slightly more formally, G' describes the set of all *paths* that the initial system G could take. This property is given as the simulation $G \preceq G'$. The reverse may not hold. That is, G' may be able to take paths which G cannot. The concrete definition of a *path* depends on type of system we are dealing with. For discrete Transition Systems, a path is the sequence states and transitions the system takes. For continuous systems, a path is a trajectory though its state space. Simulation paths for a variety of systems are detailed in [74].

This simulation relations 14 and 14 show that $\mathscr{G}_M$ and $\mathscr{G}_M{}'$ follow the same path provided that $\mathscr{G}_M$ is given the same initial conditions and inputs as $\mathscr{G}_M{}'$. The initial conditions are the first token of the starting nonterminal. For example, if $\langle \mathrm{A}\rangle$ begins with token $[\mathrm{x} \in \mathscr{R}]$, the initial condition of $\langle \mathrm{A}\rangle$ is $\mathscr{R}$. It is critical that $\mathscr{G}_M$ and $\mathscr{G}_M{}'$ are given the same input. The input u the our only way to influence the path of the system to make it *correct*.

To shorten the notation, let $\mathscr{G}_M|_{u',x'_0}$ be Motion Grammar $\mathscr{G}_M$ subject to initial condition x'_0 and input u', and likewise for the language $\mathfrak{L}_{\mathscr{G}_M|u',x'_0}$. We merge the simulation and correctness definitions as follows:

Lemma 4. $\mathscr{G}_M \preceq_c \mathscr{G}_M{}' \wedge u(t) = u'(t) \wedge x_0 = x'_0 \wedge \mathsf{correct}\left\{\mathscr{G}_M{}', \mathfrak{L}'_{\mathsf{s}}\right\} \implies \mathsf{correct}\left\{\mathscr{G}_M|_{\mathsf{x}'_0,\mathsf{u}'}, \mathfrak{L}'_{\mathsf{s}}\right\}$, *where* $\mathfrak{L}'_s, \mathfrak{L}'_{\mathscr{G}_M}, \mathfrak{L}'_{\mathscr{G}_M|u',x'_0} \subseteq Z'^*$.

Proof. From Def. 14, $\mathscr{G}_M \preceq_c \mathscr{G}_M{}' \wedge u(t) = u'(t) \wedge x_0 = x'_0 \implies x(t) = x'(t)$. From Def. 19, $x(t) = x'(t) \implies \mathfrak{L}'_{\mathscr{G}_M} = \mathfrak{L}'_{\mathscr{G}_M|u',x'_0}$. From Def. 17 and Def. 14, $\mathfrak{L}'_{\mathscr{G}_M} = \mathfrak{L}'_{\mathscr{G}_M|u',x'_0} \wedge \mathsf{correct}\left\{\mathscr{G}_M, \mathfrak{L}'_{\mathsf{s}}\right\} \implies \mathsf{correct}\left\{\mathscr{G}_M|_{\mathsf{x}'_0,\mathsf{u}'}, \mathfrak{L}'_{\mathsf{s}}\right\}$ □

From this, we can determine allowable derivations based on simulation $\preceq_c$, initial state x_0, and input u. Note that since derivations need not preserve the discrete token set Z, we must specify the correctness language $\mathfrak{L}'_s$ over token set Z'. With 4, we can now identify a set of symbolic transformations to apply to any Motion Grammar.

4.5.4 Rewrite Rules

To derive grammars for safe, Context-Free systems, we introduce a calculus of symbolic transformation rules for constructing a correct hybrid controller. This process begins with some initial model the hybrid system. The rules then *rewrite* the model step-by-step, always adhering to the simulation relation and 4 so that the correctness of the derived model implies correctness for our system. Thus we effectively change the system language until it satisfies our specification. Through this derivation process, we modify the behavior of the system to make it *correct*.

In each rule, we start with some grammar $\mathscr{G}_M$ and derive a grammar $\mathscr{G}_M{}'$. A rule is valid only if $\mathsf{correct}\left\{\mathscr{G}_M{}', \mathfrak{L}'_{\mathsf{s}}\right\} \implies \mathsf{correct}\left\{\mathscr{G}_M|_{\mathsf{x}'_0,\mathsf{u}'}, \mathfrak{L}'_{\mathsf{s}}\right\}$, which we will prove using 4. In the notation for these rules, we specify some precondition on the structure of elements of the production set P, and then specify the resulting token set Z', nonterminal set V' and production set P'.

4.5.4.1 *Input*

First, consider the very simple transformation of specifying an input u to illustrate this process. When the continuous dynamics are in the form $\dot{x} = f_0(x,u)$, we can always specify an input u.

Transform 1. *Given $p = A \rightarrow \alpha f_0(x,u)\beta$, define $f(x) = f_0(x,g(x))$. Then the new production set is $P' = P - p \cup \{A \rightarrow \alpha f(x)\beta\}$.*

Proof. For this transform to be allowable, it must satisfy the preconditions of 4. Namely we must have $\mathscr{G}_M \preceq_c \mathscr{G}_M{}' \wedge u(t) = u'(t) \wedge x_0 = x_0'$. Using $\mathscr{G}_M{}'$ to control the system means our input is given by $u'(t)$. Since we do not change the start symbol S, the initial condition $x_0 = x_0'$. Finally, in the modified production p, $\dot{x}' = f(x) = f_0(x,g(x)) = f_0(x,u'(t))$, so $\dot{x}' = \dot{x}|_{u'}$. Thus, $x_0 = x_0' \wedge \dot{x}' = \dot{x}|_{u'} \implies x(t) = x'(t) \implies \mathscr{G}_M \preceq_c \mathscr{G}_M{}'$. Thus, we satisfy the preconditions of 4 and can use correct $\{\mathscr{G}_M{}', \mathfrak{L}_\mathsf{s}'\}$ to decide correct $\{\mathscr{G}_M, \mathfrak{L}_\mathsf{s}'\}$. $\square$

4.5.4.2 *Token Splitting*

A region represented by a token can be split into two regions, creating two new tokens. We then create new productions for these new regions.

Transform 2. *Given some $\zeta_0 = [\mathrm{x} \in \mathscr{R}_0] \in Z$, create tokens $\zeta_1 = [\mathrm{x} \in \mathscr{R}_1]$ and $\zeta_2 = [\mathrm{x} \in \mathscr{R}_2]$ such that $\mathscr{R}_1 \cup \mathscr{R}_2 = \mathscr{R}_0 \wedge \mathscr{R}_1 \cap \mathscr{R}_2 = \emptyset$ and update token set $Z' = Z - \zeta_0 \cup \{\zeta_1, \zeta_2\}$. The new nonterminal set is $V' = V \cup \{A_0, A_1, A_2, A_3, A_4\}$. The new production set is $P' = P -$ $\{(A \rightarrow \alpha_1 \zeta_0 \kappa \alpha_2) \in P\} \cup \{(A \rightarrow \alpha_1 A_0), (A_0 \rightarrow A_1 | A_2) \,:\, (A \rightarrow \alpha_1 \zeta_0 \kappa \alpha_2) \in P\} \cup \{(A_1 \rightarrow \zeta_1 \kappa A_3), (A_2 \rightarrow \zeta$ $\{(A_3 \rightarrow A_2 | \alpha_2), (A_4 \rightarrow A_1 | \alpha_2) \,:\, (A \rightarrow \alpha_1 \zeta_0 \kappa \alpha_2) \in P\}$.*

Proof. Following the form of the proof for Transform 1, we need to show that $x(t) = x'(t)$. Since we have not modified the continuous dynamics, we need only show that the discrete dynamics of $\mathscr{G}_M{}'$ permit the same paths as in $\mathscr{G}_M$. In $\mathscr{G}_M$, for some production $A \rightarrow \alpha_1 \zeta_0 \kappa \alpha_2$, the system may pass from the region of α_1 into $\mathscr{R}_0$ and then on to the region of α_2. When we split $\mathscr{R}_0$, there are six cases to consider: $\mathscr{R}_{\alpha_1} \rightarrow \mathscr{R}_1$, $\mathscr{R}_{\alpha_1} \rightarrow \mathscr{R}_2$,

$\mathscr{R}_1 \to \mathscr{R}_{\alpha_2}, \mathscr{R}_2 \to \mathscr{R}_{\alpha_2}, \mathscr{R}_1 \to \mathscr{R}_2, \mathscr{R}_2 \to \mathscr{R}_1$. These cases are handled respectively by the added productions $A_0 \to A_1$, $A_0 \to A_2$, $A_3 \to \alpha_2$, $A_4 \to \alpha_2$, $A_3 \to A_2$, and $A_4 \to A_1$. Thus all paths from $\mathscr{G}_M$ and matched by $\mathscr{G}_M{}'$, so $\mathscr{G}_M \preceq_c \mathscr{G}_M{}'$. □

4.5.4.3 Adjacency Pruning

If two regions in state space are not adjacent, then the system may not pass directly between them. Thus we can eliminate productions which allow this to happen.

Transform 3. *For $p_1 = A \to r_A \kappa_A B$, $B \to \beta_1 | \ldots | \beta_n$, if r_A is not adjacent to $\mathscr{R}_0(\beta_n)$ WLOG, then $P' = P - p_1 \cup \{A \to r_A \kappa_A B'\} \cup \{B' \to \beta_1 | \ldots | \beta_{n-1}\}$*

Proof. To show $x(t) = x'(t)$, we prove by contradiction. We can say that $x(t) = x'(t)$ if and only if the removed production is unreachable, that is, the system $\mathscr{G}_M$ will never pass from r_A to $\mathscr{R}_0(\beta_n)$. Now, assume $x(t) \neq x'(t)$. Then $\mathscr{G}_M$ must pass from r_A to $\mathscr{R}_0(\beta_n)$. Since these two regions are not adjacent, this is a contradiction. Thus, we must have $x(t) = x'(t)$, so $\mathscr{G}_M \preceq_c \mathscr{G}_M{}'$. □

4.5.4.4 Barrier Pruning

The continuous dynamics f provide information that may be used to remove grammar productions. Using Theorem 5, we can show whether the system following $\dot{x} = f(x)$ may cross into any of the regions specified in the grammar.

Transform 4. *For $p_1 = A \to r_A \kappa_A B$, $B \to \beta_1 | \ldots | \beta_n$, if WLOG $\mathscr{L}_f s(p) < 0$ for all p along the level set $s(x) = 0$ which borders regions r_A and $\mathscr{R}_0(\beta_n)$, then $P' = P - p_2$.*

Proof. To show $x(t) = x'(t)$, we prove by contradiction. We can say that $x(t) = x'(t)$ if and only if the removed production is unreachable, that is, the system $\mathscr{G}_M$ will never pass from r_A to $\mathscr{R}_0(\beta_n)$. Now, assume $x(t) \neq x'(t)$. Then $\mathscr{G}_M$ may pass from r_A to $\mathscr{R}_0(\beta_n)$. By Theorem 5 and $\mathscr{L}_f s(p) < 0 \forall p \in \{x : s(x) = 0\}$, this is a contradiction. Thus, we must have $x(t) = x'(t)$, so $\mathscr{G}_M \preceq_c \mathscr{G}_M{}'$. □

4.5.4.5 *Bounce Pruning*

If the system in moving from region r_1 to region r_2 will immediately reenter r_1, then we can eliminate productions showing that the system will pass through r_2 into some third region.

Transform 5. *Given productions $p_1 = A \to r_1 \kappa_A B$, $p_2 = B \to r_2 \kappa_B C$, and $p_3 = C \to r_1 \kappa_B \alpha$, if $\mathscr{L}_{\kappa_B} s_{21}(x) > 0 \, \forall x \in \{x : s_{21}(x) = 0\}$, then $P' = P - p_1 - p_2 \cup \{(A \to r_1 \kappa_A B'), (B' \to r_2 \kappa_B r_1 \kappa_B \alpha)\}$*

Proof. To show $x(t) = x'(t)$, we must account for the removed productions by proving the system will not pass from r_1 to r_2 to some other region in $\mathscr{R}_0(C) - r_1$. Instead, the system must immediately return to r_1 from r_2. This is given directly by Theorem 5 and the Lie derivative $\mathscr{L}_{\kappa_B} s_{21}(x) > 0 \, \forall x \in \{x : s_{21}(x) = 0\}$, indicating that under mode κ_B this system will move off $s_{12}(x) = -s_{21}(x) = 0$ in the direction of r_1. Thus, p_2 will expand nonterminal C with p_3, according to the sequence given by the additional productions in P'. $\square$

4.5.5 Using the Calculus to Enforce Correctness

These rules provide important capabilities to work with hybrid models. Transform 1 allows us to specify the input to the robot to drive toward desired tokens. Transform 2 allows us to introduce new surfaces where we can discretely switch control inputs. Transform 3, Transform 4, and Transform 5 allow us to *remove* productions from the grammar. We can use this to satisfy a correctness constraint by eliminating certain bad productions causing the constraint violation. Thus, we can systematically produce a grammatical model implementing correct operation.

4.5.6 Safe Regions and New Switching Surfaces

Using our conservative reachability test from Theorem 5 and the rewrite rules of subsection 4.5.4, we can identify safe operating regions and consequently switching surfaces to maintain safe operation. Consider the example in Fig. 57 where there is some region with *good* and *bad* exit boundaries. Here, we wish to ensure that the system will cross the *good*

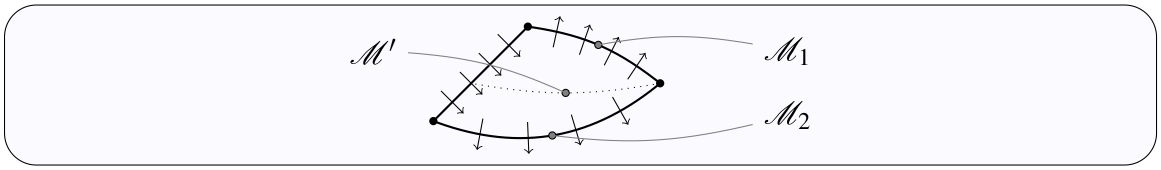


Figure 57: Splitting the region based on $\dot{x} = f(x)$ at each boundary.

manifold bounding $\mathscr{R}_0$, $\mathscr{M}_1$, and that it will not cross the other *bad* other manifold $\mathscr{M}_2$. Assume the continuous dynamics in $\mathscr{R}_0$ are $\dot{x} = f_0(x,u)$ and we have some controller $u = g(x)$, so that we can write the resulting system as the smooth function $\dot{x} = f_0(x, g(x)) = f(x)$. Then we can consider the Lie derivatives along each manifold $\mathscr{M}_1$ and $\mathscr{M}_2$, $\mathscr{L}_f s_1$ and $\mathscr{L}_f s_2$ respectively. If both $\mathscr{L}_f s_1$ and $\mathscr{L}_f s_2$ are positive, then for x close to $\mathscr{M}_1$ and $\mathscr{M}_2$, the system will cross both the safe and the unsafe boundaries. To indicate the safe subregion of $\mathscr{R}_0$, we introduce a new boundary $\mathscr{M}'$ separating $\mathscr{M}_1$ and $\mathscr{M}_2$. $\mathscr{M}'$ bounds the *region of inevitable collision*. By always staying to one side of $\mathscr{M}'$, we can be assured that by applying input function $u = g(x)$, we will not cross the unsafe boundary.

For this boundary notion to be useful during the online control of the system, we must be able to quickly test during each control cycle on which side of the boundary the system currently resides. If we express the manifold as $\mathscr{M}' = \{x : s(x) = 0\}$, then the sign of $s(x)$ will give the current side of the manifold. Thus, we must find some representation for $s(x)$ to evaluate in our control program.

We can describe the boundary between safe and unsafe regions based on the idea that boundary $\mathscr{M}'$ is composed of the family of integral curves leading to the intersection of the safe and unsafe exit surfaces. That is, for every point along $\mathscr{M}'$, the system dynamics $\dot{x} = f(x)$ will drive x to the intersection of the two exit surfaces $\mathscr{M}_1$ and $\mathscr{M}_2$. This description leads to the following two geometric properties of $\mathscr{M}' = \{x : s(x) = 0\}$. First, the gradient of $s(x)$ is orthogonal to the system vector field $f(x)$. Second, the gradient of $s(x)$ is orthogonal to the intersection $\mathscr{M}_C$ of our two exit surfaces $\mathscr{M}_1$ and $\mathscr{M}_2$.

Theorem 6. $\nabla s(x) \perp f(x)$

Proof. Consider some point p on $\mathscr{M}'$, $s(p)=0$. Point p moves according to $\dot{p}=f(p)$, and $\frac{d}{dt}s(p)=0$. $\frac{d}{dt}s(p)=\frac{\partial s}{\partial p}^T\frac{dp}{dt}=(\nabla s)(f)=0$. Since the inner product of $\nabla s(x)$ and $f(x)$ is always zero, they must be orthogonal. $\square$

Theorem 7. *Let* $\mathscr{M}_c=\left\{x:x=p(v),v\in\mathbb{R}^{n-2}\right\}$, *then* $\nabla s(x)\perp\frac{\partial p}{\partial v_i}$.

Proof. Since $\mathscr{M}_c\subset\mathscr{M}'$, $\frac{\partial s(p(v))}{\partial v_i}=0$. Then $\frac{\partial s(p(v))}{\partial v_i}=\frac{\partial s}{\partial p}^T\frac{\partial p}{\partial v_i}=(\nabla s)\left(\frac{\partial p}{\partial v}\right)$. $\square$

From Thm. 6 and 7, we can derive ∇s by finding the vector that satisfies the two orthogonal relations. For any vector ξ, we can find a vector ξ' orthogonal to $\psi=f(x)$ or $\psi=\frac{\partial p}{\partial v}$ using a projection.

$$\xi'=\xi-\mathsf{proj}_{\psi}\xi=\xi-\frac{\langle\xi,\psi\rangle}{\|\psi\|^2}\psi \tag{27}$$

Thus, we form ∇s by symbolically applying the Gram-Schmidt procedure.

$$\nabla s=\ell-\mathsf{proj}_{f(x)}\ell-\mathsf{proj}_{\frac{\partial p}{\partial v_1}}\ell-\ldots-\mathsf{proj}_{\frac{\partial p}{\partial v_{n-2}}}\ell \tag{28}$$

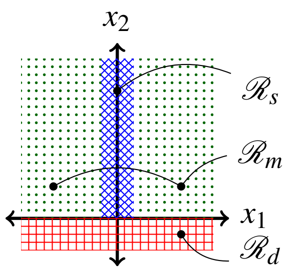
Equation (28) is useful when we can express ∇p in a form that does not include any v. For example, when $\mathscr{M}_c$ is linear, $p(v)=Mv$ and $\nabla p=M$. Then, with ∇s known, we can solve the following initial value problem to find $s(x)$,

$$s_0=g(0)\qquad\frac{\partial s}{\partial x_i}=(\nabla s)_i \tag{29}$$

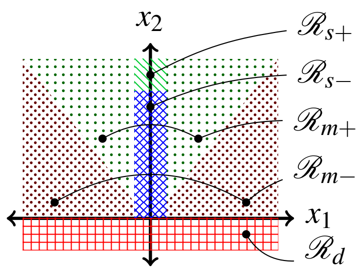
While (28) provides an analytic form for the gradient, solving (29) is nontrivial. Approximations such as a Taylor series or Padé approximant can be directly computed from (28).

4.5.7 Example Derivation

We now demonstrate this derivation approach with a simple example. Consider a mobile robot moving in one dimension, x_1, with a battery, x_2, that discharges as it moves. There is a recharging station at the zero position. When the battery level falls to zero, the robot can no longer operate. The continuous state space and initial grammar are shown in 58(a). The initial grammar for this system is given in 58(c).



(a) Initial Regions



(b) Derived Regions

$$\langle \mathrm{S} \rangle \rightarrow [\mathrm{s}] \left\{ \dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T \right\} \langle \mathrm{M} \rangle \quad (30)$$

$$\langle \mathrm{M} \rangle \rightarrow [\mathrm{m}] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{N} \rangle \quad (31)$$

$$\langle \mathrm{N} \rangle \rightarrow \langle \mathrm{S} \rangle \,|\, \langle \mathrm{D} \rangle \quad (32)$$

$$\langle \mathrm{D} \rangle \rightarrow [\mathrm{d}] \{ \dot{x} = 0 \} \quad (33)$$

(c) Initial Grammar

Figure 58: Initial grammar for 1-dimensional battery robot.

Because we want the robot to keep operating, its battery should never run down. This constraint is expressed in LTL:

$$G_s = \square (\neg [\mathrm{d}]) \quad (34)$$

The initial grammar does not satisfy (34). For example, the grammar generates the string [s] [m] [d], which violates the constraint. Thus, we must apply our transformations to the grammar in order to make it correct.

There are two main ideas to satisfying (34). First, the robot must not go far too from the charger, ensuring that it has enough charge to return. Second, the robot must wait in the charger to recharge. We apply these ideas in the derivation:

1. In (31), apply Transform 2 to split [m]. See new regions in 58(b).

$$\langle \mathrm{M} \rangle \rightarrow \cancel{[\mathrm{m}] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{N} \rangle} \tag{35}$$

$$\quad | \quad \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \tag{36}$$

$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{37}$$

$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_4 \rangle \tag{38}$$

$$\langle \mathrm{M}_3 \rangle \rightarrow \langle \mathrm{M}_2 \rangle | \langle \mathrm{N} \rangle \tag{39}$$

$$\langle \mathrm{M}_4 \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{N} \rangle \tag{40}$$

2. Duplicate (37) and replace in (40).

$$\langle \mathrm{M} \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \tag{41}$$

$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{42}$$

$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_4 \rangle \tag{43}$$

$$\langle \mathrm{M}_3 \rangle \rightarrow \langle \mathrm{M}_2 \rangle | \langle \mathrm{N} \rangle \tag{44}$$

$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{45}$$

$$\langle \mathrm{M}_4 \rangle \rightarrow \cancel{\langle \mathrm{M}_1 \rangle} \langle \mathrm{M}'_1 \rangle | \langle \mathrm{N} \rangle \tag{46}$$

3. In (43) and (45), apply Transform 1 to specify input $u = -x$.

$\dots$

$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = \begin{bmatrix} -x & -|x| \end{bmatrix}^T \right\} \langle \mathrm{M}_4 \rangle \tag{47}$$

$$\langle \mathrm{M}'_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} -x & -|x| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{48}$$

$\dots$

To simplify notation: $\kappa_- \equiv \left\{ \dot{x} = \begin{bmatrix} -x & -|x| \end{bmatrix}^T \right\}$.

4. Consider the Lie derivative between $[\mathrm{m}+]$ and $[\mathrm{m}-]$ according to κ_-. $\mathscr{M}$: $-1.5x_1 + x_2 = \varepsilon$, $x = \begin{bmatrix} 1.5v & v+\varepsilon \end{bmatrix}^T$, $v > 0$, $\nabla s = \begin{bmatrix} -1.5 & 1 \end{bmatrix}^T$, $\mathscr{L}_{\kappa_-} s|_{\mathscr{M}} = 1.5^2 v + v + \varepsilon$, always positive. In (47), (46), (48), apply Transform 5,

$$\langle \mathrm{M} \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \tag{49}$$
$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{50}$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_4 \rangle \tag{51}$$
$$\langle \mathrm{M}_3 \rangle \rightarrow \cancel{\langle \mathrm{M}_2 \rangle} \langle \mathrm{M}_2' \rangle | \langle \mathrm{N} \rangle \tag{52}$$
$$\langle \mathrm{M}_1' \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{53}$$
$$\langle \mathrm{M}_4 \rangle \rightarrow \langle \mathrm{M}_1' \rangle | \langle \mathrm{N} \rangle \tag{54}$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = \begin{bmatrix} -x & -|x| \end{bmatrix}^T \right\} \langle \mathrm{M}_4 \rangle \tag{55}$$
$$\langle \mathrm{M}_1' \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} -x & -|x| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{56}$$
$$\langle \mathrm{M}_2' \rangle \rightarrow [\mathrm{m}-]\, \kappa_-\, [\mathrm{m}+]\, \kappa_-\, \langle \mathrm{M}_3 \rangle \tag{57}$$

Thus, when the system moves from [m+] to [m−], it will switch to mode κ_- to return to the charging station.

5. Apply Transform 3 and Transform 4.

$$\langle \mathrm{M} \rangle \rightarrow \langle \mathrm{M}_1 \rangle | \langle \mathrm{M}_2 \rangle \tag{58}$$
$$\langle \mathrm{M}_1 \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \cancel{\langle \mathrm{M}_3 \rangle} \langle \mathrm{M}_3' \rangle \tag{59}$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_4 \rangle \tag{60}$$
$$\langle \mathrm{M}_3 \rangle \rightarrow \langle \mathrm{M}_2' \rangle | \langle \mathrm{N} \rangle \tag{61}$$
$$\langle \mathrm{M}_3' \rangle \rightarrow \langle \mathrm{M}_2' \rangle | \langle \mathrm{S} \rangle \tag{62}$$
$$\langle \mathrm{M}_1' \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \cancel{\langle \mathrm{M}_3 \rangle} \langle \mathrm{M}_3' \rangle \tag{63}$$
$$\langle \mathrm{M}_4 \rangle \rightarrow \langle \mathrm{M}_1' \rangle | \langle \mathrm{N} \rangle \tag{64}$$
$$\langle \mathrm{M}_2 \rangle \rightarrow [\mathrm{m}-] \left\{ \dot{x} = \begin{bmatrix} -x & -|x| \end{bmatrix}^T \right\} \langle \mathrm{M}_4 \rangle \tag{65}$$
$$\langle \mathrm{M}_1' \rangle \rightarrow [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} -x & -|x| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle \tag{66}$$
$$\langle \mathrm{M}_2' \rangle \rightarrow [\mathrm{m}-]\, \kappa_-\, [\mathrm{m}+]\, \kappa_-\, \cancel{\langle \mathrm{M}_3 \rangle} \langle \mathrm{S} \rangle \tag{67}$$

6. Switch now to (30) and split [s].

$$\langle \mathrm{S} \rangle \rightarrow \cancel{[\mathrm{s}]\left\{\dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T\right\}\langle \mathrm{M} \rangle} \quad (68)$$

$$\mid \quad S\langle \mathrm{S}_1 \rangle | \langle \mathrm{S}_2 \rangle \quad (69)$$

$$\langle \mathrm{S}_1 \rangle \rightarrow [\mathrm{s}+]\left\{\dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T\right\}\langle \mathrm{S}_3 \rangle \quad (70)$$

$$\langle \mathrm{S}_2 \rangle \rightarrow [\mathrm{s}-]\left\{\dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T\right\}\langle \mathrm{S}_4 \rangle \quad (71)$$

$$\langle \mathrm{S}_3 \rangle \rightarrow \langle \mathrm{S}_2 \rangle | \langle \mathrm{M} \rangle \quad (72)$$

$$\langle \mathrm{S}_4 \rangle \rightarrow \langle \mathrm{S}_1 \rangle | \langle \mathrm{M} \rangle \quad (73)$$

7. Give input $u = -x$.

$$\langle \mathrm{S} \rangle \rightarrow \langle \mathrm{S}_1 \rangle | \langle \mathrm{S}_2 \rangle \quad (74)$$

$$\langle \mathrm{S}_1 \rangle \rightarrow [\mathrm{s}+]\left\{\dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T\right\}\langle \mathrm{S}_3 \rangle \quad (75)$$

$$\langle \mathrm{S}_2 \rangle \rightarrow [\mathrm{s}-]\left\{\dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T\right\}\langle \mathrm{S}_4 \rangle \quad (76)$$

$$\langle \mathrm{S}_3 \rangle \rightarrow \langle \mathrm{S}_2 \rangle | \langle \mathrm{M} \rangle \quad (77)$$

$$\langle \mathrm{S}_4 \rangle \rightarrow \langle \mathrm{S}_1 \rangle | \langle \mathrm{M} \rangle \quad (78)$$

8. Consider Lie derivative between [s−] and [m] according to $\left\{\dot{x} = \begin{bmatrix} -x & k_s \end{bmatrix}^T\right\}$:

   - $\mathscr{M}$: $s(x) = x_1 = \pm\varepsilon$, $x = \begin{bmatrix} \pm\varepsilon & v \end{bmatrix}^T$, $v > 0$
   - $\nabla s = \begin{bmatrix} \pm 1 & 0 \end{bmatrix}^T$
   - $\mathscr{L}_{\kappa_-} s|_{\mathscr{M}} = -\varepsilon$ always negative

   Consider Lie derivative between [s+] and [s−] according to $\left\{\dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T\right\}$:

   - $\mathscr{M}$: $s(x) = -x_2 = -k$, $x = \begin{bmatrix} v & -k \end{bmatrix}^T$, $|v| < \varepsilon$
   - $\nabla s = \begin{bmatrix} \pm 0 & -1 \end{bmatrix}^T$
   - $\mathscr{L}_{\kappa_-} s|_{\mathscr{M}} = -k_s$ always negative

9. Apply Transform 4.

$$\langle \mathrm{S} \rangle \rightarrow \langle \mathrm{S}_1 \rangle | \langle \mathrm{S}_2 \rangle \quad (79)$$

$$\langle \mathrm{S}_1 \rangle \rightarrow [\mathrm{s}+]\left\{\dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T\right\}\cancel{\langle \mathrm{S}_3 \rangle}\langle \mathrm{M} \rangle \quad (80)$$

$$\langle \mathrm{S}_2 \rangle \rightarrow [\mathrm{s}-]\left\{\dot{x} = \begin{bmatrix} -x & k_s \end{bmatrix}^T\right\}\cancel{\langle \mathrm{S}_4 \rangle}\langle \mathrm{S}_1 \rangle \quad (81)$$

10. Combine $\langle \mathrm{S} \rangle$ and $\langle \mathrm{M} \rangle$.

$$\begin{array}{rcll}
\langle \mathrm{S} \rangle & \to & \langle \mathrm{S}_1 \rangle | \langle \mathrm{S}_2 \rangle & (82) \\
\langle \mathrm{S}_1 \rangle & \to & [\mathrm{s}+] \left\{ \dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T \right\} \langle \mathrm{M} \rangle & (83) \\
\langle \mathrm{S}_2 \rangle & \to & [\mathrm{s}-] \left\{ \dot{x} = \begin{bmatrix} -x & k_s \end{bmatrix}^T \right\} \langle \mathrm{S}_1 \rangle & (84) \\
\langle \mathrm{M} \rangle & \to & \langle \mathrm{M}_1 \rangle | \mathrm{M}_2 & (85) \\
\langle \mathrm{M}_1 \rangle & \to & [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_3 \rangle' & (86) \\
\langle \mathrm{M}_2 \rangle & \to & [\mathrm{m}-]\, \kappa_- \langle \mathrm{M}_4 \rangle \ldots & (87)
\end{array}$$

11. Apply Transform 3.

$$\ldots \langle \mathrm{M} \rangle \quad \to \quad \langle \mathrm{M}_1 \rangle \cancel{| \langle \mathrm{M}_2 \rangle} \ldots \qquad (88)$$

12. Remove unreferenced and singleton productions.

$$\boxed{\begin{array}{rcl}
\langle \mathrm{S} \rangle & \to & \langle \mathrm{S}_1 \rangle | \langle \mathrm{S}_2 \rangle \\
\langle \mathrm{S}_1 \rangle & \to & [\mathrm{s}+] \left\{ \dot{x} = \begin{bmatrix} u & k_s \end{bmatrix}^T \right\} \langle \mathrm{M}_1 \rangle \\
\langle \mathrm{S}_2 \rangle & \to & [\mathrm{s}-] \left\{ \dot{x} = \begin{bmatrix} -x & k_s \end{bmatrix}^T \right\} \langle \mathrm{S}_1 \rangle \\
\langle \mathrm{M}_1 \rangle & \to & [\mathrm{m}+] \left\{ \dot{x} = \begin{bmatrix} u & -|u| \end{bmatrix}^T \right\} \langle \mathrm{M}_3' \rangle \\
\langle \mathrm{M}_3' \rangle & \to & \langle \mathrm{M}_2' \rangle | \langle \mathrm{S} \rangle \\
\langle \mathrm{M}_2' \rangle & \to & [\mathrm{m}-]\, \kappa_-\, [\mathrm{m}+]\, \kappa_- \langle \mathrm{S} \rangle
\end{array}}$$

We have derived a grammar which guarantees the robot will never discharge. Note that within the production for $\langle \mathrm{M} \rangle$, we have produced a safe operating region as given in subsection 4.5.6.

4.6 Composing Mapping and Manipulation

In this section, we combine semantic maps with hybrid control models, generating a direct link between action and environment models to produce a control policy for mobile manipulation in unstructured environments. First, we generate a semantic map for our environment and design a base model of robot action. Then, we combine this map and action model using the Motion Grammar Calculus to produce a combined robot-environment model. Using this combined model, we apply supervisory control to produce a policy for the manipulation task. We demonstrate this approach on a Segway RMP-200 mobile platform

Semantic mapping and hybrid control are both effective approach within robotics. Semantic mapping produces detailed models of *unstructured environments* [134, 139, 175, 178, 179]; however, this approach provides no direct link to robot action. Hybrid models combine continuous and discrete robot dynamics to efficiently and verifiably represent *robot action* [6, 14, 42, 43, 45, 81]; however, there is no automatic method to produce control models for large, complicated *systems*. While superficially, it appears that semantic mapping and hybrid control are fundamentally different approaches, they are actually closely related. The topological graph of a semantic map and the discrete event system of a hybrid control model are both instances of *formal language*. Thus, we propose to combine the linguistic representations of semantic maps and robot action models to produce an efficient and verifiable control policy for mobile manipulation in unstructured environments.

This work focuses on the application domain of service robots in human environments. Previously, we developed new techniques for mapping using Semantic SLAM [134, 179] and for hybrid systems using our Motion Grammar [42, 43, 45]. Here, we integrate these approaches to produce a combined robot-environment action model. Then, we apply established methods in supervisory control [25] to derive a robot control policy for a mobile manipulation task. This control design approach formally guarantees that the resultant policy satisfies the task specification. Finally, we demonstrate of this approach on a Segway RMP-200 mobile robot.

Simultaneous Localization and Mapping (SLAM) is the concurrent pose estimation of both the robot and objects in its environment. This is a well studied area with many useful results. Smith and Cheeseman [163] proposed one of the first solutions to the SLAM problem using the Extended Kalman Filter (EKF) to jointly represent the landmark positions along with the robot pose. Folkesson and Christensen developed GraphSLAM [67], an efficient solution to the SLAM problem which preserves landmark independence and is able to find loop closures through nonlinear optimization. Semantic SLAM augments a map with semantically relevant object labels. In this work, we utilize the Semantic SLAM method of Trevor and Nieto [134, 178, 179] to compose a map and hybrid controller.

There are several related techniques and alternative approaches for the service robotics domain. Topp and Christensen, [175, 176], provide a separation of regions relating to a user's view on the environment and detection of transitions between them. O'Callaghan [139] developed a new statistical modeling technique for building occupancy maps by providing both a continuous representation of the robot's surroundings and an associated predictive variance employing a Gaussian process and Bayesian learning. In this work we focus on integrating robot mapping with hybrid control methods. The notion of affordances originated in Psychology [71] to describe interaction between agents and environments and has previously provided inspiration for robotics research [155]. We rather focus our approach on direct symbol manipulation techniques with clear algorithmic implementation.

Simultaneous Localization and Mapping (SLAM) is the concurrent execution of both *Localization* and *Mapping* on a robot. *Localization* means determining the current position of the robot based on observations. *Mapping* means determining the positions of objects in the environment based on observations. Typical SLAM implementations combine odometry and other geometric measurements such as point clouds or camera features to simultaneously produce an estimate of the position of both the robot and objects. Using this technique, the robot models unstructured environments.

We use an existing mapping system to identify surfaces and connected free spaces in the world [178, 179]. We use the surfaces, such as walls and tables, to localize the robot based on its relative position to these object. We represent free spaces as Gaussian regions in $\mathbb{R}^3$ with mean at the center of the free space and standard deviation indicating the dimensions of the free space [134]. Topological connections between these Gaussian regions indicate connected free spaces in the environment. For example, a door or hallway between two rooms would connect the Gaussian regions for those rooms.

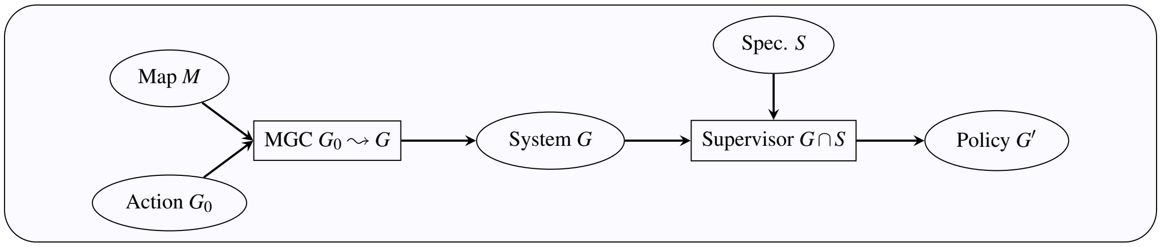
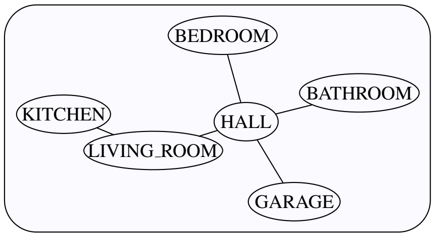


Figure 59: Sequence of operations to generate policy.



(a) Semantic Map M

$$
\begin{array}{rcl}
\langle \mathrm{S} \rangle & \to & [\mathrm{room}]\,\langle \mathrm{S} \rangle \\
 & | & [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{S}' \rangle \\
\langle \mathrm{S}' \rangle & \to & [\mathrm{room}]\,\langle \mathrm{S}' \rangle \\
 & | & [\mathrm{place}]\,\langle \mathrm{S} \rangle
\end{array}
$$

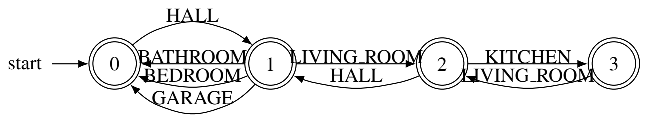
(b) Base Grammar G_0

Figure 60: Example of Semantic Map M and base manipulation grammar G_0. This map represents the Georgia Tech Aware Home.

We then extend the metric and topological information of the map surfaces and connected Gaussians with additional semantic information by labeling each of the Gaussian regions. These *Semantic Maps* provide useful information for navigation and localization of the robot. In addition, the semantic content of the map permits higher-level reasoning about the spatial regions of the environment. We exploit this semantic information in our composition of the map with a grammar for robot action.

4.6.1 Composing Maps and Grammars

We produce the control policy for the robot by composing a semantic map and a base action grammar, following Fig. 59. We will explain this approach using the example map for the Georgia Tech Aware home, 60(a), and the base grammar for mobile manipulation, 60(b). First, we convert the map graph into a grammar for the map language. Then, we compose the map grammar and the action grammar using the Motion Grammar Calculus (MGC) to model the robotic system operating within the mapped environment. Finally, we produce a task policy by applying a supervisory controller to this system model.



(a) Map Automaton

$$
\begin{array}{lcl}
\langle 0\rangle & \rightarrow & [\text{hall}]\,\langle 1\rangle \\
\langle 1\rangle & \rightarrow & [\text{bathroom}]\,\langle 0\rangle \\
 & | & [\text{bedroom}]\,\langle 0\rangle \\
 & | & [\text{garage}]\,\langle 0\rangle \\
 & | & [\text{livingroom}]\,\langle 2\rangle \\
\langle 2\rangle & \rightarrow & [\text{hall}]\,\langle 1\rangle \\
 & | & [\text{kitchen}]\,\langle 3\rangle \\
\langle 3\rangle & \rightarrow & [\text{livingroom}]\,\langle 2\rangle
\end{array}
$$

(b) Map Grammar

Figure 61: Representing maps with formal language.

To better analyze the semantic map, we first represent this map using formal language. The Gaussian free space regions of the map are arranged in a graph, indicating connectivity between these regions. The graph for the Aware Home is 60(a). This graph is equivalent to a Regular Language representing the set of all traces through the map.

Definition 20. *Let Map $M = (N,V)$, where N is a finite set of location symbols, and $V \subset N \times N$ is the set of adjacent symbols $n_i \to n_j$.*

We can transform any Map M into a regular grammar. We note that when analyzing Finite Automata, the language symbols are typically given along transitions [4, 86] wheres in a map, location symbols mark a state. For regular languages, these two conventions – terminal language symbols on states and terminal language symbols on edges – are equivalent. algorithm 11 transforms the state terminal map to an edge terminal automaton. Then, we can directly convert this automaton to a Regular Grammar.

We demonstrate the conversion for the map in 60(a). First, we apply algorithm 11 to produce a FSM from the map graph. Since the output of this algorithm is a Nondeterminisic Finite Automaton with more than the minimum necessary number of states, we convert the NFA to a DFA [4, p152] and minimize the number of DFA states with Hopcroft's Algorithm [4, p180]. This result is 61(a). Note that in this example, we save two states over the original map in 60(a). Finally, we convert the FSM to the Regular grammar in 61(b).

Algorithm 11: State to Edge Symbols

```
   Input: $Q$ ;                                   // Initial States
   Input: $E : Q \times Q$ ;                      // Initial Edges
   Output: $Q'$ ;                                 // Final States
   Output: $Z'$ ;                                 // Edge Symbols
   Output: $E' : Q' \times Z' \times Q'$ ;        // Final Edges
1  $Z' = Q$;
2  $Q' = E$ ;
3  $E' = \emptyset$;
4  forall the $q \in Q$ do
5      forall the $(e_i = Q \to q) \in E$ do
6          forall the $(e_j = q \to Q) \in E$ do
7              $E' = E' \cup e_i \xrightarrow{q} e_j$
```

4.6.2 Composition using the Motion Grammar Calculus

In order to semantically merge the robot and environment models, we apply our Motion Grammar Calculus (MGC) rewrite rules. According to these rules, we extend our action grammar with each map symbol while maintaining only those transitions allowed by the map. While supervisory control can only operate to restrict system G using existing symbols, the MGC crucially describes how to introduce *new symbols* into G. There are two relevant rewrite rules from the MGC that we use here.

By applying these transforms, we can introduce the map symbols into the action grammar while preserving the validity of the model. Each derivation step maintains the *completeness* of the model according to the path of the hybrid system. By assuming that the initial model is complete, this ensures that all derived models are also complete.

In addition to these two transforms, we also use the $\mathsf{first}()$ and $\mathsf{follow}()$ sets [4] to define initial and adjacent symbols. The $\mathsf{first}()$ set defines all terminals which may begin some derivation of a grammar symbol. The $\mathsf{follow}()$ set defines all terminals which may appear immediately to the right of some symbol in a grammatical derivation [4][p221].

Definition 21 (First Set)**.** *Define* $\mathsf{first}(X)$ *for some grammar symbol X to be the set of terminals which may begin strings derived from X.*

Definition 22 (Follow Set)**.** *Define* $\mathsf{follow}(X)$ *for grammar symbol X to be the set of terminals a that can appear immediately to the right of X in some sentential form.*

Note that for map grammars such as 61(b), the follow set for each terminal symbol is equivalent to the adjacent nodes in the map graph 60(a).

Proposition 6. *Given a grammar G representing some map M,* $\mathsf{follow}(z)$ *of some terminal symbol z of G represents the set of all map locations adjacent to z.*

Algorithm 12 describes how we apply these transforms to compose the Map and Action grammars. First, we introduce all map symbols into the action grammar by repeatedly splitting the initial terminal symbol of the action grammar by direct application of Transform 2. Next, we prune out productions indicating transitions between non-adjacent map locations. To prune these productions, we apply Transform 3 by intersecting the grammar with sets of allowable transitions. The disallowed transitions are indicated by the regular expression $L = (.^*z_1 Z_A{}^* z_2.^*)$ in line 8 of algorithm 12. The complement of this regular expression defines all paths which do not move directly from z_1 to z_2. Since z_1 and z_2 are non-adjacent, intersecting with $\overline{L}$ will preserve only paths which do not contain the disallowed transition. The result is a grammar which contains the original action model and all permissible transitions from the semantic map.

We apply algorithm 12 to combine the map grammar, 61(b), with the base grammar for mobile manipulation, 60(b). In this process, the initial nonterminal of the base grammar, [room], is repeatedly split into all the symbols of the semantic map. Then all transitions between non-adjacent map symbols are pruned away. This produces the combined grammar of 67(a).

4.6.3 Supervisory Control

Finally, we use supervisory control to produce the policy G' from our system model G and task specification S, [25, p133]. This application of supervisory control will permit only

Algorithm 12: Composing Map and Action Grammars

```
   Input: $(Z_M, V_M, P_M, S_M)$ ;                                  // Map Grammar
   Input: $(Z_A, V_A, P_A, S_A)$ ;                                  // Action Grammar
   Output: $(Z, V, P, S)$ ;                                         // Combined Grammar
 1 $(Z,V,P,S) \leftarrow (Z_A, V_A, P_A, S_A)$ ;
   /* Add map symbols by splitting first(S_A)                             */
 2 $z_0 = \mathsf{first}(S_A)$;
 3 forall the $z \in Z_M$ do
 4     $(Z,V,P,S) \leftarrow$ Transform 2 to split $z_0$ into $z$ and $z_0$
   /* Prune non-adjacent map symbols                                      */
 5 forall the $z_1 \in Z_M$ do
 6     forall the $z_2 \in Z_M$ do
 7         if $z_2 \notin \mathsf{follow}(z_1)$ then
 8             $(Z,V,P,S) \leftarrow (Z,V,P,S) \cap \overline{\mathfrak{L}\{.^*z_1 Z_A{}^* z_2.^*\}}$ ;
```

those transitions of the model G which are also contained in specification S. We represent this as the intersection,

$$G' = G \cap S \tag{89}$$

Given that G is Context-Free and S is Regular, we use the algorithm defined in [86, p135] to produce Context-Free G', ensuring that we can efficiently execute the policy given by G'. This algorithm operates on a Context-Free language model for system G and a Regular language specification for correct operation S with the assumption that we can *block* any undesirable transitions in G. The corrected system language, then, is $G' = G \cap S$, where G' is also Context-Free. We note in addition that to prune non-adjacent regions permitted by Transform 3 in algorithm 12, we apply this same language intersection operation.

4.6.4 Mobile Manipulation Demonstration

We implemented this approach on a Segway RMP-200 mobile platform as shown in Fig. 62. This platform is equipped with an ASUS Xtion PRO LIVE camera, providing RGBD information for plane and surface extraction and with a UTM-30LX Hokuyo laser used to label the spatial regions as Gaussian models. It includes a Schunk parallel jaw gripper to

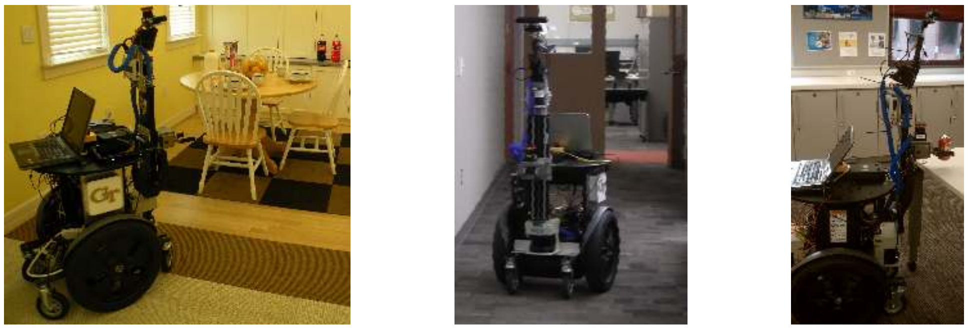


Figure 62: Segway RMP-200 mobile platform in the Georgia Tech Aware, the RIM Center, and picking a soda can.

manipulate objects. We conducted the experiments in the Georgia Tech Aware Home [102] and RIM center.

For both of the home and office environments, we first drove the robot through each area collecting 3D point clouds, laser, and odometry. Our mapper extracts planes and surfaces in the environment, building the map and localizing the robot. During the navigation, the robot partitions the environment into Gaussian regions. This produces the Gaussian map in Fig. 63. Then, we annotate the Gaussian regions of the map with semantic labels. The result is a graph, shown previously for the Aware Home in 60(a) and also for the RIM center in Fig. 64. This resulting map is suitable for both human interpretation and automatic symbol manipulation.

Next, we apply the method described in subsection 4.6.1 to generate the symbolic model for the robot in each of the environments. For the Aware home, this model is given in 67(a), and for the RIM center in Fig. 64. For the Aware Home, we asked the robot to peform the following task, *Collect a soda from the kitchen and bring it to the bedroom*, expressed as the specification in 67(b). For the RIM Center, we apply a similar supervisor in 65(b) to collect a soda from kitchen and bring it to library.

The policy for the task in the RIM environment, 65(c), is more complicated than for the Aware Home, 67(c). This is because the RIM map contains multiple paths between all rooms. Thus, all these possible paths are captured in the control policy grammar. The

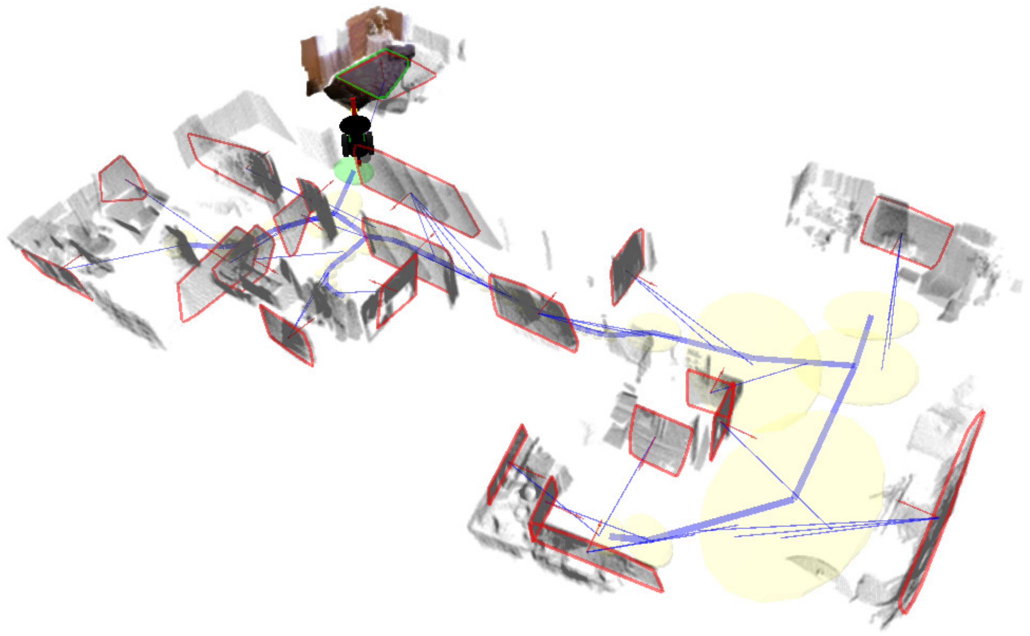
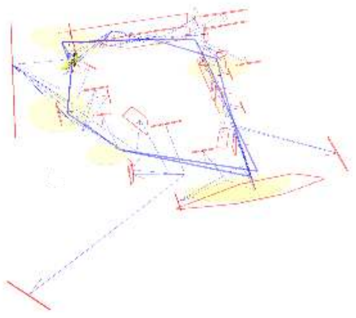
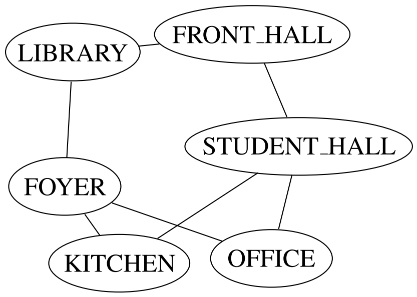


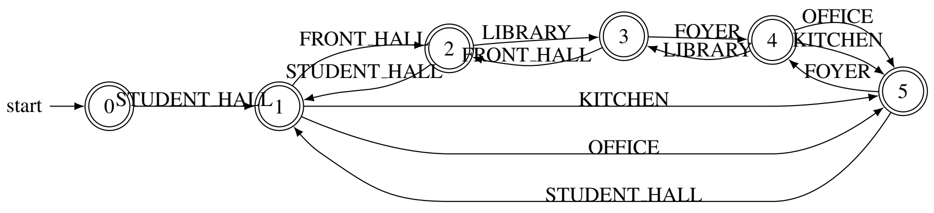
Figure 63: Generated Semantic Maps for the Aware Home. In the map, black shows 3D robot model, gray shows point clouds, yellow shows connected Gaussian regions (blue edges), and red shows the surfaces.



(a) RIM Map



(b) RIM Graph



(c) RIM FSM

Figure 64: Generated Semantic map of Georgia Tech RIM Center and the equivalent graph and Finite Automata forms.

$$
\begin{array}{rcl}
\langle \mathrm{S_0} \rangle & \rightarrow & [\mathrm{s}]\,\langle \mathrm{S} \rangle \\
\langle \mathrm{S} \rangle & \rightarrow & [\mathrm{r}]\,\langle \mathrm{R} \rangle \mid [\mathrm{o}]\,\langle \mathrm{O} \rangle \mid [\mathrm{k}]\,\langle \mathrm{K} \rangle \mid [\mathrm{pick}]\,\langle \mathrm{S'} \rangle \\
\langle \mathrm{O} \rangle & \rightarrow & [\mathrm{s}]\,\langle \mathrm{S} \rangle \mid [\mathrm{f}]\,\langle \mathrm{F} \rangle \mid [\mathrm{pick}]\,\langle \mathrm{O'} \rangle \\
\langle \mathrm{K} \rangle & \rightarrow & [\mathrm{s}]\,\langle \mathrm{S} \rangle \mid [\mathrm{f}]\,\langle \mathrm{F} \rangle \mid [\mathrm{pick}]\,\langle \mathrm{K'} \rangle \\
\langle \mathrm{F} \rangle & \rightarrow & [\mathrm{o}]\,\langle \mathrm{O} \rangle \mid [\mathrm{k}]\,\langle \mathrm{K} \rangle \mid [\mathrm{l}]\,\langle \mathrm{L} \rangle \mid [\mathrm{pick}]\,\langle \mathrm{F'} \rangle \\
\langle \mathrm{L} \rangle & \rightarrow & [\mathrm{f}]\,\langle \mathrm{F} \rangle \mid [\mathrm{r}]\,\langle \mathrm{R} \rangle \mid [\mathrm{pick}]\,\langle \mathrm{L'} \rangle \\
\langle \mathrm{R} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L} \rangle \mid [\mathrm{s}]\,\langle \mathrm{S} \rangle \mid [\mathrm{pick}]\,\langle \mathrm{R'} \rangle \\
\langle \mathrm{S'} \rangle & \rightarrow & [\mathrm{r}]\,\langle \mathrm{R'} \rangle \mid [\mathrm{o}]\,\langle \mathrm{O'} \rangle \mid [\mathrm{k}]\,\langle \mathrm{K'} \rangle \\
 & \mid & [\mathrm{place}]\,\langle \mathrm{S} \rangle \\
\langle \mathrm{O'} \rangle & \rightarrow & [\mathrm{s}]\,\langle \mathrm{S'} \rangle \mid [\mathrm{f}]\,\langle \mathrm{F'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{O} \rangle \\
\langle \mathrm{K'} \rangle & \rightarrow & [\mathrm{s}]\,\langle \mathrm{S'} \rangle \mid [\mathrm{f}]\,\langle \mathrm{F'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{K} \rangle \\
\langle \mathrm{F'} \rangle & \rightarrow & [\mathrm{o}]\,\langle \mathrm{O'} \rangle \mid [\mathrm{k}]\,\langle \mathrm{K'} \rangle \mid [\mathrm{l}]\,\langle \mathrm{L'} \rangle \\
 & \mid & [\mathrm{place}]\,\langle \mathrm{F} \rangle \\
\langle \mathrm{L'} \rangle & \rightarrow & [\mathrm{f}]\,\langle \mathrm{F'} \rangle \mid [\mathrm{r}]\,\langle \mathrm{R'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{L} \rangle \\
\langle \mathrm{R'} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L'} \rangle \mid [\mathrm{s}]\,\langle \mathrm{S'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{R} \rangle
\end{array}
$$

(a) Uncontrolled: G

- Let $\mathscr{R} = \{[\mathrm{s}], [\mathrm{k}], [\mathrm{o}], [\mathrm{f}], [\mathrm{l}], [\mathrm{r}]\}$
- Pick object in kitchen:
  $S_0 = .^*\,[\mathrm{k}]\,(\neg\mathscr{R})^*\,[\mathrm{pick}]\,.^*$
- Place object in library:
  $S_1 = .^*\,[\mathrm{l}]\,[\mathrm{place}]\,\$$
- Move the object only once:
  $S_2 = (\neg\,[\mathrm{place}])^*\,[\mathrm{place}]\,\neg([\mathrm{pick}])^*$
- Let $\mathscr{X} = (\neg\,[\mathrm{x}])^*\,[\mathrm{x}]\,(\neg\,[\mathrm{x}])^*$
- Don't revisit rooms:
  $S_3 = \bigcap_{[\mathrm{x}]\in\mathscr{R}} \mathscr{X}\,(([\mathrm{pick}] \mid [\mathrm{place}])\,\mathscr{X})^*$

(b) Supervisor: S

$$
\begin{array}{rcl}
\langle \mathrm{S_0} \rangle & \rightarrow & [\mathrm{s}]\,\langle \mathrm{S} \rangle \\
\langle \mathrm{S} \rangle & \rightarrow & [\mathrm{k}]\,\langle \mathrm{K} \rangle \mid [\mathrm{r}]\,\langle \mathrm{R} \rangle \mid [\mathrm{o}]\,\langle \mathrm{OL} \rangle \\
\langle \mathrm{K} \rangle & \rightarrow & [\mathrm{pick}]\,\langle \mathrm{K'} \rangle \\
\langle \mathrm{R} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{OL} \rangle \\
\langle \mathrm{OL} \rangle & \rightarrow & [\mathrm{f}]\,\langle \mathrm{F} \rangle \\
\langle \mathrm{F} \rangle & \rightarrow & [\mathrm{k}]\,\langle \mathrm{K} \rangle \\
\langle \mathrm{K'} \rangle & \rightarrow & [\mathrm{f}]\,\langle \mathrm{F'} \rangle \mid [\mathrm{s}]\,\langle \mathrm{S'} \rangle \\
\langle \mathrm{F'} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L'} \rangle \mid [\mathrm{o}]\,\left[\mathrm{O'}\right] \\
\langle \mathrm{S'} \rangle & \rightarrow & [\mathrm{r}]\,\langle \mathrm{R'} \rangle \\
\langle \mathrm{O'} \rangle & \rightarrow & [\mathrm{s}]\,\langle \mathrm{S'} \rangle \\
\langle \mathrm{R'} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L'} \rangle \\
\langle \mathrm{L'} \rangle & \rightarrow & [\mathrm{place}]
\end{array}
$$

(c) Controlled: G'

Figure 65: Grammars for the Uncontrolled and Controlled mobile manipulator in the RIM Center. Notice how the policy captures all possible paths through the environment that satisfy the specification.

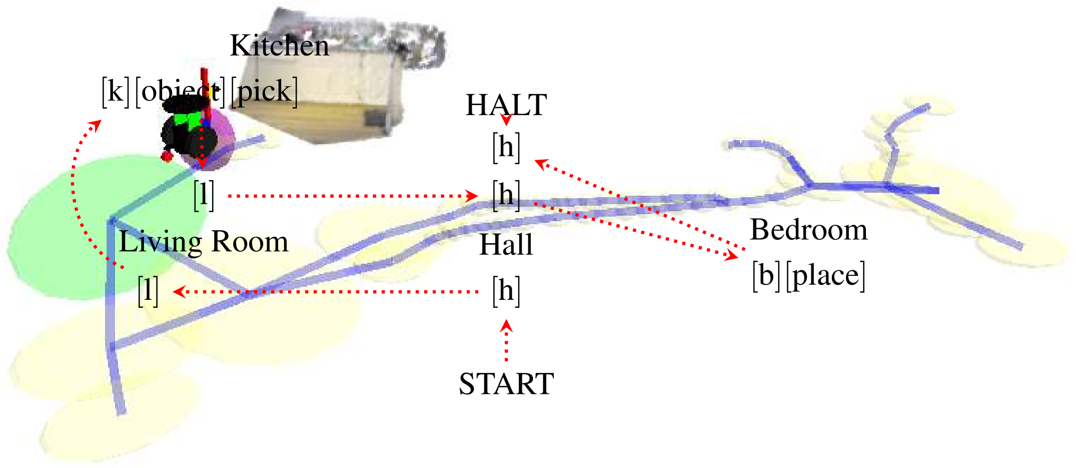


Figure 66: Path of the robot following controller in 67(c) and (91), shown as robot enters the living (green oval). Solid blue lines show the map connections between rooms, and dotted red lines show the robot path.

result is the nine strings represented by the following regular expression,

$$G' = (k|rlfk|olfk)\,[\text{pick}]\,(fl|fosrl|srl)\,[\text{place}] \tag{90}$$

These generated policies direct the robot along the path to complete the specified task. For the Aware Home, the robot fetches the object from the kitchen and delivers it to the bedroom, illustrated in Fig. 66. This figure shows the path of the robot, both as a trajectory though the map and as the sequence of language symbols.

4.6.5 Discussion

In this approach, we combine a Semantic Map and a Motion Grammar using the Motion Grammar Calculus (MGC). This ensures the validity of our final system model because each transform of the MGC preserves *completeness* of the model. Then, applying a supervisory controller guarantees that the final policy is *correct* with regard to the specification. Thus, the overall approach is *correct-by-construction* in the sense that the final system model is guaranteed by the MGC to simulate our initial system, and the resultant policy satisfies the supervisory control specification.

The defining characteristic of this method is the uniform representation of the set of all robot paths as a language with an explicit grammar. This representation allows iterative development of the grammatical control policy by the progressive application of MGC transformations and supervisory control specifications. At each step of this derivation, the mechanical application of the MGC transforms and supervisory control ensures that we maintain a valid model of the system. Furthermore, because the policy for each task is itself a grammar, we can compose multiple individual task policies to produce a system to perform each of those tasks, all within the same grammatical framework. We expect these capabilities for incremental design and policy composition to be useful as we extend our work to multiple tasks and more complicated systems with larger grammars.

While search-based motion planning could perform some of the tasks in this paper, there are certain advantages given by our linguistic formulation and use of supervisory control for policy generation. Random-sampling planners such as RRTs and PRMs assume a continuous search space, while our application domain includes discrete features for detecting and manipulating objects. General search based planning assumes an explicit goal state and produces a *plan* to reach that state. In contrast, the linguistic approach considers the set of acceptable *paths* and produces a *policy* to stay within that set of paths.

We use supervisory control of the grammar in 67(a) to perform the desired mobile manipulation task. To instruct the robot to bring an object from the kitchen to the human in the bedroom, we construct our supervisor according to the regular expressions in 67(b). Thus, our controlled system is,

$$G' = G \cap \bigcap_{i=0}^{4} S_i = \text{[h] [l] [k] [object] [pick] [l] [h] [b] [place] [h]} \tag{91}$$

$$
\begin{array}{rcl}
\langle \mathrm{S_0} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H} \rangle \\
\langle \mathrm{H} \rangle & \rightarrow & [\mathrm{r}]\,\langle \mathrm{R} \rangle \mid [\mathrm{b}]\,\langle \mathrm{B} \rangle \mid [\mathrm{o}]\,\langle \mathrm{O} \rangle \\
 & \mid & [\mathrm{d}]\,\langle \mathrm{D} \rangle \mid [\mathrm{l}]\,\langle \mathrm{L} \rangle \mid [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{H'} \rangle \\
\langle \mathrm{B} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H} \rangle \mid [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{B'} \rangle \\
\langle \mathrm{O} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H} \rangle \mid [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{O'} \rangle \\
\langle \mathrm{R} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H} \rangle \mid [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{R'} \rangle \\
\langle \mathrm{D} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H} \rangle \mid [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{D'} \rangle \\
\langle \mathrm{L} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H} \rangle \mid [\mathrm{k}]\,\langle \mathrm{K} \rangle \mid [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{L'} \rangle \\
\langle \mathrm{K} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L} \rangle \mid [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{K'} \rangle \\
\langle \mathrm{H'} \rangle & \rightarrow & [\mathrm{r}]\,\langle \mathrm{R'} \rangle \mid [\mathrm{b}]\,\langle \mathrm{B'} \rangle \mid [\mathrm{o}]\,\langle \mathrm{O'} \rangle \\
 & \mid & [\mathrm{d}]\,\langle \mathrm{D'} \rangle \mid [\mathrm{l}]\,\langle \mathrm{L'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{H} \rangle \\
\langle \mathrm{B'} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{B} \rangle \\
\langle \mathrm{O} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{O} \rangle \\
\langle \mathrm{R} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{R} \rangle \\
\langle \mathrm{D} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{D} \rangle \\
\langle \mathrm{L'} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H'} \rangle \mid [\mathrm{k}]\,\langle \mathrm{K'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{L} \rangle \\
\langle \mathrm{K'} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L'} \rangle \mid [\mathrm{place}]\,\langle \mathrm{K'} \rangle
\end{array}
$$

(a) Uncontrolled: G

- Let $\mathscr{R} = \{[\mathrm{h}], [\mathrm{r}], [\mathrm{o}], [\mathrm{d}], [\mathrm{l}]\}$
- Pick object in kitchen:
  $S_0 = .^*\,[\mathrm{k}]\,(\neg\mathscr{R})^*\,[\mathrm{pick}]\,.^*$
- Place object in bedroom:
  $S_1 = .^*\,[\mathrm{b}]\,[\mathrm{place}]\,.^*$
- Move the object only once:
  $S_2 = (\neg\,[\mathrm{place}])^*\,[\mathrm{place}]\,\neg([\mathrm{pick}])^*$
- Let $\mathscr{X} = (\neg\,[\mathrm{x}])^*\,[\mathrm{x}]\,(\neg\,[\mathrm{x}])^*$
- Don't revisit rooms:
  $S_3 = \bigcap_{[\mathrm{x}]\in\mathscr{R}} \mathscr{X}\,(([\mathrm{pick}] \mid [\mathrm{place}])\,\mathscr{X})^*$
- End in the hallway: $S_4 = .^*\,[\mathrm{h}]\,\$$

(b) Supervisor: S

$$
\begin{array}{rcl}
\langle \mathrm{S_0} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H} \rangle \\
\langle \mathrm{H} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L} \rangle \\
\langle \mathrm{L} \rangle & \rightarrow & [\mathrm{k}]\,\langle \mathrm{K} \rangle \\
\langle \mathrm{K} \rangle & \rightarrow & [\mathrm{object}]\,[\mathrm{pick}]\,\langle \mathrm{K'} \rangle \\
\langle \mathrm{K'} \rangle & \rightarrow & [\mathrm{l}]\,\langle \mathrm{L'} \rangle \\
\langle \mathrm{L'} \rangle & \rightarrow & [\mathrm{h}]\,\langle \mathrm{H'} \rangle \\
\langle \mathrm{H'} \rangle & \rightarrow & [\mathrm{b}]\,\langle \mathrm{B'} \rangle \\
\langle \mathrm{B'} \rangle & \rightarrow & [\mathrm{place}]\,\langle \mathrm{B''} \rangle \\
\langle \mathrm{B''} \rangle & \rightarrow & [\mathrm{h}]
\end{array}
$$

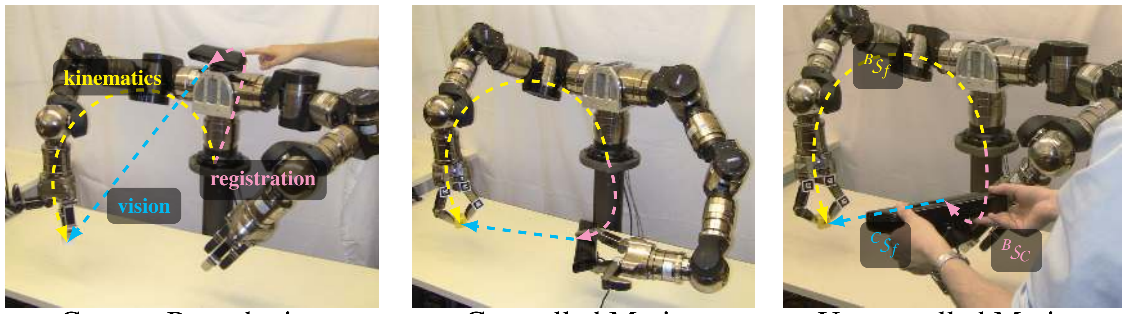
(c) Controlled: G'

Figure 67: Grammars for the Uncontrolled and Controlled mobile manipulator in the Aware Home.

CHAPTER V

PLATFORM MODELS FOR MANIPULATION

Underlying the linguistic models discussed so far are a set of control modes. For our manipulation experiments, we develop control modes to track motions in workspace, focusing on online response. To reach target workspace positions, we introduce a multi-waypoint interpolation scheme that provides more direct paths than previous methods. Integrating visual feedback from cameras requires kinematic registration between the camera and manipulator. Typically, registration is viewed as a static task: it is computed offline and assumed to be constant. In reality, camera registration changes during operation due to external perturbations, wear and tear, or even human repositioning. For example, during the recent DARPA Robotics Challenge trials, impacts from falls resulted in camera issues which significantly affected the robot behavior for some teams [95]. Fig. 68 shows additional use cases which may change the camera pose. The pose registration process should be treated as a dynamic task in which the involved parameters are continuously updated.



Camera Perturbation | Controlled Motion | Uncontrolled Motion

Figure 68: Use cases for online camera registration. We combine the visual and kinematic pose estimates of end effector and filter the result to estimate the camera pose in robot body frame.

5.1 Spherical Parabolic Blends for Workspace Trajectories

Tasks such as screwing in a light bulb or turning a doorknob impose constraints on the motion: rotation must occur along a single fixed axis. For such tasks, we focus on moving with straight-line translation and constant-axis rotation. A common way to specify motions is to provide a sequence of n workspace points for the robot to move through. While generating smooth trajectories from waypoints for both joint-space and the Euclidean-space translations is well studied, the task of continuously transitioning between constant-axis rotations is more challenging. We present a new method to transition through a sequence of constant-axis rotations based on parabolic blending of spherical linear interpolation (SLERP) segments.

The proposed method generates a robot trajectory through a sequence of waypoints such that the axis of rotation remains constant between waypoints and rotational velocity is continuous. Compared to typical approaches for interpolation of robot workspace orientations, this method provides a constant rotational axis between waypoints, is invariant to the local reference frame, and avoids gimbal lock. Compared to classic SLERP, this method transitions through multiple waypoints without stopping whereas SLERP is point-to-point. Compared to typical methods for quaternion splines, this method provides a constant axis of rotation between waypoints. We discuss the application of quaternion interpolation to robot inverse kinematics (see subsection 5.1.1). Then, we derive the equations for spherical parabolic blends to produce our desired trajectories (see subsection 5.1.2), summarizing the trajectory generation algorithm (see subsection 5.1.3). Finally, we demonstrate this method on simulated and physical robot manipulators (see subsection 5.1.4).

Joint-space interpolation is a well studied research topic [37, 112, 113]. However, because the workspace orientations, $SO(3)$, are non-Euclidean, these joint-space methods are not directly applicable to orientation interpolation, particularly when we are concerned with the path taken between waypoints.

A related approach is to apply task constraints while planning a joint-space path [16, 78, 113, 170, 171]. We are considering a different problem: computing a workspace trajectory from a given sequence of waypoints. This enables correcting tracking errors directly in the workspace space (see subsubsection 5.1.3.2).

Typical methods for interpolating robot workspace orientations use Euler angles or *rotation vectors* (the rotation axis scaled by the rotation angle or equivalently the logarithm of the rotation). Interpolating the rotation vector representation [37, p217] varies the angle of rotation, which can produce undesirable paths, see Fig. 71. Euler angle approaches must contend with singularities (gimbal lock). Another approach is to vary the angle of a relative axis-angle orientation [161, p187], though this alone does not address continuity through waypoints. Instead, the quaternion representation is well suited for orientation interpolation as it avoids singularities with Euler angles and provides better paths than rotation vectors.

Spherical Linear Interpolation (SLERP) [160] interpolates between two quaternions along the unit, 4-dimensional hyper-sphere. SLERP has been applied to robot manipulation [1, 3, 35]. SLERP provides the desired constant axis of rotation, but is point-to-point, stopping at the beginning and end of each segment. We improve upon this by transitioning through a sequence of waypoints without stopping.

There is a large body of work on quaternion splines. The primary application domain for these approaches has been computer animation where the intermediate path may not be rigidly constrained compared to the end-effector of a physical robot. Consequently, methods such as quaternion Bezier curves [103, 160], SQUAD [39], and quaternion cubic splines [97] do not provide a constant axis of rotation. A key difference in our approach from these previous methods is that we explicitly differentiate between the interpolation parameter u and time t. By considering du/dt, we can provide a smooth path with constant rotational axis for the bulk of the motion.

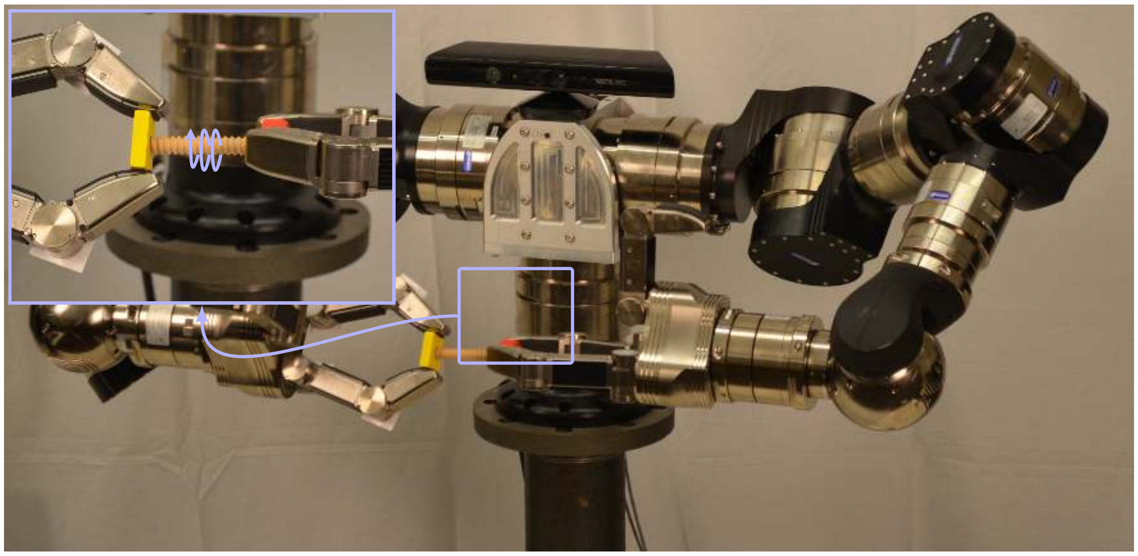


Figure 69: Demonstrating spherical blending for a screwing task on a bimanual Schunk LWA4 manipulator with Schunk SDH hands.

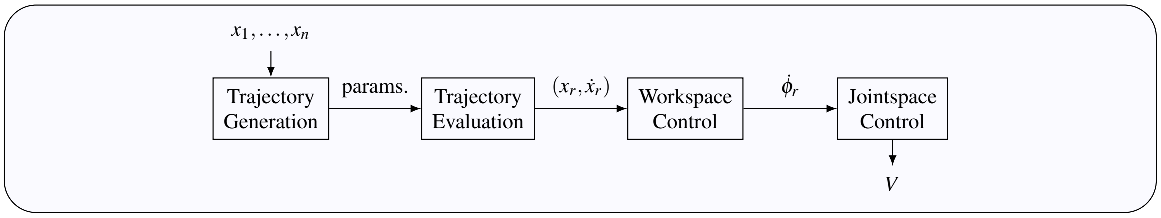


Figure 70: Layers of abstraction going from a sequence of waypoints to inputs at each manipulator axis.

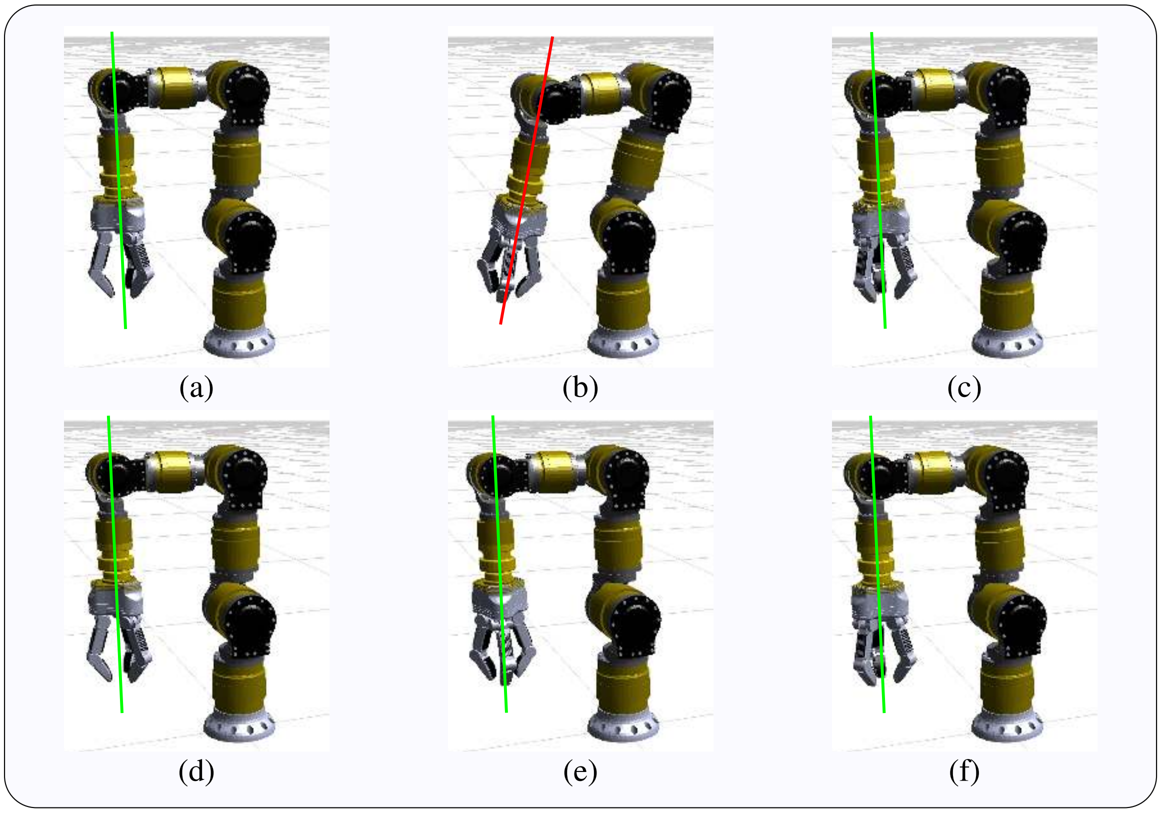


Figure 71: Comparison of rotation vector [37, p217] and Spherical Linear Interpolation. In (a)-(c), interpolating the axis angle representation can give undesirable intermediate orientation (b). In (d)-(f), Spherical Linear Interpolation maintains a constant axis of rotation.

5.1.1 SLERP for Inverse Kinematics

Spherical Linear Interpolation (SLERP) interpolates between an initial and final unit quaternion on the unit sphere [160]. SLERP can be computed as:

$$\texttt{slerp}(q_1, q_2; u) \triangleq q_1 \frac{\sin((1-u)\theta)}{\sin\theta} + q_2 \frac{\sin(u\theta)}{\sin\theta} \tag{92}$$

where q_1 and q_2 are the beginning and end points of the interpolation, interpolation parameter u varies in $[0,1]$, and $\theta = \cos^{-1}(q_1 \cdot q_2)$.[1] To track this interpolation in real-time, we compute u as a function of time.

We build upon SLERP to ensure smoothness of the path by considering the derivatives. Note that we must distinguish between the derivative with respect to interpolation

[1]A more accurate form is $\theta = 2\,\mathrm{atan2}(|q_1 - q_2|, |q_1 + q_2|)$.

parameter u and with respect to time t.

$$\frac{dq}{dt} = \frac{dq}{du}\frac{du}{dt} \tag{93}$$

For a constant q_1 and q_2, the SLERP derivative is:

$$\frac{dq}{du}(u) = q_1 \frac{-\theta \cos((1-u)\theta)}{\sin\theta} + q_2 \frac{\theta \cos(u\theta)}{\sin\theta} \tag{94}$$

Given dq/dt, we can directly compute angular velocity as:

$$\omega = 2\frac{dq}{dt} \otimes q^* \tag{95}$$

where q^* is the conjugate of q and $\otimes$ is the quaternion multiplication operation.

This is then readily applied to robot workspace control via the Jacobian pseudo-inverse:

$$\dot{\phi}_r = (J^+) \begin{bmatrix} \dot{x} \\ \omega \end{bmatrix} \tag{96}$$

where J is the manipulator Jacobian, $\dot{\phi}_r$ is the computed reference joint velocities and $\dot{x}$ is the desired translational velocity. For a robust, practical implementation, further considerations in (96) are also possible to correct position error and handle configurations near joint singularities [133].

However, we still need a method to compute du/dt and ensure continuity of dq/dt

To simplify notation, we will sometimes write the time derivative $\frac{d\alpha}{dt}$ as $\dot{\alpha}$.

5.1.2 Derivation of Spherical Parabolic Blends

SLERP is useful for robots because it provides a constant axis of rotation during the motion. However, this constant axis of rotation introduces difficulties if we want to follow a path with waypoints. Consider the path from q_i via q_j to q_k. If we SLERP from q_i to q_j and q_j to q_k, the path from i to j will have one axis of rotation and the path from j to k will have a different axis of rotation. This would produce a discontinuity in rotational velocity at point j, which could not be suitably followed by a physical robot. Thus, we must transition from the axis ij to axis jk, maintaining C^1 continuity:

Definition 23 (Differentiability class)**.** *A function $f(t)$ is C^k continuous if its derivatives $f', f'', \ldots, f^{(k)}$ exist and are continuous.*

In the constant-axis, linear region from q_i to q_j, we compute the orientation and its derivative via SLERP.

$$u = \frac{t - t_i}{t_j - t_i} \tag{97}$$

$$q(t) = \texttt{slerp}\left(q_i, q_j; u\right) \tag{98}$$

$$\frac{du}{dt} = \frac{1}{t_j - t_i} \tag{99}$$

$$\frac{dq}{dt} = \frac{dq}{du}(u)\frac{du}{dt} \tag{100}$$

where t_i and t_j are the times to reach orientations q_i and q_j, respectively.

Around point q_j, we smoothly change the axis of rotation from that of ij to that of jk. Over some blending interval t_b, we "stretch" the ij interpolation past t_j and the jk interpolation before t_j, ramping the interpolation parameters u_{ij} and u_{jk} over this time. To compute the actual q in this region, we perform a third and final interpolation between the computed values for q_{ij} and q_{jk}.

For this blend region around q_j, we compute the interpolation parameters by ramping down $\dot{u}_{ij}$, ramping up $\dot{u}_{jk}$.

$$t_{ij} = t_j - t_i \tag{101}$$

$$\Delta t = t - (t_j - t_b/2) \tag{102}$$

$$\ddot{u}_{ij} = -\frac{1}{t_{ij} t_b} \tag{103}$$

$$\dot{u}_{ij}(t) = \frac{1}{t_{ij}} + \Delta t \ddot{u}_{ij} \tag{104}$$

$$u_{ij}(t) = \frac{t_{ij} - t_b/2}{t_{ij}} + \frac{\Delta t}{t_{ij}} + \frac{\ddot{u}_{ij}}{2}(\Delta t)^2 \tag{105}$$

$$\ddot{u}_{jk} = \frac{1}{t_b\left(t_k - t_j\right)} \tag{106}$$

$$\dot{u}_{jk}(t) = \Delta t \ddot{u}_{jk} \tag{107}$$

$$u_{jk}(t) = \frac{\ddot{u}_{jk}}{2}(\Delta t)^2 \tag{108}$$

where u_{ij} and u_{jk} are the interpolation parameters from q_i to q_j and from q_j to q_k, respectively, and t_b is the blending period around q_j.

Then, we blend the two trajectories:

$$q_{ij}(t) = \texttt{slerp}\left(q_i, q_j; u_{ij}(t)\right) \tag{109}$$

$$q_{jk}(t) = \texttt{slerp}\left(q_j, q_k; u_{jk}(t)\right) \tag{110}$$

$$q(t) = \texttt{slerp}\left(q_{ij}(t), q_{jk}(t); u_j(t)\right) \tag{111}$$

Now, we return to computing the time derivitive of SLERP, with the complication that the interpolation points in this case vary over time. This will let us derive the interpolation parameter for the blend region, $u_j(t)$, such that angular velocity is continuous. We can more precisely write the SLERP formula as:

$$\begin{aligned} q(t) &= q_1(t)\frac{\sin\left((1-u(t))\theta(t)\right)}{\sin\theta(t)} + q_2(t)\frac{\sin\left(u(t)\theta(t)\right)}{\sin\theta(t)} \\ &= q_1(t)a(t) + q_2(t)b(t) \end{aligned} \tag{112}$$

where $u(t)$ and $\theta(t)$ are time varying functions, and $a(t)$ and $b(t)$ are substituted variables for the coefficients of $q_1(t)$ and $q_2(t)$.

Then, the time derivative is:

$$\frac{dq}{dt}(t) = \dot{q}_1(t)a(t) + q_1(t)\dot{a}(t) + \dot{q}_2(t)b(t) + q_2(t)\dot{b}(t) \tag{113}$$

The values of $\dot{q}_1$ and $\dot{q}_2$ can be computed from dq/du in (94) and $\dot{u}$. We differentiate to find $\dot{a}(t)$ and $\dot{b}(t)$, dropping the parameter t for brevity.

$$c_a = \cos\left(\theta\left(1-u\right)\right) \qquad s_a = \sin\left(\theta\left(1-u\right)\right)$$

$$c_b = \cos\left(\theta u\right) \qquad s_b = \sin\left(\theta u\right)$$

$$\dot{a} = \frac{c_a\left(\dot{\theta}(1-u) - \dot{u}\theta\right)}{\sin\theta} - \frac{\dot{\theta}\cos\left(\theta\right)s_a}{\sin^2\theta} \tag{114}$$

$$\dot{b} = \frac{\left(\dot{\theta}u + \theta\dot{u}\right)c_b}{\sin\theta} - \frac{\dot{\theta}\cos\left(\theta\right)s_b}{\sin^2\theta} \tag{115}$$

From (114) and (115) we can compute $u_j(t)$ to ensure that angular velocity is continuous. For this, we must ensure that the angular velocity at the beginning of the blend equals the angular velocity at the end the preceding linear segment and that angular velocity at the end of the blend equals that at the beginning of the following linear segment:

$$\dot{q}_j(t_j - t_b/2) = \dot{q}_{ij}(t_j - t_b/2) \tag{116}$$

$$\dot{q}_j(t_j + t_b/2) = \dot{q}_{jk}(t_j + t_b/2) \tag{117}$$

At the beginning of the blend segment around j where $t = t_j - t_b/2$, we know $u_j = 0$, so we can simplify the coefficients of $\dot{q}_j$ in (113) as follows:

$$\dot{q}_j(t_j - t_b/2) = \dot{q}_{ij} + q_{ij}\dot{a} + q_{jk}\dot{b}_j \tag{118}$$

$$\dot{a}_j(t_j - t_b/2) = \dot{b}_j(t_j - t_b/2) = \frac{\dot{u}_j\theta}{\sin\theta} \tag{119}$$

Thus, if we have $\dot{u}_j(t_j - t_b/2) = 0$, then the coefficients $\dot{a}$ and $\dot{b}$ will be zero and (116) will be satisfied. A similar property holds at the end of the blend region, so we must have $\dot{u}_j(t_j - t_b/2) = \dot{u}_j(t_j + t_b/2) = 0$. We satisfy this property by computing u_j based on a constant second derivative:

$$\ddot{u}_j = \frac{4}{t_b^2} \tag{120}$$

$$u_j(t) = \begin{cases} t \le t_j & 0.5\ddot{u}_j\,(\Delta t)^2 \\ t > t_j & 1 - 0.5\ddot{u}_j\left(t_j + t_b/2 - t\right)^2 \end{cases} \tag{121}$$

From (121), we compute $u_j(t)$ for (111) and $\dot{u}_j$ for (93). Fig 72 plots values of u_{ij}, u_{jk} and u_j over the linear and blend regions.

Now, we find $\dot{\theta}$. To simplify, we assume that $q_1(t)$ and $q_2(t)$ are unit quaternions.

$$\theta(t) = \cos^{-1}\left(q_1(t) \cdot q_2(t)\right)$$

$$\dot{\theta}(t) = -\frac{q_1(t)\cdot\dot{q}_2(t) + \dot{q}_1(t)\cdot q_2(t)}{\sqrt{1 - \left(q_1(t)\cdot q_2(t)\right)^2}} \tag{122}$$

Combining (113) - (122), we compute the quaternion derivative dq/dt, and with (95), we find the angular velocity ω in the blend region.

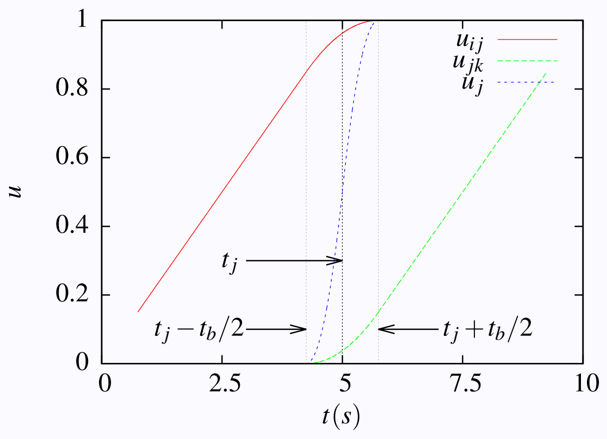


Figure 72: SLERP u values over linear and blend regions. u_{ij} is the interpolation parameter between points i and j, which ramps down during the blend. u_j is the interpolation parameter for the blend region around j when ramps up, then down during the blend. u_{jk} is the interpolation parameter between points j and k, which ramps up during the blend.

5.1.3 Generating and Tracking Trajectories

Following the derivation in subsection 5.1.2, we summarize generating trajectory parameters from a sequence of waypoints, computing reference workspace velocities, and finding corresponding reference joint velocities.

5.1.3.1 Generation

Given a sequence of orientations q_i, waypoint times t_i, and blend times t_{bi}: $(q_0,t_0,t_{b0}),(q_1,t_1,t_{b1})\ldots,(q_n,t_n$

1. Add virtual waypoints at $q = q_0$ at time $t_0 + t_b/2$ and $q = q_n$ at $t_n - t_b/2$ to the trajectory to provide blending for initial and final points.

2. For every triplet of orientations, q_i, q_j, and q_k, compute $\ddot{u}_{ij}$, $\ddot{u}_{jk}$, and $\ddot{u}_j$ according to (103), (106), and (120).

5.1.3.2 Tracking

To track a generated trajectory, alternate between a sequence of blend and linear regions. Around each waypoint q_j from $t_j - t_b/2$ to $t_j + t_b/2$, blend orientations. From $t_j + t_b/2$ to $t_k - t_b/2$, SLERP from q_j to q_k.

Note that there is no linear region between q_0 and the virtual waypoints at $t_0 + t_b/2$, nor between the virtual waypoints at $t_n - t_b/2$ and q_n.

Linear Regions: For the linear region between q_i and q_j:

1. $\dot{u}_{ij} = \frac{1}{t_j - t_i}$

2. Compute $u_{ij}(t) = (t - t_i)\,\dot{u}_{ij}$

3. Compute $q(u_{ij})$ according to (92)

4. Compute $\dot{q}(u_{ij})$ according to (94) and (93)

5. Compute $\omega(t)$ according to (95)

Blend Regions: For the blend region between q_i, q_j, and q_k:

1. Compute $\dot{u}_{ij}$, u_{ij}, $\dot{u}_{jk}$, u_{jk}, $\dot{u}_j$, and u_j according to (104), (105), (107), and (108).

2. Compute $q_{ij} = \texttt{slerp}\left(q_i, q_j; u_{ij}\right)$ and $q_{jk} = \texttt{slerp}\left(q_j, q_k; u_{jk}\right)$

3. Compute $\dot{q}_{ij}$ $\dot{q}_{jk}$ according to (94) and (93).

4. Compute $q(u_j) = \texttt{slerp}\left(q_{ij}, q_{jk}; u_j\right)$

5. Compute $\dot{q}(u_j)$ from (122), (114), (115), and (113).

6. Compute $\omega(t)$ according to (95)

5.1.3.3 Workspace Control

Now, we apply a singularity-robust Jacobian inverse kinematics to obtain joint velocities from the generated workspace trajectory [133]. To provide acceptable performance near joint singularities, we compute the damped pseudo-inverse of the Jacobian as follows:

$$J^{+} = \sum_{i=0}^{\min(m,n)} \frac{s_i}{\max({s_i}^2, {s_{\min}}^2)} v_i u_i^T \tag{123}$$

where $J = USV^T$ is the singular value decomposition of J and $s_{\min}$ is a selected constant for the minimum acceptable singular value.[2]

We compute the damped-least squares solution for feed-forward velocity and feedback-position control:

$$\begin{aligned}\dot{\phi}_r &= J^+ \left(\begin{bmatrix} \dot{x}_r \\ \omega_r \end{bmatrix} - k_x e \right) \\ &= J^+ \left(\begin{bmatrix} \dot{x}_r \\ \omega_r \end{bmatrix} - k_x \begin{bmatrix} x - x_r \\ \ln\left(q \otimes q_r^*\right) \end{bmatrix} \right) \end{aligned} \tag{124}$$

where e is the position error.

In addition, for redundant manipulators with more than six degrees of freedom, we use the null-space projection to help avoid joint limits by directing the joint positions towards a nominal zero point.

$$\dot{\phi}_r = J^+ \left(\begin{bmatrix} \dot{x}_r \\ \omega_r \end{bmatrix} - k_x \begin{bmatrix} x - x_r \\ \ln\left(q \otimes q_r^*\right) \end{bmatrix} \right) - k_\phi (J^+ J - I)(\phi - \phi_0) \tag{125}$$

where k_x is the workspace position error gain, k_ϕ is the null-space projection gain, and ϕ_0 is the nominal zero configuration.

We then use joint-level velocity control to track the reference joint velocities $\dot{\phi}_r$.

5.1.4 Trajectory Experiments

5.1.4.1 Simulation

We first demonstrate these trajectories on a kinematically simulated Schunk LWA3 robot with 7 Degrees of Freedom (DOF). Fig. 73 shows the workspace orientation and derivative through a sequence of orientations. From this, we can see that orientation is C^1 continuous. Between each of the waypoints, we have an accelerating segment, a constant-velocity segment, and a decelerating segment.

[2]On the Schunk LWA4 in Fig. 69, a reasonable value for $s_{\min}$ is .01.

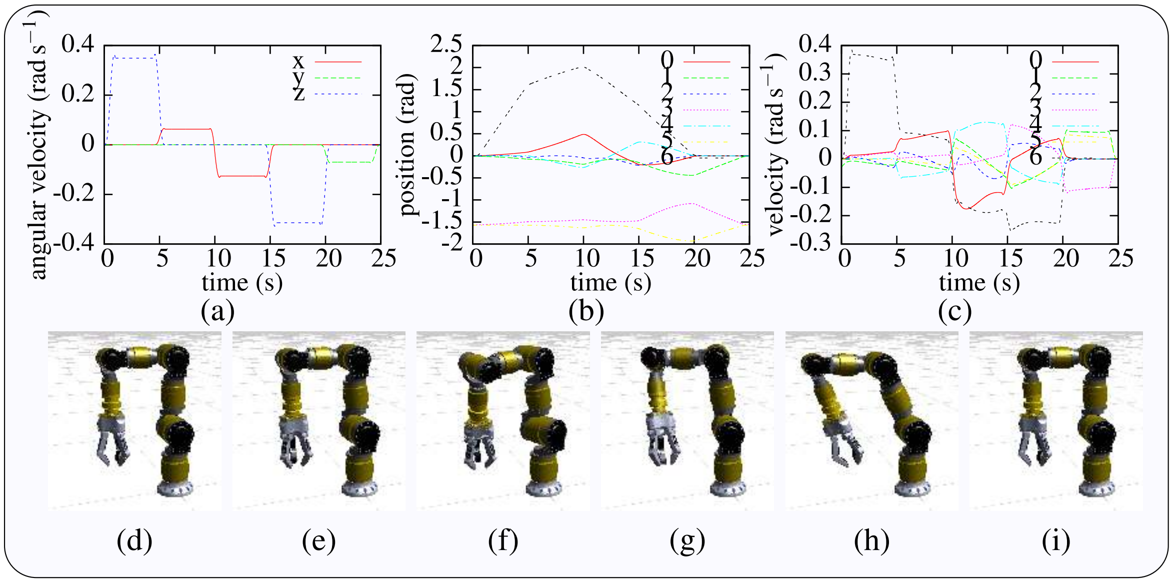


Figure 73: Simulated trajectory through the following waypoints specified in XZY Euler Angles: $(-\pi/2, 0, \pi)$, $(-\pi/2, \pi/10, \pi)$, $(-\pi/2, -\pi/10, -\pi)$, $(\pi, -\pi/10, -\pi)$, $(\pi, 0, \pi)$. Interpolation is performed on the quaternion representation. (a) Workspace Angular Velocity. (b) Joint Position. (c) Joint Velocity. (d)-(i) Via Orientations.

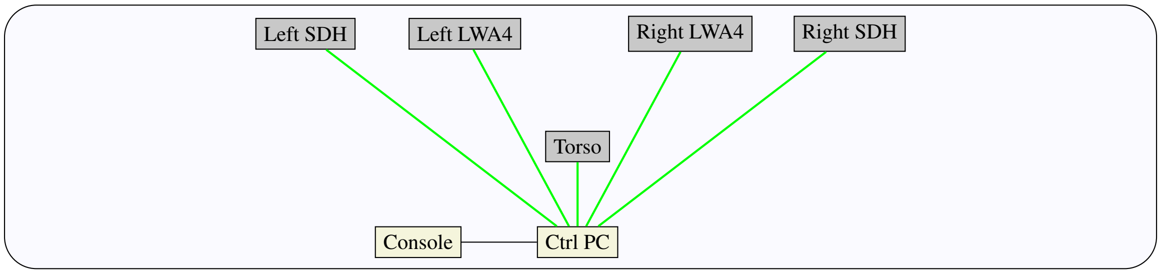


Figure 74: Block Diagram of robot manipulator hardware components. The control PC communicates with the servo controllers over several CAN buses.

5.1.4.2 Physical Implementation

We validate our trajectory generation approach on a physical Schunk LWA4 arm with Schunk SDH hand for a screwing task, Fig. 69. The LWA4 is a 7 DOF arm that offers a shorter distance between wrist point and end-effector than the LWA3. Fig. 74 gives an overview of the major physical system components. Our real-time software runs on Xeon E3-1270v2 PC under Linux 3.4.18-rt29 PREEMPT RT, and is implemented as multiple operating system processes communicating using the Ach interprocess communication library [46]. Fig. 75 summarizes the real-time software components.

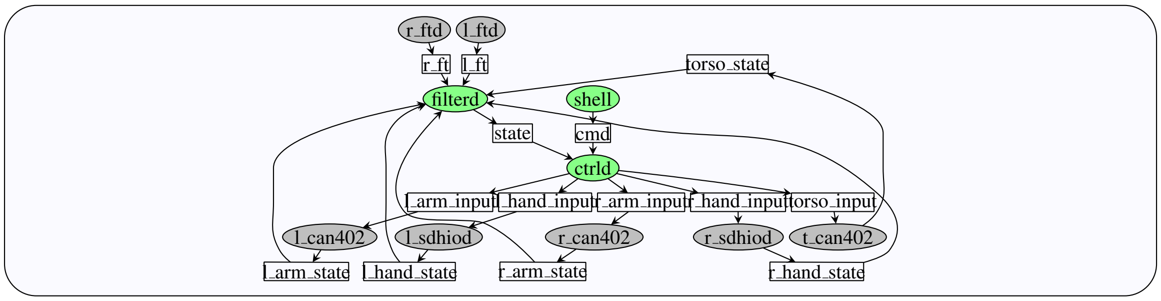


Figure 75: Block diagram of real-time software components. Gray ovals are user-space driver processes, green ovals are controller processes, and rectangles are Ach channels.

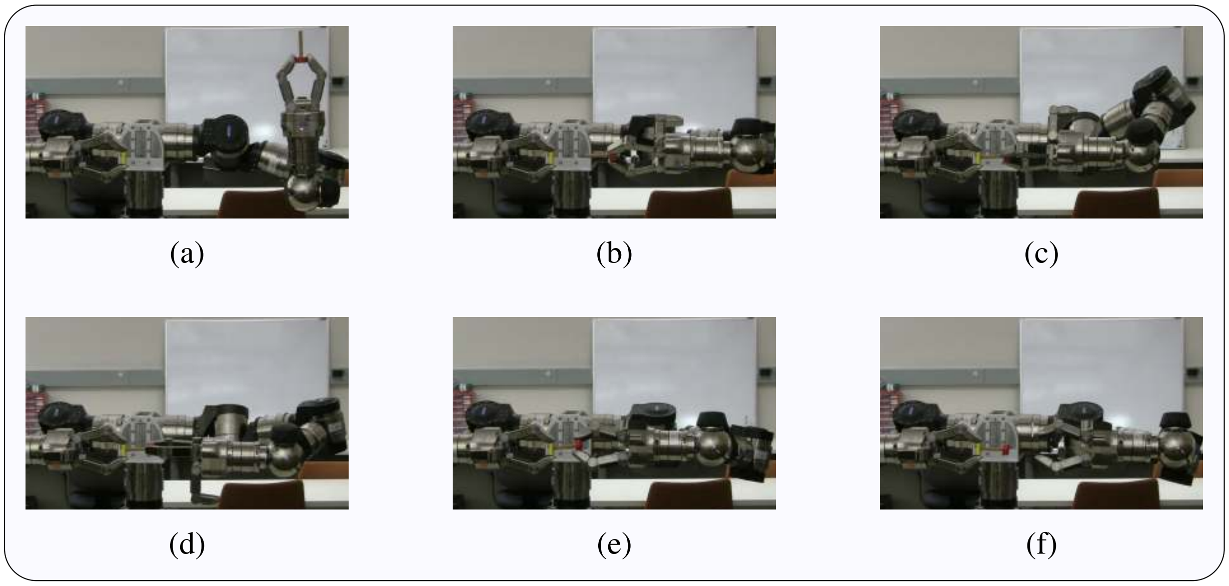


Figure 76: Physical Screwing Task. (a)-(e), trajectory waypoints. (f), the inserted screw.

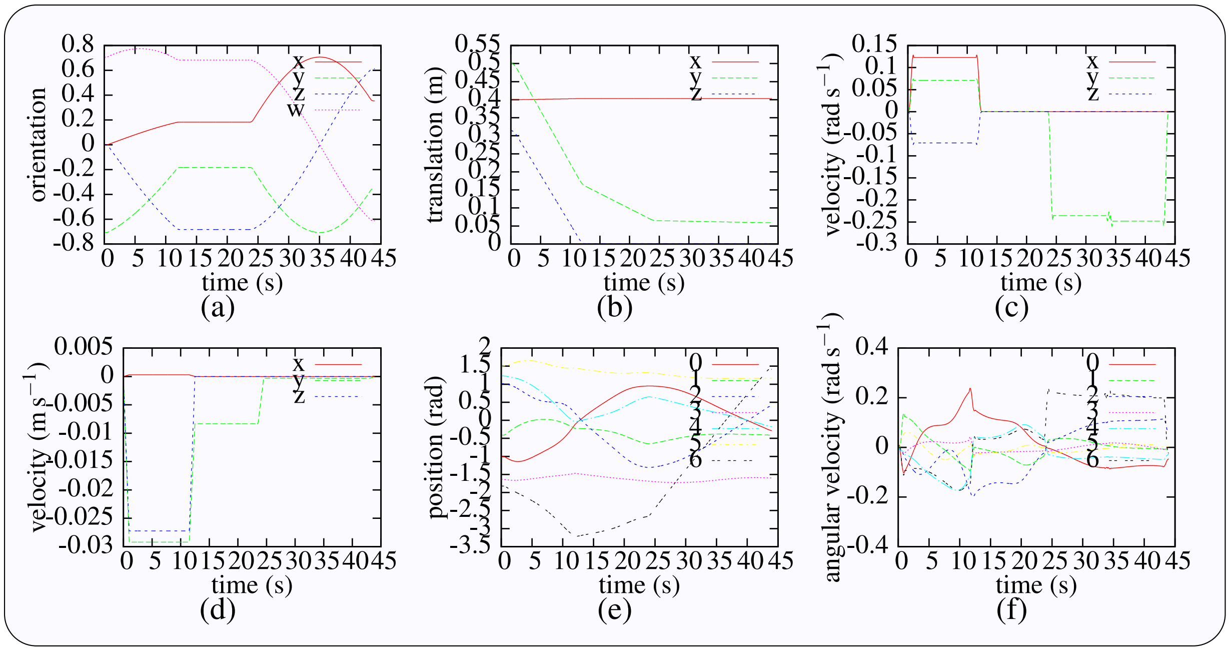


Figure 77: Plots of positions and velocities for physical screwing task. (a) Workspace orientation. (b) Workspace translation. (c) Workspace rotational velocity. (d) Workspace translational velocity. (e) Joint positions. (f) Joint velocity

Table 6: Time to compute reference parameters per control cycle. Average of 10 million evaluations on an Intel Xeon E5-1620.

Linear Region	0.39 µs
Blend Region	1.3 µs

We generate and execute a trajectory to screw together the wooden pieces. Because of the SDH's kinematic configuration, it grasps the screw such that the screw axis is offset from the last wrist axis. Thus, we cannot turn the screw by rotating only the last joint but must instead consider the entire arm. The provided waypoints to insert the screw are shown in Fig. 76, and the generated trajectory in workspace and joint space is plotted in Fig. 77. This trajectory aligns, inserts, and turns the screw in one continuous, non-stop motion.

5.1.4.3 Computational Performance

This method is computationally efficient. The generation phase to pre-compute parameters requires $O(n)$ time, where n is the number of waypoints. The tracking phase requires $O(1)$ time during each control cycle. Tracking does require evaluating a few transcendental functions during each control cycle; however, for modern CPUs at typical real-time control rates, e.g., one kilohertz, this is not a significant factor. Table 6 shows evaluation times on the order of one microsecond for a recent Intel CPU.

5.2 Online Registration of Single Arm and Camera

To address changes in camera pose during operation, we propose an online camera registration method that combines (1) visual tracking of features on the manipulator, (2) a novel expectation-maximization inspired algorithm for pose filtering and tracking, and (3) an special Euclidean group constrained extended Kalman filter. Our key insight is to use the robot body as a reference for the registration process. By tracking known patterns or objects on the robot, we can continuously collect evidence for the current camera pose. However, naïve filtering of these pose estimates can lead to large variances in the calculated poses. The challenge is obtaining sufficient accuracy for manipulation through the online registration. To address this challenge, we combine pose filtering and manipulator control, incorporating camera registration into our manipulation feedback loop.

This section presents a method for online registration and manipulation that combines object tracking, pose filtering, and visual servoing. First, we use perceptual information to identify the pose of specific features on the end-effector of the controlled robot (see subsubsection 5.2.1.1). Then, we perform an initial fit to find offsets of the features on the robot, (see subsubsection 5.2.1.2). A special Kalman filter is, then, used in conjunction with median filtering in order to perform online registration of the camera (subsubsection 5.2.1.3). In our evaluation (see subsection 5.2.3), we investigate the accuracy of the proposed method by applying it to robot grasping and manipulation tasks.

Typical camera registration methods collect a set of calibration data using an external reference object, compute the calibration, then proceed assuming the calibration is static. OpenCV determines camera registration from point correspondences, typically using a chessboard [141]. Pradeep, et. al, develop a camera and arm calibration approach based on bundle adjustment and demonstrate it on the PR2 robot [147]. This approach requires approximately 20 minutes to collect data and another 20 minutes for computation, a challenge for handling changing pose online.

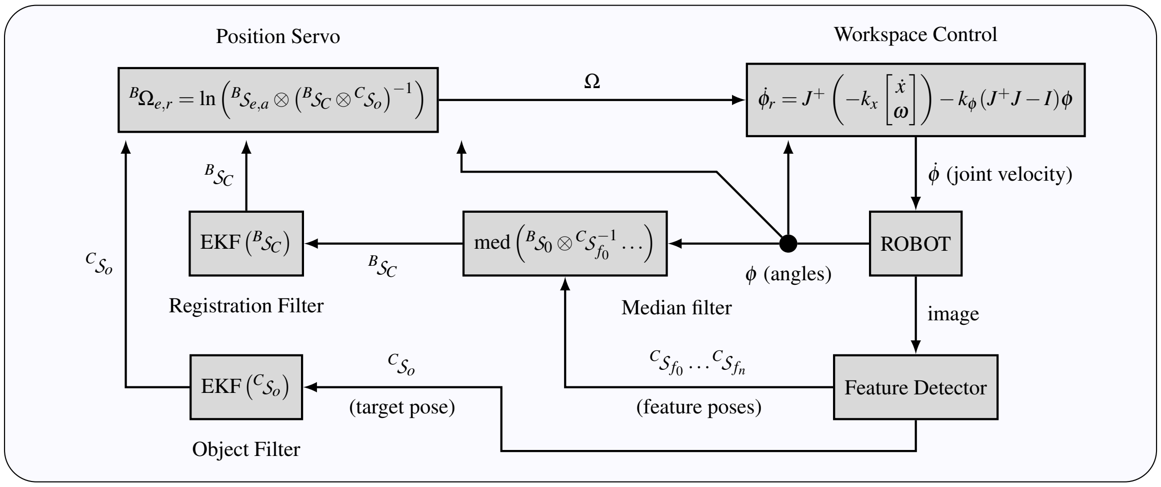
Visual servo control incorporates camera feedback into robot motion control [28, 29]. The two main types of visual servoing are image-based visual servo control (IBVS), which operates on features in the 2D image, and position-based visual servo control, which operates on 3D parameters. Both of these methods assume a given camera registration. While IBVS is locally stable with regard to pose errors, under PBVS, even small pose errors can result in large tracking error [28]. Our proposed method addresses these challenges by correcting the camera registration online. In our experiments we show the importance of treating the registration process as a dynamic task. Furthermore, we show that our online registration achieves millimeter positioning accuracy of the manipulator. This is particularly important for grasping tasks performed using multi-fingered robot hands [15]. During such grasping tasks, inaccuracies in perception and forward kinematics often lead to premature contact between one finger and the object. As a result of the ensuing object movement, the intended grasp might not be satisfactorily executed or may fail altogether.

Other recent work has explored online visual parameter identification. [105] tracks a robot arm to identify encoder offsets. This method assumes a given camera registration, but is also tolerant of some registration error. In contrast, our work identifies the camera registration online, but does not explicitly consider encoder offsets. [79] considers bimanual arm and object tracking with vision and tactile feedback. Though the hardware and implementation differ from work presented in this paper, similar accuracy is obtained. [173] uses maps generated from a Simultaneous Localization and Mapping (SLAM) algorithm to calibrate a depth sensor. In our approach, unlike typical environments for SLAM, the object to which we are trying to register our camera – the manipulator – will necessarily be in motion.

5.2.1 Technical Approach

We determine the pose registration between the camera and the manipulator by visually tracking the 3D pose of the arm. We identify the pose of texture or shape features on the arm and fit a transformation based on the corresponding kinematic pose estimates of those features. To obtain sufficient accuracy for manipulation, we combine several methods to fit and filter the visual pose estimates before servoing to the target object. This estimation and control loop is summarized in Fig. 78.

For computational reasons, we used the *dual quaternion* representation for the special Euclidean group $\mathscr{S}\mathscr{E}(3)$. Compared to matrices, the dual quaternion has lower dimensionality and is more easily normalized, both advantages for our filtering implementation. The relevant dual quaternion equations are summarized in appendix B. We represent the dual quaternion $\mathcal{S}$ for a transformation implicitly as a tuple of a rotation quaternion q and



Position Servo
Workspace Control
${}^{B}\Omega_{e,r} = \ln\left({}^{B}\mathcal{S}_{e,a} \otimes \left({}^{B}\mathcal{S}_{C} \otimes {}^{C}\mathcal{S}_{o}\right)^{-1}\right)$
Ω
$\dot{\phi}_r = J^{+}\left(-k_x \begin{bmatrix}\dot{x}\\ \omega\end{bmatrix}\right) - k_\phi (J^{+}J - I)\phi$
${}^{B}\mathcal{S}_{C}$
$\dot{\phi}$ (joint velocity)
EKF $({}^{B}\mathcal{S}_{C})$
${}^{B}\mathcal{S}_{C}$
med $\left({}^{B}\mathcal{S}_{0} \otimes {}^{C}\mathcal{S}_{f_0}^{-1} \ldots\right)$
ROBOT
${}^{C}\mathcal{S}_{o}$
ϕ (angles)
Registration Filter
Median filter
image
${}^{C}\mathcal{S}_{o}$
${}^{C}\mathcal{S}_{f_0} \ldots {}^{C}\mathcal{S}_{f_n}$
EKF $({}^{C}\mathcal{S}_{o})$
(target pose)
(feature poses)
Feature Detector
Object Filter

Figure 78: Block Diagram of Control System. 3D poses for features are detected from visual data. The median camera transform is computed over all features and then Kalman filtered. With this registration, the robot servos in workspace to a target object location.

translation vector v: $\mathcal{S} = \big(q,\, v\big)$. This requires only seven elements. For Euclidean transformations, we use the typical coordinate notation where leading superscript denotes the parent frame and following subscript denotes the child frame, i.e., ${}^{x}\mathcal{S}_{y}$ gives the origin of y relative to x. The transformation ${}^{a}\mathcal{S}_{b}$ followed by ${}^{b}\mathcal{S}_{c}$ is given as the dual quaternion multiplication ${}^{a}\mathcal{S}_{b} \otimes {}^{b}\mathcal{S}_{c} = {}^{a}\mathcal{S}_{c}$.

5.2.1.1 *Feature Estimation*

To use the robot body as a reference for camera registration, it is important to identify and track body parts, e.g., the end-effector, in 3D. These 3D poses can be estimated with marker-based [150] and model-based approaches [31], see Fig. 79. Marker-based approaches require attaching fiducials to known locations on the robot, such as the fingers. Model-based tracking, on the other hand, requires accurate polygon meshes of the tracked object. In our implementation, we use the ALVAR library [150] for marker-based tracking. For model-based tracking, we use the approach from [31]. In each frame, the 3D pose of the object is computed by projecting a 3D CAD model into the 2D image. After projection, we identify salient edges in the model and align them with edges in the 2D image. A particle filter is then used to filter the pose estimates over time. Both marker-based and model-based tracking provide 3D pose estimates of tracked features, but with frequent outliers and noise. Markers have the advantage of being easy to deploy, while model-based tracking can deal with partial occlusions of the scene.

5.2.1.2 *Offset Identification*

To improve the accuracy of kinematic pose estimates for features, we initially perform a static expectation-maximization-like [53] procedure, based on the following model:

$$ {}^{B}\mathcal{S}_{k} \otimes {}^{k}\mathcal{S}_{f} = {}^{B}\mathcal{S}_{C} \otimes {}^{C}\mathcal{S}_{f} \tag{126} $$

where ${}^{B}\mathcal{S}_{k}$ is the measured nominal feature pose in the body frame determined from encoder positions and forward kinematics, ${}^{k}\mathcal{S}_{f}$ is the unknown static pose offset of the feature due to

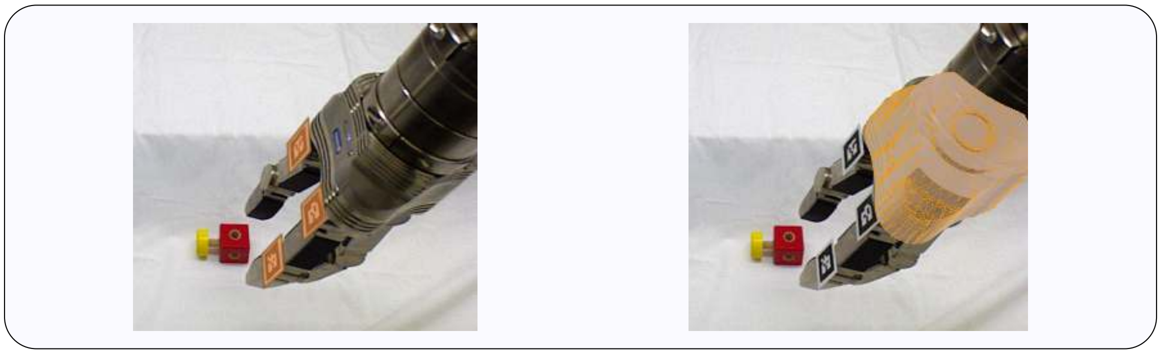


Figure 79: Marker-based tracking (left) and model-based tracking (right).

inaccuracy of manual placement, ${}^{B}\mathcal{S}_{C}$ is the unknown camera registration in the body frame, and ${}^{C}\mathcal{S}_{f}$ is the visually measured feature pose in the camera frame. These transforms are summarized in Fig. 68, with ${}^{B}\mathcal{S}_{k} \otimes {}^{k}\mathcal{S}_{f}$ combined as ${}^{B}\mathcal{S}_{f}$.

As an initialization step, we iteratively fix either ${}^{k}\mathcal{S}_{f}$ or ${}^{B}\mathcal{S}_{C}$ in (126) and solve for the other using Umeyama's algorithm [181]. This gives us the relative transforms for the features ${}^{k}\mathcal{S}_{f}$ which we assume are static.

5.2.1.3 Filtering

To compute the online registration, where ${}^{B}\mathcal{S}_{C}$ is changing, we combine median and Kalman filtering. The median filter is applied *independently* at each time step to reject major outliers in the estimated feature poses. Compared to weighted least squares methods, the median requires no parameter tuning and is especially resistant, tolerating outliers in up to 50% of the data [75]. Given the median at each step, the Kalman filter is applied *over time* to generate an optimal registration estimate under a Gaussian noise assumption.

Based on (126), each observed feature on the robot gives on estimate for the camera registration ${}^{B}\mathcal{S}_{C}$:

$$ {}^{B}\mathcal{S}_{k} \otimes {}^{k}\mathcal{S}_{f} \otimes ({}^{C}\mathcal{S}_{f})^{-1} = {}^{B}\mathcal{S}_{C} \tag{127} $$

5.2.1.4 *Median Filtering*

At each time step, we find the median registration over all observed features. Each observed feature gives a candidate registration ${}^{B}\mathcal{S}_{C}$. First, we collect a set Q of the orientation candidates:

$$Q = \left\{ ({}^{B}q_{C})_i \mid ({}^{B}\mathcal{S}_k)_i \otimes {}^{k}\mathcal{S}_f \otimes ({}^{C}\mathcal{S}_f)_i^{-1} \right\} \tag{128}$$

Then, we compute the median of the candidate orientation registrations Q. To find this median, the structure of rotations in $\mathscr{SO}(3)$ offers a convenient distance metric between two orientations: the angle between them. Using this geometric interpretation, the median orientation $\hat{q}$ is the orientation with minimum angular distance to all other orientations.

$$\widehat{{}^{B}q_{C}} = \underset{q_i \in Q}{\arg\min} \sum_{j=0}^{n} |\ln(q_i^* \otimes q_j)| \tag{129}$$

The median translation $\hat{x}$ is the conventional geometric median, the translation with minimum Euclidean distance to all other translations. First, we find the set of candidate translations Z by rotating the feature translation in camera frame ${}^{C}v_f$ and subtracting from the body frame translation ${}^{B}v_f$:

$$Z = \left\{ z_i \mid z_i = {}^{B}v_{f,i} - \widehat{{}^{B}q_{C}} \otimes {}^{C}v_{f,i} \otimes \widehat{{}^{B}q_{C}}^{*} \right\} \tag{130}$$

Then, we compute the geometric median of the candidate translations by finding the element with minimum distance to all other elements:

$$\widehat{{}^{B}v_{C}} = \underset{z_i \in Z}{\arg\min} \sum_{j=0}^{n} |z_i - z_j| \tag{131}$$

Then, the median transform is the combination of the orientation and translation parts:

$$\widehat{{}^{B}\mathcal{S}_{C}} = \left(\widehat{{}^{B}q_{C}}, \widehat{{}^{B}v_{C}} \right) \tag{132}$$

5.2.1.5 *Kalman Filtering*

Next, we use an Extended Kalman filter (EKF) to attenuate noise over time, taking care to remain in the $\mathscr{SE}(3)$ manifold. Similar Kalman filters are discussed in [32, 116]. The quasi-linearity of quaternions means the EKF is suitable for orientation estimation in this application [114].

To filter $\mathscr{SE}(3)$ poses, we consider state x composed of a quaternion q, a translation vector v, and the translational and rotational velocities, $\dot{v}$ and ω:

$$x = (q, v) = [q_x, q_y, q_z, q_w, v_x, v_y, v_z, \dot{v}_x, \dot{v}_y, \dot{v}_z, \omega_x, \omega_y, \omega_z]$$

The measurement z is the pose:

$$z = (q, v) = [q_x, q_y, q_z, q_w, v_x, v_y, v_z]$$

The general EKF prediction step for time k is:

$$\hat{x}_{k|k-1} = f(\hat{x_{k-1}}) \tag{133}$$

$$F_{k-1} = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}} \tag{134}$$

$$P_{k|k-1} = F_{k-1} P_{k-1|k-1} F_{k-1}^T + Q_{k-1} \tag{135}$$

where $\hat{x}$ is the estimated state, $f(x)$ is the process model, F is the Jacobian of f, P is the state covariance matrix, and Q is the process noise model.

The process model then integrates the translational and rotational velocity, staying in the $\mathscr{SE}(3)$ manifold using the dual quaternion exponential of the twist Ω:

$$\Omega(\omega, \dot{v}, v) = \Big(\omega,\ v \times \omega + \dot{v}\Big)$$

$$f(x) = \exp\left(\frac{\Delta t}{2}\Omega\right) \otimes (q,\ v) \tag{136}$$

Now, we find the process Jacobian F. The translation portion is a diagonal matrix of the translational velocity. For the orientation portion, we find the quaternion derivative $\dot{q}$ from the rotational velocity:

$$\dot{q} = \frac{1}{2}\omega \otimes q \tag{137}$$

This quaternion multiplication can be converted into the following matrix multiplication:

$$\frac{1}{2}\omega \otimes q = \frac{1}{2}M_r(q)\,\omega$$
$$M_r(q) = \begin{bmatrix} q_w & q_z & -q_y \\ -q_z & q_w & q_x \\ q_y & -q_x & q_w \\ -q_x & -q_y & -q_z \end{bmatrix} \tag{138}$$

Note that we omit the w column of the typical quaternion multiplication matrix because the w element of rotational velocity ω is zero.

This gives the following process 13×13 Jacobian F:

$$F = \begin{bmatrix} I_{4\times 4} & 0 & \frac{1}{2}\Delta t M_r\left(q\right) & 0 \\ 0 & I_{3\times 3} & 0 & \Delta t I_{3\times 3} \\ 0 & 0 & I_{3\times 3} & 0 \\ 0 & 0 & 0 & I_{3\times 3} \end{bmatrix} \tag{139}$$

Now we consider the EKF correction step. The general form is:

$$\hat{z}_k = h(\hat{x}_{k|k-1}) \tag{140}$$
$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}} \tag{141}$$
$$y_k = v(z_k, \hat{z}) \tag{142}$$
$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{143}$$
$$H_k P_{k|k-1} = S_k K_k^T \tag{144}$$
$$\hat{x}_{k|k} = p(\hat{x}_{k|k-1}, K_k y_k) \tag{145}$$
$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{146}$$

where z is the measurement, h is the measurement model, H is the Jacobian of h, $\hat{z}$ is the estimated measurement, R is the measurement noise model, and K is the Kalman gain, v

is a function to compute measurement residual, and p is a function to compute the state update.

We compute the EKF residuals and state updates using relative quaternions to remain in $\mathscr{S}\mathscr{E}(3)$ without needing additional normalization. The observation $h(x)$ is a pose estimate:

$$\begin{aligned} h(x) &= (q, v) \\ H &= I_{7\times 7} \end{aligned} \tag{147}$$

We compute the measurement residual based on the relative rotation between the measured and estimated pose:

$$\begin{aligned} v(z, \hat{z}) &= (y_q, y_v) \\ y_q &= \ln\left(z_q \otimes \hat{z}_q^*\right) \otimes q \\ y_v &= z_v - \hat{z}_v \end{aligned} \tag{148}$$

where y_q is the orientation part of the residual and y_v the translation part. Note that that $\ln\left(z_q \otimes \hat{z}_q^*\right)$ corresponds to a velocity in the direction of the relative transform between the actual and expected pose measurement and that we can consider y_q as a quaternion derivative. Then, the update function will integrate the pose portion of y, again using the exponential of the twist. First, we find the twist corresponding to the product of the Kalman gain K and the measurement residual y:

$$\begin{aligned} (Ky)_\phi &= (Ky)_q \otimes q^* \\ \Omega(Ky, v) &= \left((Ky)_\phi, v \times (Ky)_\phi + (Ky)_v\right) \end{aligned} \tag{149}$$

Then, we integrate estimated pose using the exponential of this twist:

$$(x_{(q,v)})_{k|k} = \exp\left(\frac{\Delta t}{2}\Omega\right) \otimes (q, v) \tag{150}$$

Finally, the velocity component of innovation y is scaled and added:

$$(x_{\omega,\dot{v}})_{k|k} = x_{\omega,\dot{v}} + (Ky)_{\omega,\dot{v}} \tag{151}$$

5.2.2 Registered Visual Servoing

We use the computed camera registration $^{B}\mathcal{S}_{C}$ to servo to a target object according to the control loop in Fig. 78. This is position-based visual servoing, incorporating the dynamically updated registration. First, we compute a reference twist $^{B}\blacksquare_{e,ref}$ from the position error using camera pose $^{B}\mathcal{S}_{C}$ and object pose $^{C}\mathcal{S}_{o}$:

$$^{B}\mathcal{S}_{e,ref} = {}^{B}\mathcal{S}_{C} \otimes {}^{C}\mathcal{S}_{obj} \tag{152}$$

$$^{B}\Omega_{e,ref} = \ln\left({}^{B}\mathcal{S}_{e,act} \otimes {}^{B}\mathcal{S}_{e,ref}^{-1}\right) \tag{153}$$

Then, we find the reference velocity for twist $^{B}\Omega_{e,ref}$:

$$\begin{bmatrix} \dot{x} \\ \omega \end{bmatrix} = \begin{bmatrix} \mathbb{D}({}^{B}\Omega_{e,ref}) - (2\mathbb{D}({}^{B}\mathcal{S}_{e}) \otimes \mathbb{R}({}^{B}\mathcal{S}_{e})^{-1}) \times \mathbb{R}({}^{B}\Omega_{e,ref}) \\ \mathbb{R}({}^{B}\Omega_{e,ref}) \end{bmatrix} \tag{154}$$

where $\mathbb{R}(X)$ is the real part of X and $\mathbb{D}(X)$ is the dual part of X.

Finally, we compute joint velocities using the Jacobian damped least squares, also using a nullspace projection to keep joints near the zero position:

$$\dot{\phi}_r = J^{+}\left(-k_x \begin{bmatrix} \dot{x} \\ \omega \end{bmatrix}\right) - k_{\phi}(J^{+}J - I)\phi \tag{155}$$

where J is the manipulator Jacobian matrix, J^{+} is its damped pseudoinverse, k_x is a gain for the position error, and k_{ϕ} is a gain for the joint error.

5.2.3 Single Arm and Camera Experiments

We implement this approach on a Schunk LWA4 manipulator with SDH end-effector, see Fig. 68, and use a Logitech C920 webcam to track the robot and objects. The Schunk LWA4 has seven degrees of freedom and uses harmonic drives, which enable repeatable positioning *precision* of $\pm 0.15\,\text{mm}$ [72]. However, absolute positioning *accuracy* is subject to encoder offset calibration and link rigidity. In practice, we achieve $\pm 1\,\text{cm}$ accuracy when using only the joint encoders for feedback. The Logitech C920 provides a resolution of

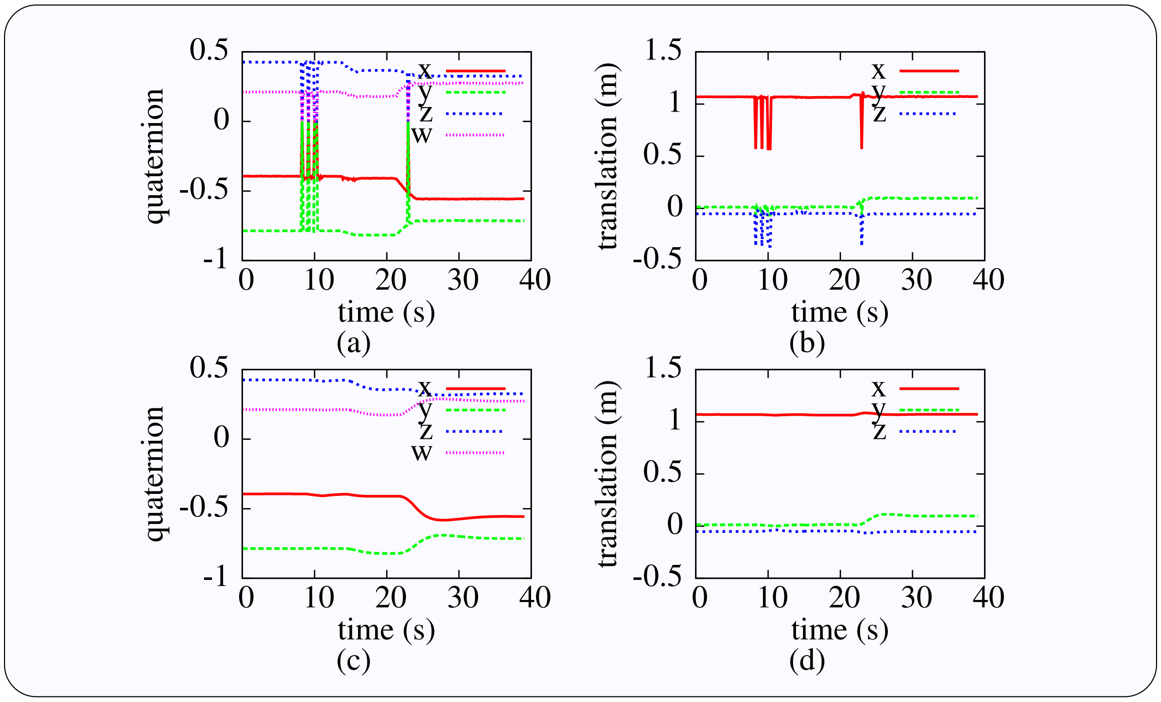


Figure 80: Registration while camera is bumped (8 s), rotates (15 s) and translated (24 s). camera is bumped. (a)-(b) registration from raw visual pose estimates of one feature. Contains many outliers. (c)-(d) filtered registration. Outliers and noise eliminated.

1920x1080 at 15 frames per second. To measure ground-truth distances, we used a Bosch DLR165 laser rangefinder and a Craftsman 40181 vernier caliper.

We initially test the convergence and resistance of our approach while moving the camera. With the camera mounted on a tripod, we compute the filtered registration while the camera is perturbed, rotated, and translated.

The resulting registrations under moving camera are plotted in Fig. 80. The visual pose estimates contain frequent outliers in addition to a small amount of noise. The filtered registration removes the outliers and converges within 5 s.

To demonstrate the suitability of this approach for manipulation tasks, we test the positioning accuracy attainable with this online registration. As shown in Fig. 81, we place a marker on a table, measure linear distance to the marker with a laser ranger, servo the end-effector to the visually estimated marker position using the control loop in Fig. 78, and measure the distance to the end-effector which should be directly over the marker.

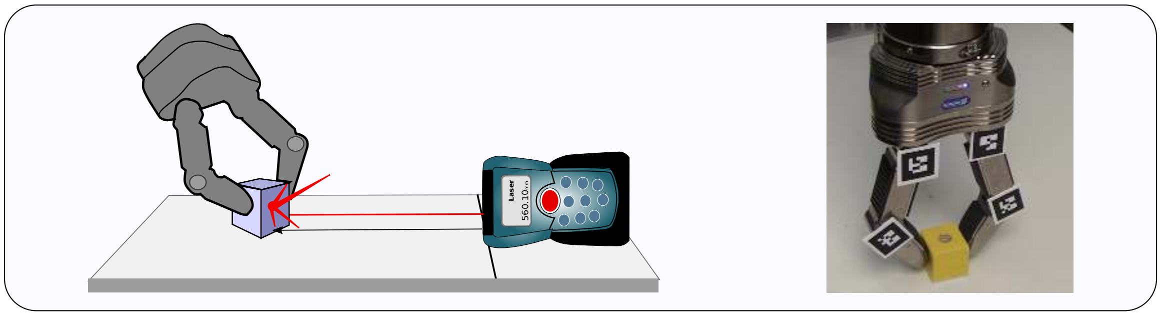


Figure 81: Experimental setup for evaluating the positioning accuracy during camera registration. A cube was placed on a marker and the distance to a laser ranger was captured. Subsequently, the cube was placed in the hand of the robot, which, then, servoed to the position of the marker. Again, the distance was measured using the laser ranger.

The resulting position accuracy achievable with online registration is summarized in Table 7. For an ideal camera placement with close, direct view of the end-effector (i.e. the angle δ between the camera and the markers is 45° or less), positioning accuracy is in the submillimiter range. Larger camera distances and angles, resulted in positioning error of $1-2$ mm.

Finally, we test the pre-grasp positioning accuracy of this method as shown in Fig. 82. We place an object, in particular, a cup, at a variety of locations on the table, servo the end-effector to the visually detected object position using the control loop in Fig. 78, and then measure the distance of each finger to the object using a vernier caliper.

The results of the pre-grasp positioning are summarized in Table 8. A small number of trials resulted in centimeter-level error for objects placed near the edge of the image frame.

Table 7: Positioning experiment results. Average and standard deviation [mm] of measured difference between commanded position and object location.

Setup	Average	Stdev
$\delta \leq 45°$	0.5mm	0.52mm
$\delta > 45°$	1.5mm	1.26mm

Data	Average	Stdev
All	5.8 mm	8.5 mm
Inliers	3.3 mm	2.3 mm

Table 8: Pre-grasp experiment results. Average and standard deviation [mm] of measured difference between object and end-effector position

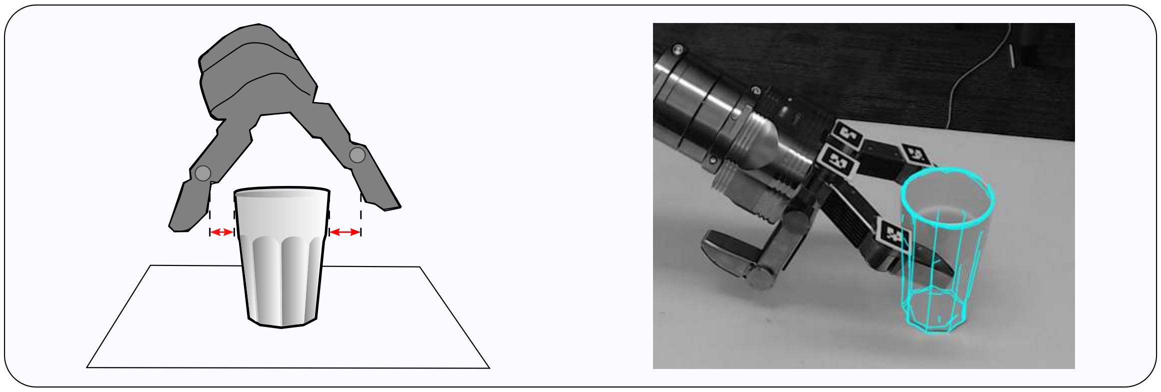


Figure 82: Pre-grasp experiment: using the introduced camera registration, the open robot hand is servoed to the position of a glass. The distances between the fingers and the glass are then measured. Since the glass is rotationally symmetric, the distances of both used robot fingers should be identical in the ideal case.

Ommitting these outliers, the average positioning error of the pre-grasp configuration was 3.3 mm.

5.3 *Online Registration of Multiple Arms and Cameras*

Bimanual manipulation requires accurate coordination of both end-effectors. To perform smooth and accurate bimanual manipulation, we extend the single camera and arm registration to include (1) visual tracking of the manipulators, (2) co-estimation of poses for cameras and end-effectors using a the special Euclidean group median and extended Kalman filter, and (3) continuous geometric interpolation on the special Euclidean group. The key insight is to combine perception and control online, using the robot body frame as a reference.

5.3.1 Asynchronous Pose Co-Estimation

Each camera image provides pose measurements for visible end-effector features. To reduce estimation latency, we process and filter the measurements from each camera asynchronously as they arrive rather than collecting images from all cameras at a fixed timestep.

The kinematic chain through the manipulator, feature, and camera is defined as:

$$ {}^{b}\mathcal{S}_{w_i} \otimes {}^{w_i}\mathcal{S}_{w_i'} \otimes {}^{w_i'}\mathcal{S}_{f_p} = {}^{b}\mathcal{S}_{c_j} \otimes {}^{c_j}\mathcal{S}_{f_p} \qquad (156) $$

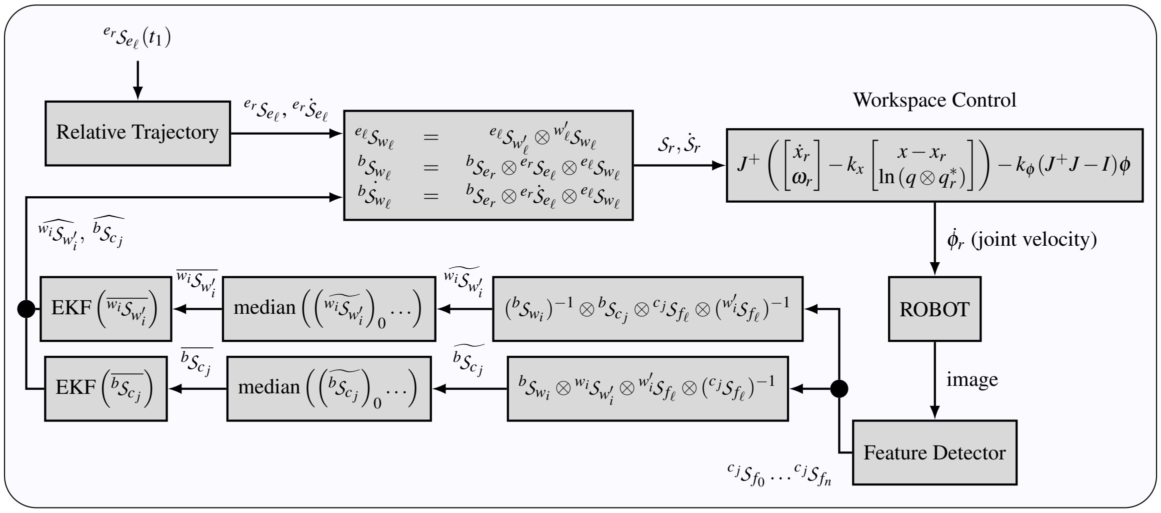

${}^{e_r}S_{e_\ell}(t_1)$
Relative Trajectory
${}^{e_r}S_{e_\ell}, {}^{e_r}\dot{S}_{e_\ell}$
${}^{e_\ell}S_{w_\ell} = {}^{e_\ell}S_{w'_\ell} \otimes {}^{w'_\ell}S_{w_\ell}$
${}^{b}S_{w_\ell} = {}^{b}S_{e_r} \otimes {}^{e_r}S_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell}$
${}^{b}\dot{S}_{w_\ell} = {}^{b}S_{e_r} \otimes {}^{e_r}\dot{S}_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell}$
$S_r, \dot{S}_r$
Workspace Control
$J^+\left(\begin{bmatrix}\dot{x}_r\\ \omega_r\end{bmatrix} - k_x\begin{bmatrix}x - x_r\\ \ln(q \otimes q_r^*)\end{bmatrix}\right) - k_\phi(J^+J - I)\phi$
$\dot{\phi}_r$ (joint velocity)
ROBOT
image
Feature Detector
${}^{c_j}S_{f_0} \ldots {}^{c_j}S_{f_n}$
$\widetilde{{}^{w_i}S_{w'_i}}, \widetilde{{}^{b}S_{c_j}}$
EKF $\left(\overline{{}^{w_i}S_{w'_i}}\right)$
$\overline{{}^{w_i}S_{w'_i}}$
median $\left(\left(\widetilde{{}^{w_i}S_{w'_i}}\right)_0 \ldots\right)$
$\widetilde{{}^{w_i}S_{w'_i}}$
$({}^{b}S_{w_i})^{-1} \otimes {}^{b}S_{c_j} \otimes {}^{c_j}S_{f_\ell} \otimes ({}^{w'_i}S_{f_\ell})^{-1}$
EKF $\left(\overline{{}^{b}S_{c_j}}\right)$
$\overline{{}^{b}S_{c_j}}$
median $\left(\left(\widetilde{{}^{b}S_{c_j}}\right)_0 \ldots\right)$
$\widetilde{{}^{b}S_{c_j}}$
${}^{b}S_{w_i} \otimes {}^{w_i}S_{w'_i} \otimes {}^{w'_i}S_{f_\ell} \otimes ({}^{c_j}S_{f_\ell})^{-1}$

Figure 83: Block diagram of the control system. 3D feature poses ${}^{c_j}S_{f_p}$ are detected from visual data. Instantaneous wrist offsets $\widetilde{{}^{w_i}S_{w'_i}}$ and camera registrations $\widetilde{{}^{b}S_{c_j}}$ are computed. Then the median of these poses is taken over a sliding window and subsequently Kalman-filtered. The filtered poses are used to track a relative left-right workspace trajectory, and the Jacobian damped-least squares gives the reference joint velocities $\dot{\phi}_r$.

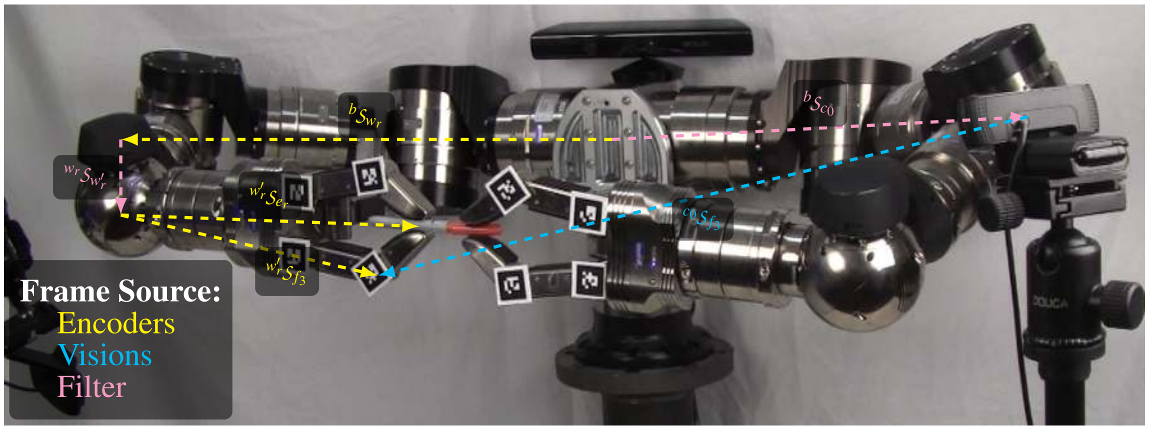


Figure 84: Kinematic frames for one arm, camera, and feature.

where ${}^{b}\mathcal{S}_{w_i}$ is the encoder-measured pose of wrist i in body frame, ${}^{w_i}\mathcal{S}_{w'_i}$ is the estimated offset pose of wrist i, ${}^{w'_i}\mathcal{S}_{f_p}$ is the encoder-measured transform from wrist i to feature p on the hand, ${}^{b}\mathcal{S}_{c_j}$ is the estimated pose registration of camera j, and ${}^{c_j}\mathcal{S}_{f_p}$ is the visually-measured pose feature p in camera j. For a depiction of the setup see Fig. 84.

Based on (156), we produce measurements for wrist offset ${}^{w_i}\mathcal{S}_{w'_i}$ and camera registration ${}^{b}\mathcal{S}_{c_j}$:

$$\widetilde{{}^{w_i}\mathcal{S}_{w'_i}} = ({}^{b}\mathcal{S}_{w_i})^{-1} \otimes \widehat{{}^{b}\mathcal{S}_{c_j}} \otimes {}^{c_j}\mathcal{S}_{f_p} \otimes ({}^{w'_i}\mathcal{S}_{f_p})^{-1} \tag{157}$$

$$\widetilde{{}^{b}\mathcal{S}_{c_j}} = {}^{b}\mathcal{S}_{w_i} \otimes \widehat{{}^{w_i}\mathcal{S}_{w'_i}} \otimes {}^{w'_i}\mathcal{S}_{f_p} \otimes ({}^{c_j}\mathcal{S}_{f_p})^{-1} \tag{158}$$

where $\widetilde{{}^{w_i}\mathcal{S}_{w'_i}}$ is the wrist offset measurement from this image and feature, $\widetilde{{}^{b}\mathcal{S}_{c_j}}$ is the camera registration measurement, $\widehat{{}^{w_i}\mathcal{S}_{w'_i}}$ is the currently estimated wrist offset, and $\widehat{{}^{b}\mathcal{S}_{c_j}}$ is the currently estimated camera pose.

We apply median and extended Kalman filtering in the special Euclidean group $\mathscr{SE}(3)$ to the measurements for wrist offset $\widetilde{{}^{w_i}\mathcal{S}_{w'_i}}$ and camera registration $\widetilde{{}^{b}\mathcal{S}_{c_j}}$, similar to the approach in [40]. First, to reject outliers, we compute the median measurement over a sliding time window. Then, we use an extended Kalman Filter over time to compute optimal pose estimates under a Gaussian noise assumption.

5.3.2 Control: Bimanual Workspace Trajectories

To perform smooth, bimanual motion, we compute a relative workspace trajectory between the two manipulators, transform the relative pose and velocity of the trajectory to the body frame, then compute joint velocities using the Jacobian damped least squares pseudoinverse.

We compute a relative trajectory trajectory for the two end-effectors using the spherical parabolic blends to provide a straight-line, constant-axis, and continuous-velocity workspace path for the end-effector.

From the relative reference pose ${}^{e_r}\mathcal{S}_{e_\ell}$ and velocity ${}^{e_r}\dot{\chi}_{e_l}$ between the left and right end-effectors, we control the left arm in workspace, by first converting the relative pose and velocity to the body frame, then computing the Jacobian damped-least-squares inverse kinematics solution.

The left-arm wrist pose ${}^{b}S_{w_\ell}$ follows directly from the kinematic chain through the right arm:

$$\begin{aligned} {}^{b}S_{w_\ell} &= {}^{b}S_{e_r} \otimes {}^{e_r}S_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell} \\ {}^{e_\ell}S_{w_\ell} &= {}^{e_\ell}S_{w'_\ell} \otimes {}^{w'_\ell}S_{w_\ell} \end{aligned} \tag{159}$$

Next, we compute the body-frame feedforward reference velocity, ${}^{a}\dot{\chi}_b$. Since there is only one changing frame, ${}^{e_r}S_{e_\ell}$, we could find the corresponding body frame motion by rotating the velocity. However, the typical computation is notationally cumbersome [37, p140].[3] Instead, we find an elegant and more general solution by merely taking the derivative of the pose:

$$\begin{aligned} {}^{b}S_{w_\ell} &= {}^{b}S_{e_r} \otimes {}^{e_r}S_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell} \\ \Rightarrow {}^{b}\dot{S}_{w_\ell} &= {}^{b}\cancelto{0}{\dot{S}_{e_r}} \otimes \left({}^{e_r}S_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell}\right) + {}^{b}S_{e_r} \otimes \frac{d}{dt}\left({}^{e_r}S_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell}\right) \\ \Rightarrow {}^{b}\dot{S}_{w_\ell} &= {}^{b}S_{e_r} \otimes \left({}^{e_r}S_{e_\ell} \otimes \cancelto{0}{{}^{e_\ell}\dot{S}_{w_\ell}} + {}^{e_r}\dot{S}_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell} \right) \\ \Rightarrow {}^{b}\dot{S}_{w_\ell} &= {}^{b}S_{e_r} \otimes {}^{e_r}\dot{S}_{e_\ell} \otimes {}^{e_\ell}S_{w_\ell} \end{aligned} \tag{160}$$

where $\cancelto{0}{S}$ indicates that S cancels to zero, and we assume the right arm and left fingers are stationary ($0 = {}^{b}\dot{S}_{e_r} = {}^{e_\ell}\dot{S}_{w_\ell}$). Note that relative motion with both arms moving could be computed by including the nonzero derivative ${}^{b}\dot{S}_{e_r}$ in the computation.

[3] The complexity of the velocity transformation notation in [37, p140] stems from its representation using Gibbs's vector calculus which decouples the quaternion multiplication into separate dot and cross products. Hamilton's and Study's classical quaternion and dual quaternion notation is simpler and more elegant for this kinematic computation. A similar computation is also possible using transformation matrices and their derivatives, but these matrices are more difficult to normalize than quaternions, increasing numerical error.

Velocity and the dual quaternion derivative are related as follows:

$$\begin{aligned}\frac{d\mathbb{R}(S)}{dt} &= \frac{1}{2}\omega \otimes \mathbb{R}(S) \\ \frac{d\mathbb{D}(S)}{dt} &= \frac{1}{2}\left(\dot{x} \otimes \mathbb{R}(S) + x \otimes \frac{d\mathbb{R}(S)}{dt}\right)\end{aligned} \tag{161}$$

where $\mathbb{R}(S)$ is the real part of S, $\mathbb{D}(S)$ is the dual part of S, ω is rotational velocity, and x is translation.

Finally, we compute reference joint velocities using the Jacobian damped least squares with a nullspace projection to keep joints near the zero position:

$$\dot{\phi}_r = J^+ \left(\begin{bmatrix} \dot{x}_r \\ \omega_r \end{bmatrix} - k_x \begin{bmatrix} x - x_r \\ \ln\left(q \otimes q_r^*\right) \end{bmatrix} \right) - k_\phi (J^+ J - I)\phi \tag{162}$$

where x is the actual translation, q is the actual orientation quaternion, x_r is the reference translation, q_r is the reference orientation quaternion, ω is the actual rotational velocity, ω_r is the reference rotational velocity, k_x is the workspace position error gain, k_ϕ is the null-space projection gain, and ϕ is the configuration. We then use joint-level velocity control to track the reference joint velocities $\dot{\phi}_r$. A block diagram depicting the components of the control system and their interplay can be found in Fig. 78.

5.3.3 Multiple Arm and Camera Experiments

We implement this approach on a pair of Schunk LWA4 manipulators with SDH end-effectors, and use a pair Logitech C920 webcams to track the robot and objects. Our estimation and control software is implemented as a distributed system using the Ach real-time communication library [48]. The Schunk LWA4 has seven degrees of freedom and uses harmonic drives, which enable repeatable positioning *precision* of $\pm 0.15\,\text{mm}$ [72]. However, absolute positioning *accuracy* is subject to encoder offset calibration and link rigidity. In practice, we achieve $\pm 15\,\text{mm}$ accuracy when using only the joint encoders for feedback, as can be seen in Fig. 85. The Logitech C920 provides a resolution of 1920x1080 at 15 frames per second. To measure ground-truth distances, we use a ruler and meter-stick.

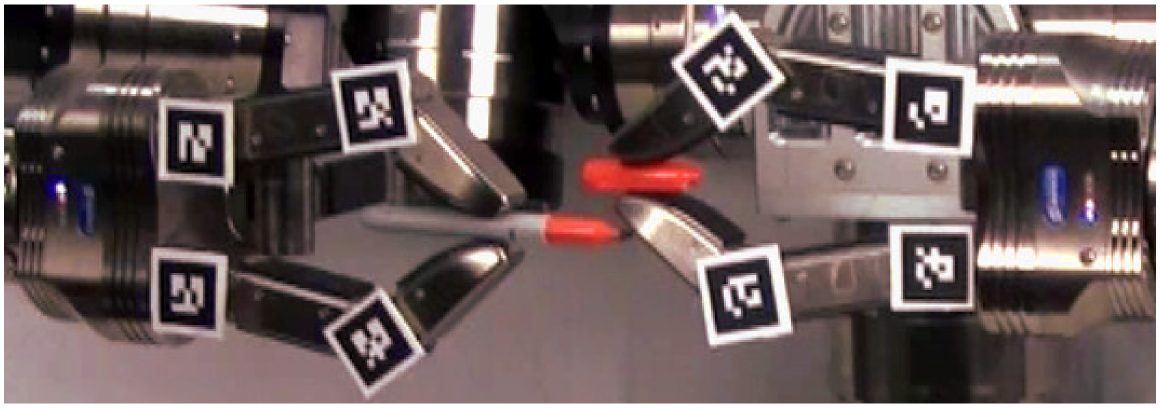


Figure 85: Manipulation error using only encoders for position feedback. Without using visual feedback, there is a 15 mm relative positioning error between the two end-effectors.

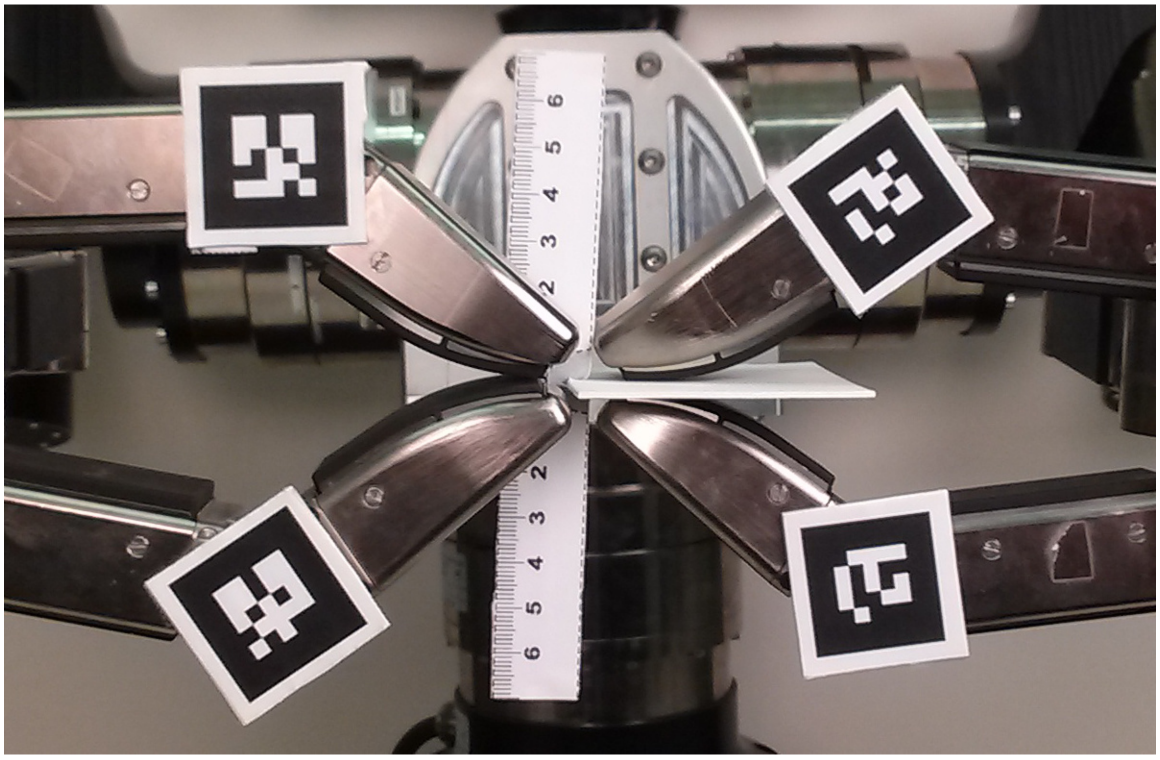


Figure 86: Testing relative positioning accuracy by aligning the end-effectors. Incorporating visual feedback and online registration reduces manipulation error from 15 mm to ≈ 2 mm.

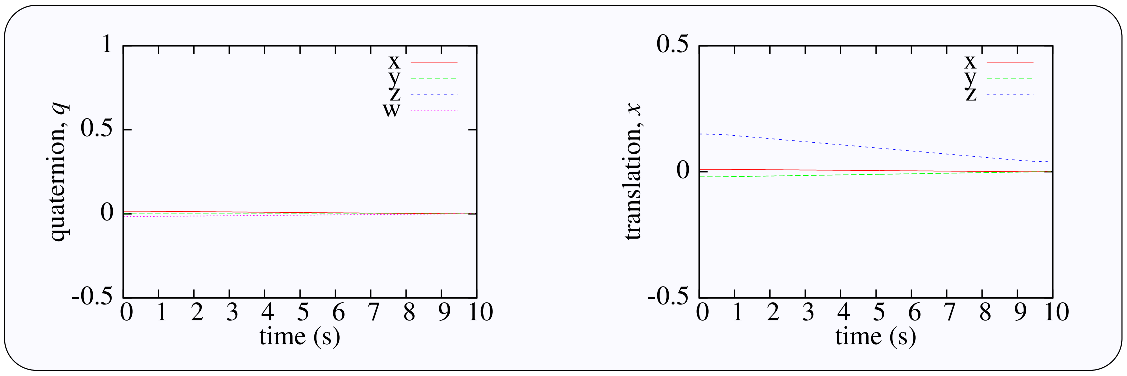


Figure 87: Relative trajectory of ${}^{e_r}S_{e_\ell}$ between left and right end-effectors for pen-capping. The trajectory has constant acceleration, constant velocity, and constant deceleration segments.

Table 9: Positioning Test Results (mm)

	Mean		Std. Dev.	
	encoder	**visual**	encoder	**visual**
No Offset	16.5	2.2	0.5	0.94
shoulder: 15°	155	2.8	0.6	0.78
shoulder: 30°	280	1.3	0	0.95
shoulder & elbow: 15°	240	0.95	0	1.1

To test the relative positioning accuracy of our implementation, we servo the end-effectors to a reference zero relative alignment, Fig. 86, and then measure the actual relative error between the two end-effectors. We conduct this test using only encoder feedback, then with visual feedback. We also repeat the test injecting encoder error of 15° at the initial shoulder joint, 30° at the shoulder, and 15° at both the shoulder and elbow. The results of this test are summarized in Table 9.

In addition, we use this method to perform the pen-capping task show in Fig. 84 and the object hand off task shown in Fig. 88. The relative trajectory of ${}^{e_r}S_{e_\ell}$ for the pen-capping task is plotted in Fig. 87

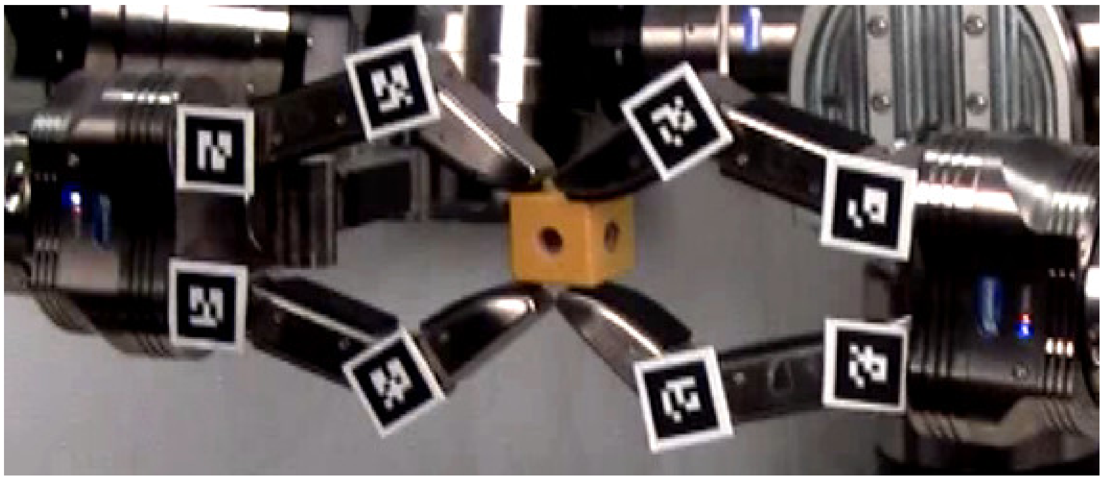


Figure 88: An object hand-off task.

5.4 Discussion

There are a number of error sources we must handle in this system. For the kinematics, error from encoder offsets in the arm, imprecise link lengths, and flexing of links all contribute inaccurate kinematic pose estimates. For perception, error from inaccurate camera intrinsics, imprecise fiducial sizes, offsets in object models, and noise in the image all contribute to error in visual pose estimates. To achieve accurate manipulation, we must account for these potential sources of error.

The key point of the servo loops in Fig. 78 and Fig. 83 is that we depend not on minimizing *absolute* error, but on minimizing *relative* error. We are minimizing error between end-effector pose $\mathcal{S}_e$ and target pose $\mathcal{S}_o$. Because we continually update the camera registration, we effectively minimize this error in the image. As long as there is distance between camera frame poses ${}^C\mathcal{S}_e$ and ${}^C\mathcal{S}_o$, we will move the end-effector towards the target, and as long as the visual distance estimate is zero when we reach the target, the arm will stop at the target. Thus, even if there is absolute registration error due to, e.g., unmodeled lens distortion, it is only necessary that relative error between visual estimates of the end-effector and target be small and converge to zero. The relative error between end-effector and target is crucial in manipulation, and our technique is well suited to minimizing this error.

The position of the tracked features on the robot has an important effect on error correction. Kinematic errors between the robot body origin and the tracked features, e.g., due to flex or encoder offsets, are incorporated into the camera registration and handled through the servo loop. Error between the observed features and the end-effector cannot be corrected. Thus, it is better to track features as close to the end-effector as possible. Consequently, we placed the fiducial markers on the fingers of the SDH end-effector.

The principal challenge in the implementation stems from observing the robot pose using small, $\approx 3\,\mathrm{cm}$, markers. While marker translation is reliably detected, outliers in orientation are frequent. Ample lighting improves detection but does not eliminate outliers. The median pose, (129)-(131), was effective at eliminating outliers from visual estimates. Alternative methods for combining orientations estimates include Davenport's q-method [126] and the Huber loss function [90]. In contract to these other methods, the median has no parameters such as thresholds which require adjustment. Thus, it is especially suited to this online registration application where outlier frequency may vary depending on camera placement, lighting, etc. A potential challenge is that the direct computation of (129) leads to an $O(n^2)$ algorithm in the number of orientations. However, for the small number of poses we consider at each step here, the computation time is negligible. On a Xeon E5-1620 CPU, computing the median of 32 orientations requires $30\,\mu\mathrm{s}$.

One source of error for manipulation that we do not address is error in grasping. Because we track only the robot hand, any error in the relative pose between the hand and grasped object is not corrected. In reality, when grasping an object, the *object* itself becomes the robot's end-effector. Thus, to accurately manipulate in-hand objects, it would be better to track the objects themselves. Since a grasped object is likely to be partially occluded, model-based tracking such as [31], which is robust to occlusions, is a potential approach.

A crucial additional consideration in manipulation is force and tactile sensing. Using visual feedback without force and tactile sensing already reduces the error to a few millimeters and allows the robot to perform tasks such as pen capping and object hand-off. However, considering the generated contact forces during the manipulation would further improve performance and allow even more accurate operation, in particular during the post-contact phase. This is a key area for improvement in this approach.

CHAPTER VI

MODELING AND PROGRAMMING CONCURRENCY

While the linguistic approach used so far operates serially, we must also consider concurrency of the underlying robot hardware, which contains many networked components operating in parallel. In addition, it is desirable to limit the potential scope of errors which occur outside the formally verifiably linguistic framework. We can address both of these issues by adopting a multi-process software design. In a real-time, multi-process system, it is critical to communicate the latest data sample with minimum latency. There are many communication approaches intended for both general purpose and real-time needs [89, 136, 149, 158, 167]. Typical methods focus on reliable communication or network-transparency and accept a trade-off of increased message latency or the potential to discard newer data. By focusing instead on the specific case of real-time communication on a single host, we reduce communication latency and guarantee access to the latest sample. We present a new Interprocess Communication (IPC) library, *Ach*,[1] which addresses this need, and discuss its application for real-time, multiprocess control on three humanoid robots (Fig. 89).

There are several design decisions that influenced this robot software and motivated development of the Ach library. First, to utilize decades of prior development and engineering, we implement our real-time system on top of a POSIX-like Operating System (OS)[2]. This provides us with high-quality open source platforms such as GNU/Linux and a wide variety of compatible hardware and software. Second, because safety is critical for these

[1]Ach is available at `http://www.golems.org/projects/ach.html`. The name "Ach" comes from the common abbreviation for the motor neurotransmitter Acetylcholine and the computer networking term "ACK."

[2]POSIX is IEEE standard 1003.1 for a portable operating system interface. It enables software portability among supporting operating systems such as GNU/Linux, MacOSX, Solaris, and QNX.

Figure 89: Hubo, Golem Krang, and Nao: Existing Robotic Systems where Ach provides communications between hardware drivers, perception, planning, and control algorithms.

robots, the software must be robust. Therefore, we adopt a multiple process approach over a single-process or multi-threaded application to limit the potential scope of errors [162]. This implies that sampled data must be passed between OS processes using some form of Interprocess Communication (IPC). Since general purpose IPC favors older data [167] (see section 6.1), while real-time control needs the latest data, we have developed a new IPC library.

This article discusses a POSIX Interprocess Communication (IPC) library for the real-time control of physical processes such as robots, describes its application on three different humanoid platforms, and compares this IPC library with a variety of other communication methods. This library, called *Ach*, provides a message-bus or publish-subscribe communication semantics – similar to other real-time middleware and robotics frameworks [136, 149] – but with the distinguishing feature of favoring newer data over old. Ach is formally verified, efficient, and it always provides access to the most recent data sample. To our knowledge, these benefits are unique among existing communications software.

6.1 Review of POSIX IPC

POSIX provides a rich variety of IPC that is well suited for general purpose information processing, but none are ideal for real-time robot control. Typically, a physical process such as a robot is viewed as a set of continuous, time-varying *signals*. To control this physical process with a digital computer, one must *sample* the signal at discrete time intervals and perform control calculations using the sampled value. To achieve high-performance control of a physical system, we must process the latest sample with minimum latency. This differs from the requirements of general computing systems which focus on throughput over latency and favor prior data over latter data. Thus, for robot control, it is better to favor new data over old data whereas nearly all POSIX IPC favors the old data. This problem is typically referred to as *Head of Line (HOL) Blocking*. The exception to this is POSIX shared memory. However, synchronization of shared memory is a difficult programming problem, making the typical and direct use of POSIX shared memory unfavorable for developing robust systems. Furthermore, some parts of the system, such as logging, may need to access older samples, so this also should be permitted at least on a best-effort basis. Since no existing implementation satisfied our requirements for low-latency exchange of most-recent samples, we have developed a new open source IPC library.

The three main types of POSIX IPC are streams, datagrams, and shared memory. We review each of these types and consider why these general-purpose IPC mechanisms are not ideal for real-time robot control. Table 10 contrasts the response of each method to a full buffer, and Table 11 summarizes the pros and cons of each method. A thorough survey of POSIX IPC is provided in [167].

6.1.1 Streams

Stream IPC includes pipes, FIFOs, local-domain stream sockets, and TCP sockets. These IPC mechanisms all expose the *file* abstraction: a sequence of bytes accessed with `read` and `write`. All stream-based IPC suffers from the HOL blocking problem; we must read all the old bytes before we see any new bytes. Furthermore, to prevent blocking of the reading or writing process, we must resort to more complicated *nonblocking* or *asynchronous I/O*.

6.1.2 Datagrams

6.1.2.1 Datagram Sockets

Datagram sockets perform better than streams in that they are less likely to block the sender. Additionally, some types of datagram sockets can *multicast* packets, efficiently transmitting them to multiple receivers. However, datagram sockets give a variation on the HOL blocking problem where newer messages are simply lost if a buffer fills up. This is unacceptable since we require access to the most recent data.

6.1.2.2 POSIX Message Queues

POSIX Message Queues are similar to datagram sockets and also include the feature of message priorities. The downside is that it is possible to block if the queue fills up. Consider a process that gets stuck and stops processing its message queue. When it starts again, the process must still read or flush old messages before getting the most recent sample.

6.1.3 Shared Memory

POSIX shared memory is very fast and one could, by simply overwriting a variable, always have the latest data. However, this provides no recourse for recovering older data that may have been missed. In addition, shared memory presents synchronization issues which are notoriously difficult to solve [101], making direct shared memory use less suitable for safety critical real-time control.

The data structure which Ach most closely resembles is the circular array. Circular arrays or ring buffers are common data structures in device drivers and real-time programs, and the implementation in Ach provides unique features to satisfy our requirements for a multi-process real-time system. Typical circular buffers allow only one producer and one consumer with the view that the producer inserts data and the consumer removes it. Our robots have multiple producers and multiple consumers writing and reading a single sequence of messages. A message reader cannot *remove* a message, because some other process may still need to read it. Because of this different design requirement, Ach uses a different data structure and algorithm in order to perform real-time IPC among multiple processes.

6.1.4 Further Considerations

6.1.4.1 Nonblocking and Asynchronous IO approaches

There are several approaches that allow a single process or thread to perform IO operations across several file descriptions. Asynchronous IO (AIO) may seem to be the most appropriate for this application. However, the current implementation under Linux is not as mature as other IPC mechanisms. Methods using select/poll/epoll/kqueue are widely used for network servers. Yet, both AIO and select-based methods only mitigate the HOL problem, not eliminate it. Specifically, the sender will not block, but the receiver must read or flush the old data from the stream before it can see the most recent sample.

6.1.4.2 Priorities

To our knowledge, none of the stream or datagram forms of IPC consider the issue of process priorities. Priorities are critical for real-time systems. When there are two readers that want the next sample, we want the real-time process, such as a motor driver, to get the data and process it before a non real-time process, such as a logger, does anything.

Table 10: Full Buffer Semantics

Method	Action on full buffer
Stream	Block sender, or Error
Datagram	Drop newest message, or Error
Message Queue	Block sender, or Error
Ach	Drop oldest message

Table 11: POSIX IPC Summary, pros and cons for real-time

Method	Pro	Con	Examples
Streams	Reliable, Ordered	Head-of-Line Blocking	pipes, TCP, Local Socket
Datagrams	Multicast, no sender blocking	Full buffer blocks or discards new data	UDP, Local Socket
Message Queues	Can avoid blocking sender	Full buffer blocks or discards new data	POSIX Message Queues
Shared Memory	Fast	Last only, Synchronization issues	POSIX Shared Mem., mmap
Asynchronous I/O	No blocking	Immature, favors old data	POSIX Asynchronous I/O
Nonblocking I/O	No blocking	Must retry, favors old data	O_NONBLOCK
Multiplexed I/O	Handles many connections	Receiver must read/discard old data	select, poll, epoll, kqueue

6.1.5 General, Real-Time, Robotics Middleware

In addition to the core POSIX IPC mechanisms, there are many messaging middlewares and robot software architectures. However, these are either not Open Source or not ideal for our multi-process real-time domain. Many of these approaches build on an underlying POSIX IPC method, inheriting that method's strengths and weaknesses. Furthermore, our benchmark results for some of these methods (see Fig. 95) show that they impose noticeable overhead compared to the underlying kernel IPC.

Most middleware addresses general purpose rather than real-time communication. The Message Passing Interface (MPI) is ubiquitous in high-performance computing, but its focus is on maximizing message throughput for networked clusters [73]. The robot control domain centers around minimizing sample latency on a single host. The Advanced Message Queuing Protocol (AMQP) [184] is a network message distribution middleware focused on business applications; it does not address low-latency real-time systems. ZeroMQ provides IPC based on TCP and local-domain sockets which have the HOL blocking condition. Remote Procedure Call (RPC) methods such as ONC RPC [164] allow synchronous point-to-point communication but they do not directly allow efficient communication between multiple senders and receivers and also do not address HOL blocking.

Several frameworks and middleware focus on real-time control or robotics. The Orocos Real-Time Toolkit [22] and NAOqi [2] are two architectures for robot control, but they do not meet our requirements for flexible IPC. iRobot's Aware2.0 [92] is not open source, and Microsoft Robotics Studio [128] is not open source and does not run on POSIX systems. ROS [149] provides open source TCP and UDP message transports, which suffer from the aforementioned HOL blocking problem. CORBA [137] provides object-oriented remote procedure call, an event notification service, and underlies the OpenRTM middleware [10]; our benchmark results (see Fig. 95) show that TAO CORBA [158], a popular implementation, gives poor messaging performance compared to alternatives.

In contrast, Data Distribution Service [136] and LCM [89] are publish-subscribe network protocols. LCM is based on UDP multicast which efficiently uses network bandwidth to communicate with multiple subscribers. However, UDP does drop newer packets when the receiving socket buffer is full. These protocols may be complementary to the efficient and formally verified IPC we present here.

In conclusion, none of these middlewares met our needs for an open source, lightweight, and non-HOL blocking IPC. However, the design of Ach facilitates integration with some of these other frameworks (see subsection 6.3.2 and subsection 6.3.3).

6.2 The Ach IPC Library

Ach provides a message bus or publish-subscribe style of communication between multiple writers and multiple readers. A real-time system has multiple Ach channels across which individual data samples are published. Messages are sent as byte arrays, so arbitrary data may be transmitted such as floating point vectors, text, images, and binary control messages. Each channel is implemented as two circular buffers, (1) a data buffer with variable sized entries and (2) an index buffer with fixed-size elements indicating the offsets into the data buffer. These two circular buffers are written in a channel-specific POSIX shared memory file. Using this formulation, we solve and formally verify the synchronization problem exactly once and contain it entirely within the Ach library.

The Ach interface consists of the following procedures:

- `ach_create`: Create the shared memory region and initialize its data structures
- `ach_open`: Open the shared memory file and initialize process local channel counters
- `ach_put`: Insert a new message into the channel
- `ach_get`: Receive a message from the channel
- `ach_close`: Close the shared memory file

Channels must be created before they can be opened. Creation may be done directly by either the reading or writing process, or it may be done via the shell command, `ach mk channel_name`, before the reader or writer start. This is analogous to the creation of FIFOs with `mkfifo` called either as a shell command or as a C function. After the channel is created, each reader or writer must open the channel before it can get or put messages.

6.2.1 Channel Data Structure

The core data structure of an Ach channel is a pair of circular arrays located in the POSIX shared memory file, Fig. 90. It differs from typical circular buffers by permitting multiple consumers to access the same message from the channel. The data array contains variable sized elements which store the actual message frames sent through the Ach channel. The index array contains fixed size elements where each element contains both an offset into the data array and the length of that data element. A head offset into each array indicates both the place to insert the next data and the location of the most recent message frame. Each reader maintains its own offset into the index array, indicating the last message seen

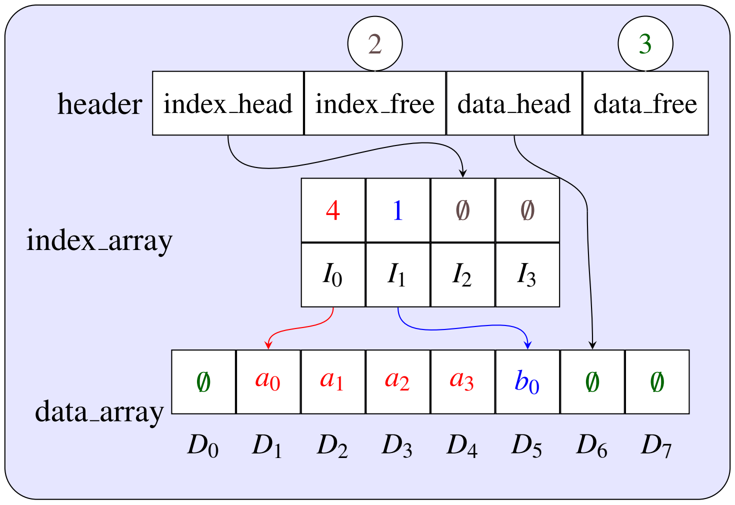


Figure 90: Logical Memory Structure for an Ach shared memory file. In this example, I_0 points to a four byte message starting at D_1, and I_1 points to a one byte message starting at D_5. The next inserted message will use index cell I_2 and start at D_6. There are two free index cells and three free data bytes. Both arrays are circular and wrap around when the end is reached.

by that reader. This pair of circular arrays allows readers to find the variable size message frames based on the index array offset and the corresponding entry in the data array.

Access to the channel is synchronized using a mutex and condition variable. This allows readers to either periodically poll the channel for new data or to wait on the condition variable until a writer has posted a new message. Using a read/write lock instead would have allowed only polling. Additionally, synchronization using a mutex prevents starvation and enables proper priority inheritance between processes, important to maintaining real-time performance.

6.2.2 Core Procedures

Two procedures compose the core of ach: `ach_put` and `ach_get`.

6.2.2.1 ach_put

The procedure `ach_put` inserts new messages into the channel. It is analogous to `write`, `sendmsg`, and `mq_send`. The procedure is given a pointer to the shared memory region for the channel and a byte array containing the message to post. There are four broad steps

```
Procedure achput
  Input: c : ach channel ;                              // shared memory file
  Input: b : byte array ;                                  // message buffer
  Input: n : integer ;                                   // length of message
  Output: status : integer ;                                  // status code
  if $n > length(c.data\_array)$ then return OVERFLOW;
  LOCK($c$); // take the mutex
  /* Get a index entry                                                   */
  if $0 = c.index\_free$ then
      $c.data\_free += c.index\_array[c.index\_head].size$;
      $c.index\_free \leftarrow 1$;
  /* Make room in data array                                             */
  $i \leftarrow (c.index\_head + c.index\_free)\ \%\ c.index\_cnt$;
  while $c.data\_free < n$ do
      $c.data\_free += c.index\_array[i].size$;
      $c.index\_free{++}$;
      $i \leftarrow (i+1)\ \%\ c.index\_cnt$;
  /* Copy Buffer                                                         */
  if $c.data\_size - c.data\_head \geq n$ then
      /* Simple Copy                                                     */
      MEMCPY($c.data\_array + c.data\_head, b, n$);
  else
      /* Wraparound Copy                                                 */
      $e \leftarrow c.data\_size - c.data\_head$;
      MEMCPY($c.data\_array + c.data\_head, b, e$);
      MEMCPY($c.data\_array, b + e, n - e$);
  /* Modify Counts                                                       */
  $c.index\_array[c.index\_head].size = n$;
  $c.index\_array[c.index\_head].offset = c.data\_head$;
  $c.data\_head \leftarrow (c.data\_head + n)\ \%\ length(c.data\_array)$;
  $c.data\_free -= n$;
  $c.index\_head \leftarrow (c.index\_head + 1)\ \%\ c.index\_cnt$;
  $c.index\_free{--}$;
  UNLOCK($c$); // release the mutex
  NOTIFY($c$); // wake readers on cond.  var.
  return OK;
```

to the procedure:

1. Get an index entry. If there is at least one free index entry, use it. Otherwise, clear the oldest index entry and its corresponding message in the data array.

2. Make room in the data array. If there is enough room already, continue. Otherwise, repeatedly

free the oldest message until there is enough room.

3. Copy the message into data array.

4. Update the offset and free counts in the channel structure.

6.2.2.2 ach_get

The procedure `ach_get` receives a message from the channel. It is analogous to `read`, `recvmsg`, and `mq_receive`. The procedure takes a pointer to the shared memory region, a storage buffer to copy the message to, the last message sequence number received, the next index offset to check for a message, and option flags indicating whether to block waiting for a new message and whether to return the newest message bypassing any older unseen messages. There are four broad steps to the procedure:

1. If we are to wait for a new message and there is no new message, then wait. Otherwise, if there are no new messages, return a status code indicating this fact.

2. Find the index entry to use. If we are to return the newest message, use that entry. Otherwise, if the next entry we expected to use contains the next sequence number we expect to see, use that entry. Otherwise, use the oldest entry.

3. According to the offset and size from the selected index entry, copy the message from the data array into the provided storage buffer.

4. Update the sequence number count and next index entry offset for this receiver.

6.3 Case Studies

6.3.1 Dynamic Balance on Golem Krang

Golem Krang, Fig. 91, is a dynamically balancing, bi-manual mobile manipulator designed and built at the Georgia Tech Humanoid Robotics Lab [169]. All the real-time control for Krang is implemented through the Ach IPC library. This approach has improved software robustness and modularity, minimizing system failures and allowing code reuse both within Krang with other projects [47] sharing the same hardware components.

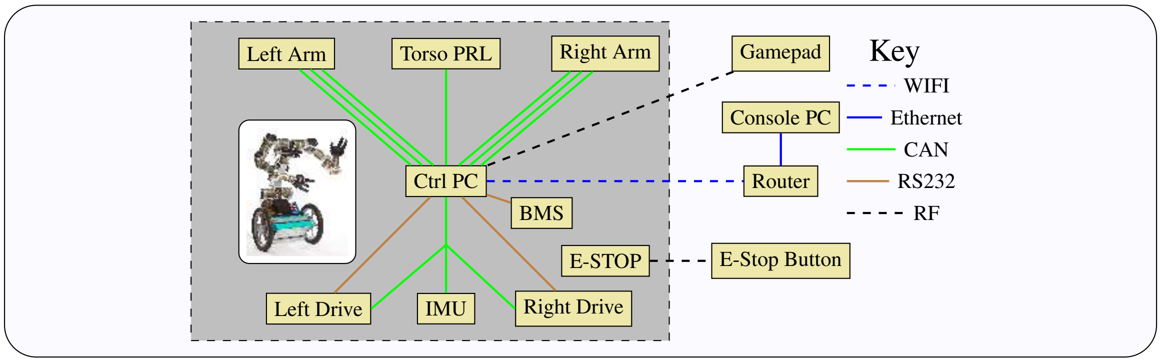


Figure 91: Block diagram of electronic components in Golem Krang. Blocks inside the dashed line are onboard and blocks outside are offboard. Control software runs on the Pentium-M Control PC under Ubuntu Linux, which communicates over eight Controller Area Network (CAN) buses to the embedded hardware. The arms are Schunk LWA3s with ATI wrist force-torque sensors and Robotiq adaptive grippers. The torso is actuated using three Schunk PRL motor modules. The wheels are controlled using AMC servo drives. The battery management system (BMS) monitors the lithium cells.

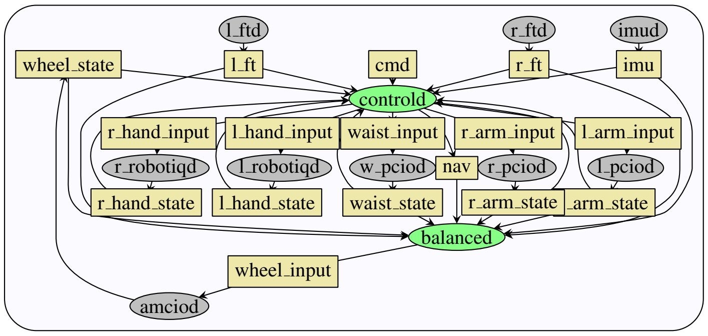


Figure 92: Block diagram of primary software components on Golem Krang. Gray ovals are user-space driver processes, green ovals are controller processes, and rectangles are Ach channels. Each hardware device, such as the IMU or LWA3, is managed by a separate driver process. Each driver process sends state messages, such as positions or forces, over a separate state channel. Devices that take input, such as a reference velocity, have a separate input channel.

The software for Krang is implemented as a collection of processes communicating over Ach channels, Fig. 92. In this design, providing a separate state Ach channel for each hardware device ensures that the current state of the robot can always be accessed through the newest messages in each of these channels. Additionally, splitting the control into separate `balanced`, for stable balancing, and `controld`, for arm control, processes promotes robustness by isolating the highly-critical balance control from other faults. This collection of driver and controller daemons communicating over Ach channels implements the real-time, kilohertz control loop for Golem Krang.

This design provides several advantages for control on Krang. The low overhead and suitable semantics of Ach communication permits real-time control under Linux using multiple processes. In several cases, Krang contains multiple identical hardware devices. The message-passing, multi-process design aids code reuse by allowing access to duplicated devices with multiple instances of the same daemon binary – two instances of the `ftd` daemon for the F/T sensors, two instances of the `robotiqd` daemon for the grippers, and three instances of the `pciod` daemon for two arms and torso. The relative independence of each running process makes this system robust to failures in non-critical components. For example, an electrical failure in a waist motor may stall the `w_pciod` process, but – without any additional code – the `balanced` controller and `amciod` driver daemons continue running independently, ensuring that the robot does not fall. Thus, Ach helps enhance the safety of this potentially dangerous robot.

6.3.2 Speed Regulation on Nao

The Aldebaran Nao is a 0.5 m, 5 kg bipedal robot with 25 degrees-of-freedom (DOF). It contains an on-board Intel Atom PC running a GNU/Linux distribution with the NAOqi framework to control the robot. User code is loaded into the NAOqi process as dynamic library modules. We used Ach to implement Human-Inspired Control [145] on the Nao [49]. The Human-Inspired Control approach achieves provably stable, human-like walking on robots by identifying key parameters in human gaits and transferring these to the robot through an optimization process. To implement this approach, real-time control software to produce the desired joint angles must run on the NAO's internal computer.

The NAOqi framework provides an interface to the robot's hardware; however, it presents some specific challenges for application development – and for the implementation of Human-Inspired Control in particular. NAOqi is slow and memory-intensive, consuming at idle 15% of available CPU time and 20% of available memory. Additionally, real-time user code must run as a callback function, which is awkward for the desired controller implementation. Using Ach to move the controller to a separate process improves the implementation.

A multi-process software design, Fig. 93, addresses these challenges with NAOqi and enhances the robustness and efficiency of Human-Inspired Control on the Nao. Each process runs independently, so an error in a non-critical process, such as `logger/debugger`, cannot affect other processes, eliminating a potential failure. The user processes can be stopped and started within only a few seconds. In contrast, NAOqi takes about 15 seconds to start. The independence of processes means NAOqi need not be restarted so long as `libamber` is unchanged. Since `libamber` is a minimal module, only interfacing with the Ach channels and accessing the Nao's hardware, it can be reused unmodified for different applications on the Nao. Different projects can run different controller processes, using Ach and `libamber` to access Nao's hardware, all without restarting the NAOqi process. In addition, using standard debugging tools such as GDB is much easier since the user code can be executed within the debugger independently of the NAOqi framework. Thus, converting the NAO's control software to a multi-process design simplified development and improved reliability.

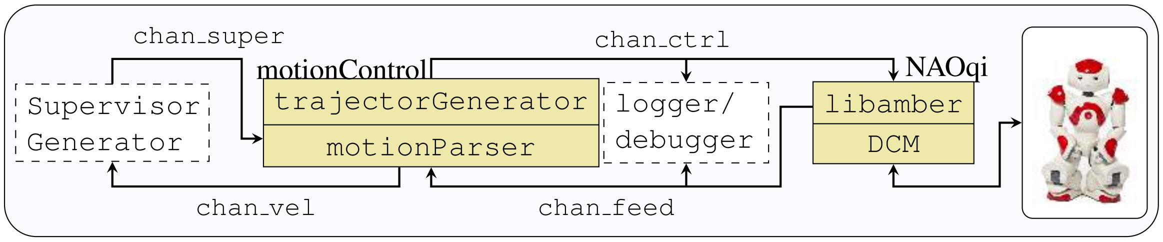


Figure 93: Block diagram of primary software components on Nao. Solid blocks are real-time processes, and dashed blocks are non-real-time processes. `NAOqi` loads the `libamber` module to communicate over Ach channels. The `motionControl` process performs feedback control while the `logger/debugger` process records data from the Ach channels. The `Supervisor Generator` process performs high-level policy generation for speed control.

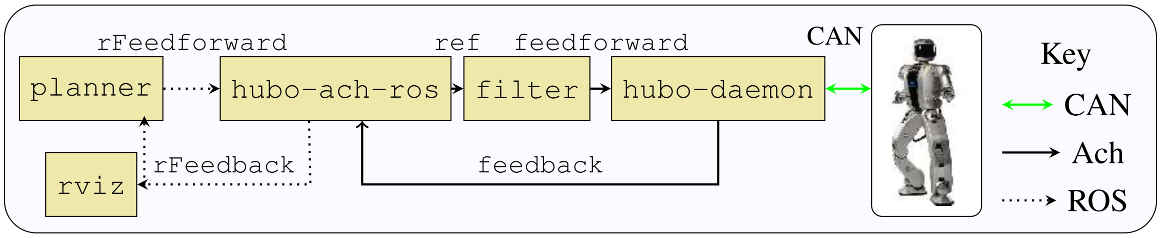


Figure 94: Block diagram of feedback loop integrating Hubo-Ach and ROS. The `planner` process computes trajectories, and the `rviz` process displays a 3D model of Hubo's current state. The `hubo-ach-ros` process bridges the Ach channels with ROS topics. The `filter` process smooths trajectories to reduce jerk. `Hubo-daemon` communicates with the embedded motor controllers.

6.3.3 Reliable Software for the Hubo2+

The Hubo2+ is a 1.3 m tall, 42 kg full-size humanoid robot, produced by the Korean Advanced Institute of Science and Technology (KAIST) and spinoff company Rainbow Inc. [30]. It has 38 DOF: six per arm and leg, five per hand, three in the neck, and one in the waist. Sensors include three-axis force-torque sensors in the wrists and ankles, accelerometers in the feet, and an inertial measurement unit (IMU). The sensors and embedded motor controllers are connected via a Controller Area Network to a pair of Intel Atom PC104+ computers.

Hubo-Ach[3] is an Ach-based interface to Hubo's sensors and motor controllers [120].

[3]Available under permissive license, `http://github.com/hubo/hubo-ach`

This provides a conventional GNU/Linux programming environment, with the variety of tools available therein, for developing applications on the Hubo. It also links the embedded electronics and real-time control to popular frameworks for robotics software: ROS [149], OpenRAVE, and MATLAB.

Reliability is a critical issue for software on the Hubo. As a bipedal robot, Hubo must constantly maintain dynamic balance; if the software fails, it will fall and break. A multi-process software design improves Hubo's reliability by isolating the critical balance code from other non-critical functions, such as control of the neck or arms. For the high-speed, low-latency communications and priority access to latest sensor feedback, Ach provides the underlying IPC.

Hubo-Ach handles CAN bus communication between the PC and embedded electronics. Because the motor controllers synchronize to the control period in a *phase lock loop* (PLL), the single `hubo-daemon` process runs at a fixed control rate. The embedded controllers lock to this rate and linearly interpolate between the commanded positions, providing smoother trajectories in the face of limited communication bandwidth. This communication process also avoids bus saturation; with CAN bandwidth of 1 Mbps and a 200Hz control rate, `hubo-daemon` utilizes 78% of the bus. `Hubo-daemon` receives position targets from a `feedforward` channel and publishes sensor data to the `feedback` channel, providing the direct software interface to the embedded electronics. Fig. 94 shows an example control loop integrating Hubo-Ach and ROS.

Hubo-Ach is in use for numerous projects at several research labs. Users include groups at MIT, WPI, Ohio State, Purdue, Swarthmore College, Georgia Tech, and Drexel University. These projects primarily revolve around the DARPA Robotics Challenge (DRC)[4] team DRC-Hubo[5]. The DRC includes rough terrain walking, ladder climbing, valve turning, vehicle ingress/egress and more.

[4]`http://www.theroboticschallenge.org/`

[5]`http://drc-hubo.com/`

Hubo-Ach helps the development of reliable, real-time applications on the Hubo. Separating software modules into different processes increases system reliability. A failed process can be independently restarted, minimizing the chance of damage to the robot. In addition, the controllers can run at fast rates because Ach provides high-speed, low-latency communication with `hubo-daemon`. Hubo-Ach provides a C API callable from high-level programming languages, and it integrates with popular platforms for robot software such as ROS and MATLAB, providing additional development flexibility. Hubo-Ach is a validated and effective interface between the mechatronics and the software control algorithms of the Hubo full-size humanoid robot.

6.4 Performance and Discussion

6.4.1 Formal Verification

We used the SPIN Model Checker [84] to formally verify Ach. Formal verification is a method to enhance the reliability of software by first *modeling* the operation of that software and then *checking* that the model adheres to a specification for performance. SPIN models concurrent programs using the Promela language. Then, it enumerates all possible world states of that model and ensures that each state satisfies the given specification. This can detect errors that are difficult to find in testing. Because process scheduling is non-deterministic, testing may not reveal errors due to concurrent access, which could later manifest in the field. However, because model checking enumerates all possible process interleavings, it is guaranteed to detect concurrency errors in the model.

We verified the `ach_put` and `ach_get` procedures using SPIN. Our model for Ach checks the consistency of channel data structures, ensures proper transmission of message data, and verifies freedom from deadlock. Model checking verifies these properties for all possible interleavings of `ach_put` and `ach_get`, which would be practically impossible to achieve through testing alone. By modeling the behavior of Ach in Promela and verifying its performance with SPIN, we eliminated errors in the returned status codes and simplified our implementation, improving the robustness and simplicity of Ach.

6.4.2 Benchmarks

We provide benchmark results of message latency for Ach and a variety of other kernel IPC methods as well as the LCM, ROS, and TAO CORBA middleware[6]. Latency is often more critical than bandwidth for real-time control as the amount of data per sample is generally small, e.g., state and reference values for several joint axes. Consequently, the actual time to copy the data is negligible compared to other sources of overhead such as process scheduling. The benchmark application performs the following steps:

1. Initialize communication structures
2. `fork` sending and receiving processes
3. Sender: Post timestamped messages at the desired frequency
4. Receivers: Receive messages and record latency of each messaged based on the timestamp

We ran the benchmarks under two kernels: Linux PREEMPT_RT and Xenomai. PREEMPT_RT is a patch to the Linux kernel that reduces latency by making the kernel fully preemptible. Any Linux application can request real-time priority. Xenomai runs the real-time Adeos hypervisor alongside a standard Linux kernel. Real-time applications communicate through Adeos via an API *skin* such as RTDM, ITRON, or POSIX; these applications are not binary compatible with Linux applications, though the POSIX skin is largely source compatible.

Fig. 95 shows the results of the benchmarks, run on an Intel Xeon 1270v2 under both Linux PREEMPT_RT and Xenomai's POSIX skin. We used Linux 3.4.18 PREEMPT_RT,

[6]Benchmark code available at `http://github.com/ndantam/ipcbench`

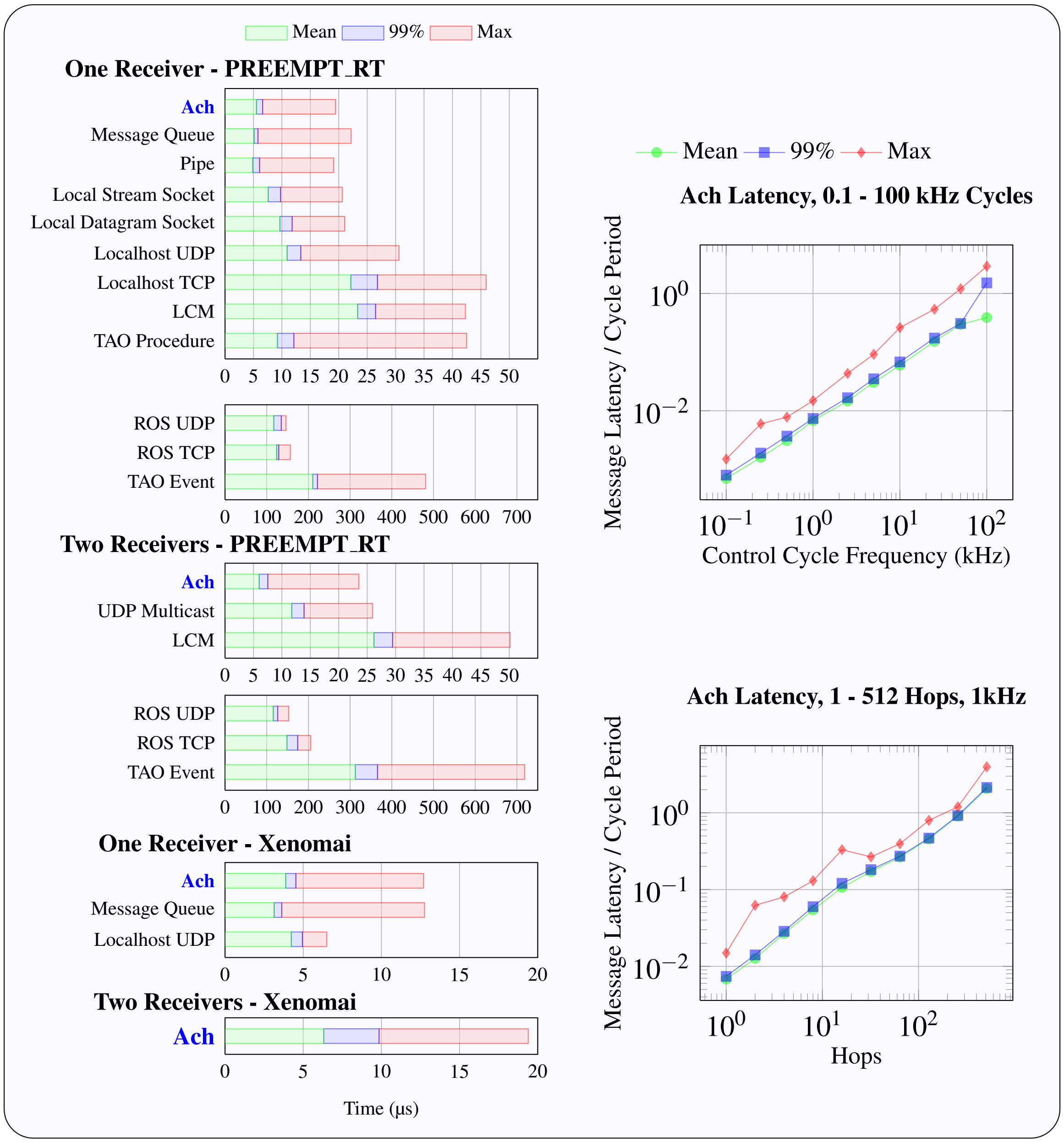


Figure 95: Message Latency for Ach, POSIX IPC, and common middleware. "Mean" is the average over all messages, "99%" is the latency that 99% of messages beat, and "Max" is the maximum recorded latency. Right-side plots show the limits of Ach performance on Linux PREEMPT_RT, with a $10^0 = 1$ latency ratio indicating latency of an entire cycle. The upper plot shows the latency ratio for various control cycle frequencies. The discontinuity above 50 kHz occurs due to transmission time exceeding the cycle period and consequent missed messages. The lower plot shows the latency ratio resulting from passing the message through multiple intermediate processes.

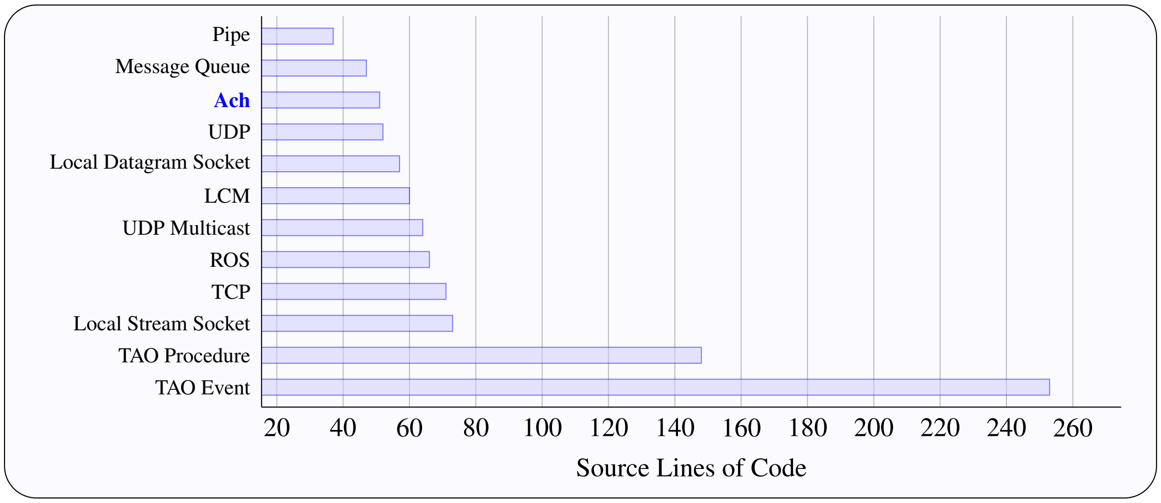


Figure 96: Source Lines of Code for each Benchmarked Method

Xenomai 2.6.2.1/RTnet 0.9.13/Linux 3.2.21[7], Ach 1.2.0, LCM 1.0.0, ROSCPP 1.9.50, and TAO 2.2.1 with ACE 6.2.1. We benchmarked one and two receivers, corresponding to the communication cases in section 6.3. Each test lasted for 600 s, giving approximately 6×10^5 data points per receiver. These results show that for the use cases in section 6.3, where communication is between a small number of processes, Ach offers a good balance of performance in addition to its unique latest-message-favored semantics.

As an approximate measure of programmer effort required for each of these methods, Fig. 96 summarizes the Source Lines of Code[8] for the method-specific code in the benchmark program. Counts include message and interface declarations and exclude generated code. To give a more fair comparison, we attempted to consistently check errors across all methods. Most methods have similar line counts, with sockets usually requiring a small amount of extra code to set up the connection. The pipe code is especially short because the file descriptors are passed through `fork`; this would not work for unrelated processes. The networked methods in the test do not consider security, which would necessarily increase

[7]While we were able to test RTnet's loopback performance, the RTnet driver for our Ethernet card caused a kernel panic. Similar stability issues with Xenomai were noted in [21]

[8]Measured using `http://www.dwheeler.com/sloccount/`

complexity of networked real-world applications, while Ach, Message Queues, and Local Domain Sockets implicitly control local data access based on user IDs. TAO CORBA stands out with several times more code than the other methods. It is also notable that the higher-level frameworks in this test did not result in significantly shorter communication code than direct use of kernel IPC.

6.4.3 Discussion

The performance limits illustrated in Fig. 95 indicate the potential applicability of Ach. The latency ratio compared to hop count is particularly important because it bounds the minimum granularity at which the control system can be divided between processes. On our test platform, significant overhead, i.e., exceeding 25% of the 1 kHz control cycle, is incurred when information must flow serially through approximately 32 processes. This cost is import to consider when dividing computation among different processes. For higher control rates, our test platform reaches 25% messaging overhead at approximately 10 kHz—. For the robots in section 6.3, the embedded components, particularly the CAN buses, effectively limit control rates to 1 kHz or lower; Ach is not the bottleneck for these systems. However, implementing systems that do require 10 kHz or greater control rates would be difficult with Ach on Linux PREEMPT_RT. These performance considerations show the range of systems for which this software design approach is suitable.

In addition to performance considerations, it is also critical to note the semantic differences between communication methods. The primary unique feature of Ach is that newer messages always supersede older messages. The other message-passing methods give priority to older data, and will block or drop newer messages when buffers are full. CORBA also differs from the other methods by exposing a remote procedure call rather than a message-passing interface, though the CORBA Event Service layers message passing on top of remote procedure call. Selecting appropriate communication semantics for an application simplifies implementation.

Some of the benchmarked methods also operate transparently across networks. This can simplify distributing an application across multiple machines, though this process is not seamless due to differences between local and network communication [154]. Processes on a single host can access a unified physical memory which provides high bandwidth and assumed perfect reliability; still, care must be taken to ensure memory consistency between asynchronously executing processes. In contrast, real-time communication across a network need not worry about memory consistency, but must address issues such as limited bandwidth, packet loss, collisions, clock skew, and security. With Ach, we have focused on efficient, latest-message-favored communication between a few processes on a single host. We intend the Ach double-circular-buffer implementation to be complementary to, and its message-passing interface compatible with, networked communication. This meets the communication requirements for systems such as those in section 6.3.

An important consideration in the design of Ach is the idea of *Mechanism, not Policy* [162]. Ach provides a mechanism to move bytes between processes and to notify callers of errors. It does not specify a policy for serializing arbitrary data structures or handling all types of errors. Such policies are application dependent and even within our own research groups have changed across different applications and over time. This separation of *policy* from *mechanism* is important for flexibility.

This flexibility is helpful when integrating with other communication methods or frameworks. To integrate with ROS on Hubo (see subsection 6.3.3), we created a separate process to translate between real-time Ach messages and non-real-time ROS messages. This approach is straightforward since both Ach and ROS expose a publish/subscribe message passing interface. On the other hand, NAOqi exposes a callback interface. Still, we can integrate with this (see subsection 6.3.2) by relaying Ach messages within the NAOqi callback. In general, integrating Ach with other frameworks requires serializing framework data structures to send over an Ach channel. However, since Ach works with raw byte arrays, it is possible directly use existing serialization methods such as XDR, Boost.Serialization, ROS Genmsg, Google Protocol Buffers, or contiguous C structures.

Achieving real-time bounds on general-purpose computing systems presents an overall challenge. The Linux PREEMPT_RT patch seamlessly runs Linux applications with significantly reduced latency compared to vanilla Linux, and work is ongoing to integrate it into the mainline kernel. However, it is far from providing formally guaranteed bounds on latency. Xenomai typically offers better latency than PREEMPT_RT [21] but is less polished and its dual kernel approach complicates development. There any many other operating systems with dedicated focus on real-time, e.g., VxWorks, QNX, TRON. In addition to operating system selection, the underlying hardware can present challenges. CPU frequency scaling, which reduces power usage, can significantly increase latency. On x86/AMD64 processors, System Management Interrupts[9] preempt all software, including the operating system, potentially leading to latencies of hundreds of microseconds. A fundamental challenge is that general purpose computation considers time not in terms of correctness but only as a quality metric – faster is better – whereas real-time computation depends on timing for correctness [115]. These issues are important in overall real-time system design.

[9]`http://www.intel.com/design/processor/manuals/253669.pdf`

```
Procedure achget
    Input: $c$ : ach channel ;                         // shared memory file
    Input: $b$ : byte array ;                          // storage for message
    Input: $n$ : integer ;                             // size of b
    Input: $s$ : integer ;                             // last seq. num. seen
    Input: $i$ : integer ;                             // next index to read
    Input: $o_w$ : boolean ;                           // wait for new message?
    Input: $o_l$ : boolean ;                           // get newest msg.?
    Output: integer $\times$ integer ;                 // size, status
    Output: $s$ : integer ;                            // new last seq. num.
    Output: $i$ : integer ;                            // new next index
 1  LOCK($c$); // take the mutex
 2  if $c.seq\_num = s \wedge o_w$ then
 3      WAIT($c$); // condition variable wait
 4  if $c.last\_seq = s \vee 0 = c.last\_seq$ then
 5      UNLOCK(c);
 6      return $(0 \times STALE)$; // no entries
    /* Find index array offset, j */
 7  if $o_l$ then
        /* newest index */
 8      $j \leftarrow (c.index\_head + c.index\_cnt - 1)\ \%\ c.index\_cnt$;
 9  else if $\neg o_l \wedge c.index\_array[i].seq\_num = s + 1$ then
10      $j \leftarrow i$; // next index
11  else
        /* oldest index */
12      $j \leftarrow (c.index\_head + c.index\_free)\ \%\ c.index\_cnt$;
    /* Now read frame from data array */
13  $x = c.index\_array[j]$;
14  if $x.size > n$ then
15      UNLOCK(c);
16      return $(x.size \times OVERFLOW)$;
17  if $x.offset + x.size < c.data\_size$ then
18      MEMCPY($b$, $c.data\_array + x.offset$, $x.size$);
19  else
20      $e = c.data\_size - x.offset$;
21      MEMCPY($b$, $c.data\_array + x.offset$, $e$);
22      MEMCPY($b + e$, $c.data\_array$, $x.size - e$);
23  $s' \leftarrow s$;
24  $s \leftarrow x.seq\_num$;
25  UNLOCK(c);
26  $i \leftarrow (i + 1)\ \%\ c.index\_cnt$;
27  if $x.seq\_num > s' + 1$ then
28      return $(x.size \times MISSED)$;
29  else
30      return $(x.size \times OK)$;
```

CHAPTER VII

EXECUTING LANGUAGE MODELS

In this section, we consider the issues that arise in the execution of the linguistic models we have developed. We describe the properties and constraints in online parsing for control as opposed to the more typical parsing domain of program translation. Then, we present a variation on the LL(1) parsing algorithm that enables generation of real-time, bounded memory parsers.

7.1 Online Parsing

We describe the linguistic properties of the Motion Grammar that arise from the online parsing of the system language. While a translating parser such as a compiler is typically given its input as a file, a Motion Parser must act token-by-token continually driving the system. This temporal constraint restricts the ability of the Motion Parser to *lookahead* and *backtrack*. Thus, we cannot apply an arbitrary Syntax-Directed Definition to an online system but are instead restricted on the type of parser we may use and the allowable ordering of attribute semantics. We now consider the issues of discrete vs. continuous time, selection of productions during parsing, and computation of attributes.

7.1.1 Discrete vs. Continuous Time

The continuous dynamics of a system may be modeled and controlled in either continuous or discrete time. For the purpose of modeling, these representations are functionally equivalent. Discrete time models can approximate continuous time by using a sufficiently short timestep, and continuous time models can represent discrete time using timeout events. For implementation on a microprocesser, we must ultimately adopt a discrete time representation; however, this can be obtained by simply discretizing the continuous-time model. The

$$\begin{array}{lcl} \langle \mathrm{A} \rangle & \rightarrow & [\mathrm{a}]\,\{u = 1\}\,\langle \mathrm{B} \rangle \\ & \mid & [\mathrm{a}]\,\{u = 1\}\,\langle \mathrm{C} \rangle \end{array}$$

(a) Semantically LL(1)

$$\begin{array}{lcl} \langle \mathrm{A} \rangle & \rightarrow & [\mathrm{a}]\,\{u = 1\}\,\langle \mathrm{B} \rangle \\ & \mid & [\mathrm{a}]\,\{u = 2\}\,\langle \mathrm{C} \rangle \end{array}$$

(b) Not Semantically LL(1)

Figure 97: Examples grammar fragments that are and are not Semantically LL(1)

Syntax-Directed Definition of the Motion Grammar can thus be written in either continuous or discrete time as is convenient.

7.1.2 Selecting Productions and Semantic Rules

We next compare the Motion Grammar to the LL(1) class of grammars. LL(1) grammars can be parsed by recursively descending through productions, picking the next production to expand using only a single token of lookahead and without backtracking [4, p.222]. While we could satisfy the Motion Grammar's temporal constraint by restricting to an LL(1) grammar, we can relax this restriction slightly. The actual requirement is not that the Motion Parser must immediately know which production it is expanding. Instead, the parser must immediately provide some input to the robot. Thus the parser may use additional lookahead, but only if all productions it is deciding between have *identical semantic rules*. This way, the parser can immediately execute the semantic rule, and use some additional lookahead to figure which production it is really expanding. We describe this property as *Semantically LL(1)*.

Definition 24. *A Syntax-Directed Definition is Semantically LL(1) if for all strings in its language, the correct semantic rule to execute can be determined using a single token of lookahead and without backtracking.*

Claim 1. *A Motion Grammar must be Semantically LL(1).*

Proof. The Motion Parser derived from the Motion Grammar, $\mathscr{G}_M$, must be able to immediately provide the system with an input $u \in \mathscr{U}$ in response to each token, and it cannot change the value of inputs already sent. Suppose that $\mathscr{G}_M$ were not Semantically LL(1).

This would mean it could use multiple tokens of lookahead or backtrack before deciding on a semantic rule to calculate u. Since u must be known before more tokens are accepted and previous u values cannot be changed, this a contradiction. Thus $\mathscr{G}_M$ must be Semantically LL(1). $\square$

The Semantically LL(1) property is useful because it allows grammars to be parsed in real-time. Examples of grammars that do and do not satisfy this property are given in Fig. 97. In addition, Fig. 17 is an example of a grammar that is not LL(1) but is Semantically LL(1). This property also permits ambiguous grammars – where multiple parse trees may exist for a given string. This is acceptable because the output of the parser, u sent to the robot, will be the same regardless of which parse tree is selected, and thus the particular resolution of the ambiguity is irrelevant.

When designing our Motion Grammar, we must ensure LL(1) semantics. This is possible with any strictly LL(1) grammar. Non-LL(1) grammars will contain conflicts where two alternative productions may begin with the same token [4, p.222]. If for any conflict, all productions contain the same semantic rules, then the grammar is Semantically LL(1). Generation of efficient parsers for LL(k) and LL(*) grammars is discussed in [142]. If the intended Motion Grammar is not Semantically LL(1), we must either rework the grammar or instruct the parser as to the appropriate precedence levels so that it can resolve any conflicting productions.

7.1.3 Attribute Inheritance and Synthesis

Now we consider the structure of the attribute semantics in the Motion Grammar. Attributes are the additional values attached to tokens and nonterminals in an SDD. For the Motion Grammar, these represent the continuous domain values x, z, and u. In our SDD, the attributes of some given nonterminal are calculated from the attributes of other tokens and nonterminals; this introduces a dependency graph into the syntax tree. We must ensure that the dependency graph has no cycles or we will not be able to evaluate the SDD [4, p.310].

The temporal nature of the Motion Grammar constrains the attribute dependencies even further; during parsing, we only have access to information from the past because the future has not happened yet. Attributes can be described as either *synthesized* or *inherited* based on their dependencies. Synthesized attributes depend on the children of the nonterminal while inherited attributes depend on the nonterminal's parent, siblings, and other attributes of the nonterminal itself. The temporal constraint of the Motion Grammar corresponds to a particular class of SDDs called *L-attributed definitions* for the left-to-right dependency chain. A nonterminal X in an L-attributed definition may only have attributes that are synthesized or are inherited with dependencies on inherited attributes of X's parent, attributes of X's siblings that precede it in the production, or on X itself in ways that do not result in a cycle [4, p.313].

Claim 2. *A Motion Grammar must have L-attributed semantics.*

Proof. We must determine the attributes in a single pass because parsing is online, so the past cannot be changed, and the future is unknown. Let the inherited attributes of nonterminal V be $V.h$, and let its synthesized attributes be $V.s$. For all productions $p = A \rightarrow X_1 X_2 \ldots X_n$, consider the attributes of X_i. While expanding X_i, $A.h$ are known. All X_j, $j < i$ in this production have already been expanded because they represent past action, so $X_j.h$ and $X_j.s$ are also known. However, X_k, $k > i$ represent future actions, so $X_k.h$ and $X_k.s$ are unknown. This also means that $A.s$ is unknown because its value may depend on $X_k.h$ and $X_k.s$. Consequently, $X_i.h$ may only depend on $A.h$, $X_j.h$, and $X_j.s$. $X_i.s$ may depend on attributes from its children because they will be known after X_i has been expanded. These constraints on attributes synthesis and inheritance correspond to L-attributed definitions. $\square$

7.2 Real-Time LL(1) Parser Generation

There is a rich literature on parser-generation to draw upon [4]; however, online parsing for real-time control presents a few challenges compared to traditional applications such as compilers.

- Compilers can look forward and backward in the input file, while a Motion Parser must provide immediate input to the system without seeing the future.
- Parse trees that represent the structure of program source code have limited depth based on size of the source code, while parse trees for a Motion Grammar may be arbitrarily large since the system may run arbitrarily long.

To handle these constraints, we place some restrictions on the grammar and perform some optimizations when generating the parser. We can conservatively satisfy these two parsing requirements with the LL(1) class of grammars [4, p.222]. LL(1) grammars require only one symbol of lookahead, need no backtracking, and operate on a single left-to-right scan of the input string. They can be parsed with constant O(1) time at each step. LL(1) grammars are rich enough for most programming language constructs [4, p.223] and for a broad class of robotic systems.

7.2.1 Bounding Memory Use

The deeply recursive nature of grammatical productions is another issue in online parsing. In these grammars (Fig. 29), concatenation and looping are implemented recursively, as nonterminals at the end of some parent production. If the goal were to build an explicit parse tree for some input file, this would present no issue. However, building such a parse tree, either as an explicit data structure or implicitly through recursive function calls, during a long running robot control operation could exhaust all memory in the computer. Thus we must avoid building such a large tree or call stack in memory.

We can avoid this arbitrarily large memory use with *tail call optimization* (TCO), as used in functional programming languages like Scheme and ML.[1] A tail call is when some function returns a value returned by another function it calls; an example is Fig. 98. The

[1]Though the language standards do not mandate it, certain compilers for certain other languages – e.g.,

```
(define (some-function a b)
  (a-tail-call (another-call a) b))
```

Figure 98: A tail recursive function in the Scheme programming language. Function `some-function` immediately returns the value returned by `a-tail-call`.

optimization is for the tail-called function to reuse the stack frame of its parent function, avoiding additional memory usage for a fresh stack frame by converting the `call` machine instruction into a `jump`. We perform a similar optimization in our parser generator. Whenever some parent production has a nonterminal in the final position of its body, there is no need to continue in the parent after expanding the child. Thus, we jump – `goto` in C – to the code for that nonterminal rather than recursively expanding it. TCO limits memory usage for the deeply recursive Motion Parser.

Claim 3. *When tail recursion is optimized to a jump, call stack use by tail-recursive calls is constant.*

Proof. A jump instruction does not create a new stack frame, and thus cannot grow the call stack. □

A formal semantics for and proof of 3 is given by [132].

Claim 4. *Consider grammar productions of the form $p = A \to x_1 \ldots x_n B$, where each x_i is a terminal and B is a nonterminal. An LL(1) parser with TCO requires constant space to expand each production of the form p.*

Proof. Expanding each x_i requires constant space as we must only check that x_i matches the next symbol in the input string. Expanding B requires constant space according on 3

GCC for C and SBCL for Common Lisp – will perform TCO when given certain optimization options. However, this optimization is not always performed, so it cannot be depended upon for correctness. For parser generators emitting code in a language that guarantees TCO, e.g., Scheme, it is unnecessary to explicitly optimize tail calls within the parser generator itself. However, languages such as Scheme are garbage-collected, which presents additional issues for real-time control.

because B is expanded with a tail call. □

Claim 5. *Consider grammar productions of the form $p = A \rightarrow x_1 \dots x_i B x_{i+1} \dots x_n C$, where each x_i is a terminal and B and C are nonterminals. If every production for B is of the form $B \rightarrow y_1 \dots y_n$ or $B \rightarrow y_1 \dots y_n D$, where $y_1 \dots y_n$ are terminals and D is a nonterminal, then p can be expanded with constant space.*

Proof. Every terminal $x_1 \dots x_n$ can be expanded in constant space. From 4, every production for B can be expanded in constant space. From 3, the C in the tail position of p requires no additional stack space. Therefore, p can be expanded in constant space. □

Claim 6. *$\hat{G}$ can be parsed in constant space.*

Proof. Every production of $\hat{G}$ is of the form given in 3 or 5, which can be parsed in constant space. Therefore, $\hat{G}$ can be parsed in constant space. □

7.2.2 Parser Implementation

We now implement LL(1) parser generation to construct a Motion Parser. To generate standard C, we need to optimize tail calls to `goto`. Since C `goto` can only target a label in the same function, we must implement our parser as a single function. As a design choice, we use the C call stack for the context-free parsing stack, which is simpler to implement than maintaining an explicit stack data structure. Consequently, the parsing function is self-recursive. The nonterminals and productions in the parsing function are arranged in a jump table, represented as a C `switch-case` with one `case` for each nonterminal and each production. The block for each nonterminal first identifies which production for that nonterminal to expand, based on the set of initial terminals possible for that nonterminal. We then expand each symbol in that production. For nonterminals not in the tail position, we recursively call the parsing function and switch to the appropriate case in the jump table for that nonterminal. For tail nonterminals, we directly jump to the case for that nonterminal. With this design, we can parse arbitrarily long strings for tail-recursive LL(1) grammars while using a bounded amount of memory.

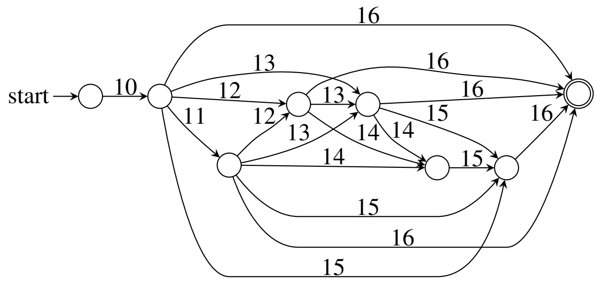
```
 1 int super_mgparse
 2 ( mg_context_t* context,
 3   mg_supervisor_table_t *table, int i )
 4 { //...
 5 case 424:
 6 nonterm__lp_0_sp__dot__sp_g1_rp_:
 7   // (STEP TIME-ZERO STEP-1)
 8   if ( ((mg_supervisor_allow(table, 5)) &&
 9          (0 == (time_zero(context)))) )
10   {
11     (table->state) =
12       (mg_supervisor_next_state(table, 5));
13   case 425:
14   prod__lp__lp_0_sp__dot__sp\
15   _g1_rp__sp_time_zero_rp_:
16     goto nonterm__lp_1_sp__dot__sp_g1_rp_;
17   }
18   return -1;
19   //...
20 }
```

Figure 99: Example of parsing code

Fig. 99 shows a fragment of the generated parser, corresponding to the first production of Fig. 29. This parser first calls `time_zero`, line 10. Notice the `goto` in line 16, implementing the tail recursive expansion. The full generated code for the 504 production grammar amounts to 4,606 lines of lines of C code. When compiled with gcc 4.7.2 `-O2`, this produces 12,013 lines of AMD64 assembly.

7.2.3 Online Supervision

We implement online supervisory control with a minor extension to our LL(1) parser generator. Effectively, we execute the LL(1) parser for the initial grammar $\hat{G}$ in parallel with the supervisory Finite Automaton S_c, transitioning only when both allow it [86, p.135]. Before the parser checks any terminal symbol or executes any semantic rule, it first ensures that the action is allowed from the current state of supervisor S_c. After reading the terminal or executing the semantic rule, the parser updates the state of S_c for the transition on that symbol. The result is implements the supervised system, $G' = \hat{G} \cap S$.


(a) Ascending

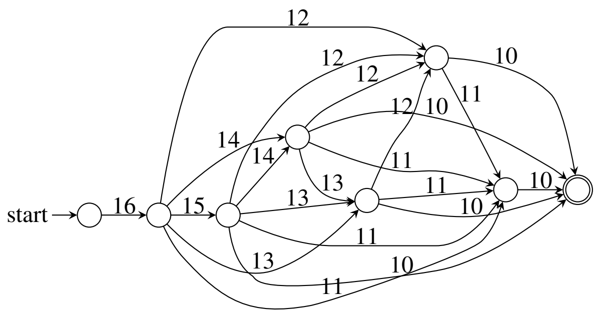

(b) Descending

Figure 100: Monotonically ascending and descending Speed Finite Automata, A_m

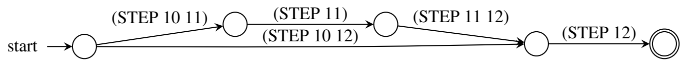


Figure 101: Ascending FA with transition steps, A_t

We apply this approach to perform speed controlled walking on the Nao. We algorithmically derive the specification for a supervisor to take the NAO between any two speeds via an optimal sequence of steps. Then, we provide this supervisor to our parser. By following the generated supervisor, the parser performs speed control.

To generate the supervisor, we first start with the speed FA G_s as in Fig. 26 and transform it to an FA with monotonically ascending or descending speeds, Fig. 100. This ensures that the robot will continually increase or decrease in speed. We produce these monotonic FA by repeatedly intersecting the speed FA G_s with a languages S_m to enforce ordering for each terminal symbol. For terminal y, this ordering language S_m is given by the regular expression,

$$S_m = \{x : x < y\}^* y \{x : x > y\}^* \tag{163}$$

This specification ensures that all symbols before y in the string are less than y and all symbols after y in the string are greater than y, enforcing an ascending constraint. The reverse enforces a descending constraint. By applying an S_m for each speed, we produce the monotonic FA A_m in Fig. 100.

Next, we insert the transition steps into the monotonic FA A_m. The result A_t for the

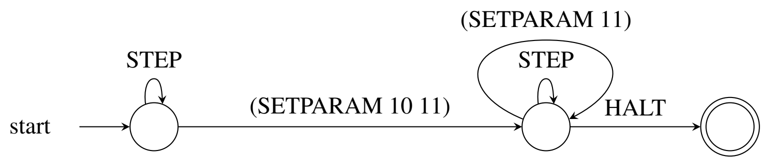


Figure 102: Supervisor for transitioning from 10 to 11 cm/s. Here, STEP corresponds to the union of all terminals in Fig. 29

ascending case is shown in Fig. 101.

From A_t, we find an optimal sequence of steps σ to reach a desired speed. Each transition [step i] and [step i j] in A_t (Fig. 101) is assigned a cost based on its stability margin. Then, we apply Djikstra's Algorithm [36, p.595] to identify the minimum cost path to the target speed. For Fig. 101, this gives the following two steps

$$\sigma = \text{[step 10 17]}\ \text{[step 17]} \tag{164}$$

From the string σ in (164), we generate a regular expression S_c for the supervisor. Initially, let S_c be $\langle\text{step}\rangle^*$, which here denotes the union of all terminal symbols in Fig. 29. For each [step a b] in σ, we concatenate to S_c the expression [setparam a b] $\langle\text{step}\rangle^*$. For the last symbol in σ, indicating the target speed, concatenate ([setparam a] $\langle\text{step}\rangle$)* [HALT]. The result for (164) is the following regular expression, show as an FA in Fig. 102.

$$\begin{aligned} S = \langle\text{step}\rangle^*\,\text{[setparam 10 17]}\,\langle\text{step}\rangle^*\,\text{[setparam 17]} \\ (\text{[setparam 17]}\,\langle\text{step}\rangle^*)^*\,\text{[HALT]} \end{aligned} \tag{165}$$

Theorem 8. *Every σ in G' is stable. G' contains no unstable paths.*

Proof. From 5, every $\sigma \in \hat{G}$ is stable. Since $G' \subseteq \hat{G}$, every σ in G' is also in $\hat{G}$. Since, $\sigma \in \hat{G}$ implies σ is stable, every $\sigma \in G'$ must stable. □

CHAPTER VIII

CONCLUSION AND FUTURE WORK

This thesis has developed a pipeline for robot policy specification, analysis, and execution through the use of language and grammars. Using formal language as the intermediate representation for robot policies, we implement various previously disparate techniques – including logical planning, learning from demonstration, and semantic mapping for mobile manipulation – as instances of this approach. These techniques provide alternative ways to automatically specify robot behavior. This formal language framework connects these specification approaches with the large body of work on discrete event and hybrid systems, providing tools for policy verification. In our primary application domain of robot manipulation, we develop techniques for workspace interpolation and online camera registration. To communicate with embedded hardware and tolerate software errors outside the formally-modeled system, we develop a new real-time IPC system. Finally, we consider the execution properties of hierarchical policies represented as context-free grammars and introduce a variation on the LL(1) parsing algorithm.

8.1 Contributions

The contributions of this thesis are as follows.

8.1.1 Integrate specification, analysis, and execution

This thesis develops a formal language framework to combine system specification, analysis, and execution. The key insight is to use grammars and automata as an intermediate representation. We translate various approaches for policy specification, e.g. logical planning domains and human demonstrations, into a linguistic representation, apply analysis techniques such as model checking and supervisory control, and synthesize control software for the robot.

8.1.2 Data-Driven Specification

This thesis demonstrates data driven approaches for specification. We apply human-inspired control, which generates control modes for bipedal walking from human data, and from this generated formal model, synthesize software to run the robot. We also frame learning from demonstration for an assembly task as a grammatical inference problem, and use visual analysis of assembly demonstrations to generate automata for the task.

8.1.3 Logical Domain Policies

This thesis gives a method to produce minimum finite-state policies for logical planning domains. We modify Hopcroft's finite automata minimization algorithm to directly convert the logical domain to the minimized automaton, without require an intermediate automaton.

8.1.4 Hierarchical Policy Compaction

This thesis further reduces the memory usage of logical policies by inferring hierarchies. From the finite automataon, we find repeated submachines and combine these machines together. This process can even operate recursively to find multiple levels of hierarchy.

8.1.5 Generating Real-Time Software

This thesis presents a method to generate software directly from the previously developed formal models. We modify the classic LL(1) parser generation algorithm – which traditionally focused on program translation – to operate in real-time and with bounded memory use.

8.1.6 Real-Time Communication

This thesis presents a high-performance, real-time, communication library, Ach, based on a double circular-buffer data structure. The Ach library provides the unique feature of latest-message-favored semantics with multiple senders and receivers. Our benchmark results show that Ach offers significantly lower latency than popular robotics middleware such as TAO-CORBA and ROS, and notably that it has lower latency than even direct use of Linux sockets.

8.1.7 Direct, Nonstop Workspace Interpolation

This thesis presents a new method for multi-point workspace interpolation in the manipulation domain. Classic approaches take either indirect paths or are point to point. Our method blends subsequent linear spherical interpolation phases to transition through a sequence of waypoints with continuous, nonstop motion.

8.2 *Future Work*

There are many ways to build on this work. Modeling with probabilities can help performance in the absence of complete information, and stochastic grammars have been successful for activity recognition. Coupling this approach with grammars for online control could accommodate imperfect perception over whole tasks. This would be a special class of POMDPs, but grammars help represent task structure and the efficient parsing algorithms may help make the problem tractable. In section 3.4, we applied a simple form a grammatical inference to transfer human demonstrations to robot policies. Extending the approach using more advanced stochastic or querying inference algorithms would enable learning policies for more complex tasks. Recursive process algebras, which extend grammars with operators for concurrency, may be useful for reasoning about multi-robot systems but would require addressing challenges such as limited communication and distributed computation. General software verification approaches may be useful for hybrid dynamical systems. Type-checking in particular is appealing given its efficiency and may be useful for representing conditions like reachability or reducing the state space by encoding some information as types. A type system could also be developed to encode time constraints. Time is generally considered distinctly from program correctness, yet it is vital for correct real-time systems. Testing software for cyber-physical systems is challenging because it is often critical to avoid any failures on the actual system. Testing in simulation is an alternative, but producing sufficiently accurate simulations is a challenge itself. When linguistic policies are difficult to construct, searching through simulation-preserving languages is an option; our rewrite rules in [44] are one basis for this.

Developing applications for real-world environments also means addressing software systems challenges where end-to-end guarantees may be impractical. While our Ach IPC library has lower latency than Linux sockets, there are still challenges with many processes and connections. Better synchronization using futexes, a lock-free approach, or transactional memory could improve concurrency, and an in-kernel implementation could support I/O multiplexing and improve robustness. Dynamic memory allocation is a challenge for real-time control since typical allocators introduce unpredictable pauses, and allocation errors can be catastrophic. Completely avoiding dynamic allocation, though, can excessively restrict software development. Existing real-time allocators, such as TLSF, have room for improvement in efficiency and error handling.

APPENDIX A

FORMAL LANGUAGE

Grammars define *languages*. For instance, C and LISP are computer programming languages, and English is a human language for communication. A formal grammar defines a formal language, a set of strings or sequences of discrete *tokens*.

Definition 25 (Context-Free Grammar, CFG)**.** $G = (Z,V,P,S)$ *where* Z *is a finite alphabet of symbols called* tokens*,* V *is a finite set of symbols called* nonterminals*,* P *is a finite set of mappings* $V \mapsto (Z \cup V)^*$ *called* productions*, and* $S \in V$ *is the* start symbol*.*

The productions of a CFG are conventionally written in Backus-Naur form. This follows the form $A \rightarrow X_1X_2\dots X_n$, where A is some nonterminal and $X_1 \dots X_n$ is a sequence of tokens and nonterminals. This indicates that A may *expand* to all strings represented by the right-hand side of the productions. The symbol ε is used to denote an empty string. For additional clarity, nonterminals may be represented between angle brackets $\langle\rangle$ and tokens between square brackets [].

Grammars have equivalent representations as *automata* which *recognize* the language of the grammar. In the case of a Regular Grammar – where all productions are of the form $\langle\text{A}\rangle \rightarrow [\text{a}]\,\langle\text{B}\rangle$, $\langle\text{A}\rangle \rightarrow [\text{a}]$, or $\langle\text{A}\rangle \rightarrow \varepsilon$ – the equivalent automaton is a Finite Automaton (FA), similar to a Transition System with finite state. A CFG is equivalent to a Pushdown Automaton, which is an FA augmented with a stack; the addition of a stack provides the automaton with memory and can be intuitively understood as allowing it to count.

Definition 26 (Finite Automata, FA)**.** $M = (Q,Z,\delta,q_0,F)$*, where* Q *is a finite set of* states*,* Z *is a finite alphabet of* tokens*,* $\delta : Q \times Z \mapsto Q$ *is the* transition function*,* $q_0 \in Q$ *is the* start state*,* $F \in Q$ *is the set of* accept states*.*

Definition 27 (Acceptance and Recognition)**.** *An automaton M* accepts *some string σ if M is in an accept state after reading the final element of σ. The set of all strings that M accepts is the* language *of M, $\mathfrak{L}(M)$, and M is said to* recognize $\mathfrak{L}(M)$.

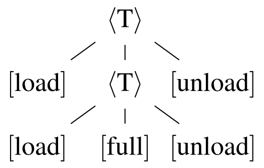
Regular Expressions [86] and Linear Temporal Logic (LTL) [13] are two alternative notations for finite state languages. The basic Regular Expression operators are concatenation $\alpha\beta$, union $\alpha|\beta$, and Kleene-closure α^*. Some additional common Regular Expression notation includes $\overline{\alpha}$ which is the complement of α, the dot (.) which matches any token, and α? which is equivalent to $\alpha|\varepsilon$. Regular Expressions are equivalent to Finite Automata and Regular Grammars. LTL extends propositional logic with the binary operator *until* $\cup$ and unary prefix operators *eventually* $\diamond$ and *always* $\square$. LTL formula are equivalent to Büchi automata, which represent *infinite* length strings, termed ω-Regular languages. We can also write ω-Regular Expressions by extending classical Regular expressions with infinite repetition for some α given as α^ω. These additional notations are convenient representations for finite state languages.

Any string in a formal language can be represented as a *parse tree*. The root of the tree is the start symbol of the grammar. As the start symbol is recursively broken down into tokens and nonterminals according to the grammar syntax, the tree is built up according to the productions that are expanded. The production $A \to X_1 \dots X_n$ will produce a piece of the parse tree with parent A and children $X_1 \dots X_n$. The children of each node in the parse tree indicate which nonterminals or tokens that node *expands* to in a given string. Internal tree nodes are nonterminals, and tree leaves are tokens. The parse tree conveys the full syntactic structure of the string.

An example CFG and parse tree are given in Fig. 103 for a loading and unloading task. In production (166), the system will repeatedly perform [load] operations until receiving a [full] token from production (167). Then the system will perform [unload] operations of the same number as the prior [load] operations. This simple use of *memory* is possible with Context-Free systems. Regular systems are not powerful enough.

$$
\begin{array}{rcll}
\langle \mathrm{T} \rangle & \rightarrow & [\mathrm{load}]\,\langle \mathrm{T} \rangle\,[\mathrm{unload}] & (166) \\
 & | & [\mathrm{full}] & (167)
\end{array}
$$

(a) Grammar



(b) Parse Tree

Figure 103: Example Context-Free Grammar for a load/unload task and parse tree for string "[load] [load] [full] [unload] [unload]"

While grammars and automata describe the structure or *syntax* of strings in the language, something more is needed to describe the meaning or *semantics* of those strings. One approach for defining semantics is to extend a CFG with additional *semantic rules* that describe operations or actions to take at certain points within each production. Additional values computed by a semantic rule may be stored as *attributes*, which are parameters associated with each nonterminal or token, and then reused in other semantic rules. The resulting combination of a CFG with additional semantic rules is called a *Syntax-Directed Definition* (SDD) [4, p.52].

APPENDIX B

DUAL QUATERNIONS

Quaternions are a convenient representation for spatial motion that provides some computational advantages over other methods.

The straightforward definitions of many quaternion quantities, particularly exponentials, logarithms, and derivatives, contain singularities where a denominator goes to zero. We can avoid computational problems at these points by computing key factors near the singularity using a Taylor series, though this may require some careful rearrangement of terms to identify suitable factors and series.

A Taylor series evaluated near point a is:

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \\ &= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \\ &\qquad \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots \end{aligned} \tag{168}$$

To evaluate the infinite series to machine precision, we only need to compute up the term below floating point round-off.

The resulting approximation is a polynomial which can be efficiently evaluated using Horner's Rule, Algorithm 13. The coefficients are the terms $\frac{f(n)(a)}{n!}$ and the indeterminate variable is $x-a$. Note than many Taylor series have zero coefficents for the odd or even terms. We can produce a more compact Horner polynomial by omitting the zero coefficients, using $(x-a)^2$ as the indeterminate variable, and perhaps multiplying the whole result by $(x-a)$.

We adopt the following abbreviations to condense notation:

Algorithm 13: Horner's Rule

```
Input: $b_0, b_1, \dots b_n$ : Coefficients
Input: $z$ : Indeterminate Variable
Output: $y$ : Result
1 $y \leftarrow b_n$
2 $y \leftarrow b_{n-1} + zy$
3 $y \leftarrow b_{n-2} + zy$
4 $\dots$
5 $y \leftarrow b_0 + zy$
```

- Quaternions are typeset as q.
- Dual Quaternions are typeset as $\mathcal{S}$.
- Vectors are typeset as $\vec{x}$.
- Matrices are typeset as $\mathbf{A}$.
- Time derivatives of variable x are given as $\dot{x}$.
- Sines and cosines are abbreviated as s and c.

B.1 Quaternions

Quaternions are an extension of the complex numbers, using basis elements i, j, and k defined as:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{169}$$

From (169), it follows:

$$jk = -kj = i \tag{170}$$

$$ki = -ik = j \tag{171}$$

$$ij = -ji = k \tag{172}$$

A quaternion, then, is:

$$q = w + xi + yj + zk \tag{173}$$

Table 12: Algebraic Quaternion Properties

Associative	$p \otimes (q \otimes r) = (p \otimes q) \otimes r$
Distributive	$p \otimes (q + r) = p \otimes q + p \otimes r$
NOT Commutative	$p \otimes q \neq q \otimes p$
Conjugate Mul.	$(p \otimes q)^* = q^* \otimes p^*$
Conjugate Add.	$(p + q)^* = q^* + p^*$

B.1.1 Representation

We represent a quaternion as a 4-tuple of real numbers:

$$\begin{aligned} q &= w + xi + yj + zk \\ &= (x\,y\,z\,w) \\ &= (q_v,\ w) \end{aligned} \tag{174}$$

Historically, q_v is called the vector part of the quaternion and q_w the scalar part.

It is convenient to define quaternion operations in terms of vector and matrix operations, so we also the whole quaternion as a column vector. This also provides an in-memory storage representation.

$$\vec{q} = [x\,y\,z\,w]^T \tag{175}$$

$$\vec{q}_v = [x\,y\,z]^T \tag{176}$$

A alternate convention stores terms in *wxyz* order, so when using different software packages, it is sometimes necessary to convert between orderings.

B.1.2 Multiplication

From the definition of the basis elements (169), we obtain a formula for quaternion multiplication.

B.1.2.1 Cross and dot product definition

We define quaternion multiplication in terms of cross products and dot products of its elements:

$$q \otimes p = \begin{pmatrix} \vec{q}_v \times \vec{p}_v + q_w \vec{p}_v + p_w \vec{q}_v \\ q_w p_w - \vec{q}_v \cdot \vec{p}_v \end{pmatrix} \tag{177}$$

B.1.2.2 Matrix definition

Expanding the above terms, we can express quaternion multiplication as matrix multiplication:

$$\begin{aligned} q \otimes p &= \\ \mathbf{Q_L}\vec{p} &= \begin{bmatrix} q_w & -q_z & q_y & q_x \\ q_z & q_w & -q_x & q_y \\ -q_y & q_x & q_w & q_z \\ -q_x & -q_y & -q_z & q_w \end{bmatrix} \vec{p} = \\ \mathbf{P_R}\vec{q} &= \begin{bmatrix} p_w & p_z & -p_y & p_x \\ -p_z & p_w & p_x & p_y \\ p_y & -p_x & p_w & p_z \\ -p_x & -p_y & -p_z & p_w \end{bmatrix} \vec{q} = \\ &\begin{bmatrix} q_x p_w + q_y p_z + q_w p_x - q_z p_y \\ q_z p_x + q_w p_y + q_y p_w - q_x p_z \\ q_w p_z + q_z p_w + q_x p_y - q_y p_x \\ -(q_y p_y + q_x p_x + q_z p_z - q_w p_w) \end{bmatrix} \end{aligned} \tag{178}$$

This matrix form is more suitable for efficient implementation computation using SIMD instructions.

B.1.2.3 Properties

Quaternion multiplication is associative and distributive, but it is not commutative.

B.1.2.4 Pure Multiplication

When multiplying by a pure quaternion, i.e., zero scalar part, we can simplify:

$$q \otimes (v,0) = \begin{bmatrix} q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \\ -q_x & -q_y & -q_z \end{bmatrix} v =$$

$$\begin{bmatrix} q_y v_z \\ q_z v_x \\ q_x v_y \\ -q_x v_x \end{bmatrix} - \begin{bmatrix} q_z v_y \\ q_x v_z \\ q_y v_x \\ q_y v_y \end{bmatrix} + \begin{bmatrix} q_w v_x \\ q_w v_y \\ q_w v_z \\ -q_z v_z \end{bmatrix} \tag{179}$$

$$(v,0) \otimes q = \begin{bmatrix} q_w & q_z & -q_y \\ -q_z & q_w & q_x \\ q_y & -q_x & q_w \\ -q_x & -q_y & -q_z \end{bmatrix} v =$$

$$\begin{bmatrix} v_y q_z \\ v_z q_x \\ v_x q_y \\ -v_x q_x \end{bmatrix} - \begin{bmatrix} v_z q_y \\ v_x q_z \\ v_y q_x \\ v_y q_y \end{bmatrix} + \begin{bmatrix} v_x q_w \\ v_y q_w \\ v_z q_w \\ -v_z q_z \end{bmatrix} \tag{180}$$

$$(u,0) \otimes (v,0) = \begin{bmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \\ -u_x v_x - u_y v_y - u_z v_z \end{bmatrix} = \begin{bmatrix} u \times v \\ -u \cdot v \end{bmatrix} \tag{181}$$

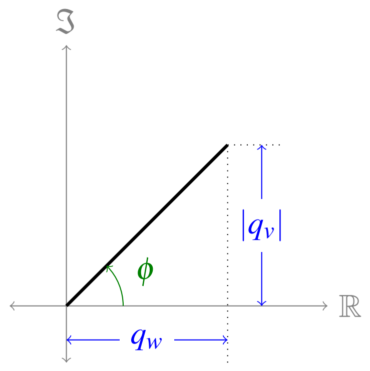


Figure 104: Imaginary Plane for Quaternions

Thus, the case of multiplying two pure quaternion simplifies to the commonly used cross ($\times$) and dot ($\cdot$) products.

B.1.3 Norm

$$|q| = \sqrt{\vec{q} \cdot \vec{q}} \tag{182}$$

A unit quaternion has norm of one.

B.1.4 Conjugate

$$q^* = (-q_v,\ q_w) \tag{183}$$

B.1.5 Inverse

$$q^{-1} = \frac{q^*}{\vec{q} \cdot \vec{q}} \tag{184}$$

Note that for unit quaternion, the inverse is equal to the conjugate.

B.1.6 Exponential

The exponential shows the relationship between quaternions and complex numbers. Recall Euler's formula for complex numbers:

$$e^{i\theta} = \cos\theta + i \sin\theta \tag{185}$$

which relates the exponential function with angles in the complex plane. Similarly for quaternions, we can consider the angle between the real and imaginary parts, Fig. 104, yielding some useful trigonometric ratios for analyzing quaternion functions:

$$\phi = \operatorname{atan2}(|q_v|, q_w) \tag{186}$$

$$\sin\phi = \frac{|q_v|}{|q|} \tag{187}$$

$$\cos\phi = \frac{q_w}{|q|} \tag{188}$$

The quaternion exponential is:

$$e^q = e^{q_w}\left(q_v \frac{\sin|q_v|}{|q_v|},\ \cos|q_v|\right) \tag{189}$$

When $|q_v|$ approaches zero, we can use the Taylor series approximation:

$$\frac{\sin\theta}{\theta} = 1 - \frac{\theta^2}{6} + \frac{\theta^4}{120} - \frac{\theta^6}{5040} + \dots \tag{190}$$

For a pure quaternion, the exponential simplifies to:

$$q_w = 0 \implies \begin{cases} e^q = \left(q_v \frac{\sin|q_v|}{|q_v|},\ \cos|q_v|\right) \\ |e^q| = 1 \end{cases} \tag{191}$$

B.1.7 Logarithm

To compute the logarithm, first consider the angle between the vector and scalar parts of the quaternion.

$$\phi = \cos^{-1}\left(\frac{q_w}{|q|}\right) = \sin^{-1}\left(\frac{|q_v|}{|q|}\right) = \operatorname{atan2}(|q_v|, q_w) \tag{192}$$

The atan2 form to compute ϕ is generally best for numerical stability.

$$\ln q = \left(\frac{\phi}{|q_v|} q_v,\ \ln(|q|)\right) \tag{193}$$

When $|q_v|$ approaches zero, we can compute $\frac{\phi}{|q_v|}$ as follows:

$$\frac{\phi}{|q_v|} = \frac{\frac{\phi}{|q|}}{\frac{|q_v|}{|q|}} = \frac{\frac{\phi}{|q|}}{\sin\phi} = \frac{\frac{\phi}{\sin\phi}}{|q|} \tag{194}$$

Then, $\frac{\phi}{\sin\phi}$ can be approximated by Taylor series:

$$\frac{\theta}{\sin\theta} = 1 + \frac{\theta^2}{6} + \frac{7\theta^4}{360} + \frac{31\theta^6}{15120} + \dots \tag{195}$$

For a unit quaternion, the logarithm simplifies to:

$$\left|q\right| = 1 \implies \ln(q) = \left(\frac{\phi}{\sin\phi} q_v,\, 0\right) \tag{196}$$

B.1.8 Power

$$q^t = e^{t \ln q} \tag{197}$$

B.1.9 Pure Exponential Derivative

The derivative of the exponential for a pure quaternion is:

$$\phi = |q_v| = \sqrt{q_v \cdot q_v} \tag{198}$$

$$\dot{\phi} = \frac{d\,|q_v|}{dt} = \frac{q_v \cdot \dot{q}_v}{\phi} \tag{199}$$

$$e^q = \left(\frac{\sin\phi}{\phi} q_v,\, \cos\phi\right) \tag{200}$$

$$\left(\frac{de^q}{dt}\right)_w = -\sin\phi\dot{\phi} = -\left(q_v \cdot \dot{q}_v\right)\frac{\sin\phi}{\phi} \tag{201}$$

$$\begin{aligned}\left(\frac{de^q}{dt}\right)_v &= \frac{s}{\phi}\dot{q}_v + \left(\frac{\dot{\phi}c}{\phi} - \frac{\dot{\phi}s}{\phi^2}\right)q_v = \\ &\frac{s}{\phi}\dot{q}_v + \left(\frac{c}{\phi^2} - \frac{s}{\phi^3}\right)\left(q_v \cdot \dot{q}_v\right)q_v = \\ &\frac{s}{\phi}\dot{q}_v + \left(\frac{c - \frac{s}{\phi}}{\phi^2}\right)\left(q_v \cdot \dot{q}_v\right)q_v\end{aligned} \tag{202}$$

Then, we handle the singularity for $\phi = 0$ using (190) and the following:

$$\frac{c}{\phi^2} - \frac{s}{\phi^3} = -\frac{1}{3} + \frac{\phi^2}{30} - \frac{\phi^4}{840} + \frac{\phi^6}{45360} + \dots \tag{203}$$

B.1.10 Unit Logarithm Derivative

The derivative of the unit quaternion logarithm is:

$$\begin{aligned} \dot{\phi} = |q_v|\,\dot{q}_w - q_w \frac{d\,|q_v|}{dt} = |q_v|\,\dot{q}_w - q_w \frac{q_v \cdot \dot{q}_v}{\sin\phi} = \\ s\dot{q}_w - \frac{cq_v \cdot \dot{q}_v}{s} \end{aligned} \tag{204}$$

$$\frac{d \ln q(t)}{dt} = \left(\frac{\phi}{s}\dot{q}_v + \left(\frac{\dot{\phi}}{s} - \frac{\phi\dot{\phi}c}{s^2} \right) q_v,\ 0 \right) \tag{205}$$

This form has a singularity at $\phi = 0$. We can handle one factor with (195) and the other as follows:

$$\begin{aligned} \frac{\dot{\phi}}{s} - \frac{\phi\dot{\phi}c}{s^2} = \dot{q}_w - \frac{\dot{q}_w\phi c}{s} - \frac{q_v \cdot \dot{q}_v c}{s^2} + \frac{q_v \cdot \dot{q}_v \phi c^2}{s^3} = \\ \dot{q}_w \left(1 - \frac{\phi}{s}c\right) + q_v \cdot \dot{q}_v \left(\frac{\phi c^2}{s^3} - \frac{c}{s^2} \right) \end{aligned} \tag{206}$$

$$\frac{\phi c^2}{s^3} - \frac{c}{s^2} = -\frac{1}{3} + \frac{\phi^2}{30} + \frac{53\phi^4}{2520} + \frac{367\phi^6}{75600} + \dots \tag{207}$$

Thus, for small ϕ, we use the Taylor series for $\frac{\sin\phi}{\phi}$ (190), $\frac{\phi}{\sin\phi}$ (195), and $\frac{\phi c^2}{s^3} - \frac{c}{s^2}$ (207). Note that since $\frac{\sin\phi}{\phi} \approx 1$ near $\phi = 0$, we can safely compute $\frac{\phi}{\sin\phi} = 1/\frac{\sin\phi}{\phi}$, which should be more efficient that a second Taylor series evaluation.

Alternatively, one could also compute the Jacobian $\frac{\partial \ln q}{\partial q}$ [56].

B.1.11 Unit Quaternion Angle

We can compute the angle between the vector forms of two unit quaternion as follows:

Table 13: Storage Requirements for Orientation Representations

Representation	**Storage**
Quaternion	4
Axis-Angle	4
Rotation Vector	3
Euler Angles	3
Rotation Matrix	9

Table 14: Computational Requirements for Orientation Representations

Representation	**Chain**	**Rotate Point**
Quaternion	16 multiply, 12 add	15 multiply, 15 add
Rotation Matrix	27 multiply, 18 add	9 multiply, 6 add

$$\angle(\vec{q}_1, \vec{q}_2) = \cos^{-1}(\vec{q}_1 \cdot \vec{q}_2) =$$
$$2\ \mathrm{atan2}\left(\left|q_1 - q_2\right|, \left|q_1 + q_2\right|\right) \tag{208}$$

The atan2 form is more accurate [96].

B.1.12 Product Rule

Because quaternion multiplication is a linear operation, the product rule applies:

$$\frac{d}{dt}\left(q_1 \otimes q_2\right) = \dot{q}_1 \otimes q_2 + q_1 \otimes \dot{q}_2 \tag{209}$$

B.2 Representing Orientation

A unit quaternion ($\left|q\right| = 1$) can represent an angular orientation.

B.2.1 Rotating a vector

We can rotate point v by unit quaternion q by computing $v' = \mathrm{rot}\left(q, v\right) = q \otimes v \otimes q^*$. Note that v is augmented with 0 in it's w position to perform the quaternion multiplication operation. Given this 0 value, the computation can be simplified to the following:

$$v' = \mathrm{rot}\left(q, v\right) =$$

$$q \otimes v \otimes q^{*} = 2\vec{q}_v \times (\vec{q}_v \times v + q_w v) + v \tag{210}$$

which we can rewrite in a more SIMD-friendly form as:

$$\begin{aligned} a &= q_v \times v + q_w v \\ b &= q_v \times a \\ v' &= b + b + v \end{aligned} \tag{211}$$

B.2.2 Chaining rotations

Rotations q_1 and q_2 are chained by multiplying the two quaternions: $q_1 \otimes q_2$.

B.2.3 Angular Derivatives

Rotational velocity ω is related to the quaternion derivative as follows:

$$\dot{q} = \frac{1}{2}\omega \otimes q \tag{212}$$

$$\omega = 2\dot{q} \otimes q^{*} \tag{213}$$

Rotational acceleration $\dot{\omega}$ is related to the quaternion derivative as follows:

$$\ddot{q} = \frac{1}{2}\left(\dot{\omega} \otimes q + \omega \otimes \dot{q}\right) \tag{214}$$

$$\dot{\omega} = 2\left(\ddot{q} \otimes q^{*} + \dot{q} \otimes \dot{q}^{*}\right) \tag{215}$$

B.2.4 Axis-Angle

The axis-angle form, $a = (\hat{u}, \theta)$ represents rotation by angle θ around unit axis $\hat{u}$. We can also normalize the representation by scaling the axis by the angle $v = \theta\hat{u}$, which is sometimes called the rotation vector form.

Rotation vectors are related to unit quaternions through the exponential and logarithm.

$$q = \left(\hat{u}\sin\frac{\theta}{2},\, \cos\frac{\theta}{2}\right) = e^{\frac{\theta}{2}\hat{u}} = \left(\frac{v}{|v|}\sin\frac{|v|}{2},\, \cos\frac{|v|}{2}\right) = e^{\frac{v}{2}} \tag{216}$$

$$\theta = 2\cos^{-1}(q_w) = 2\tan^{-1}(|q_v|, q_w) = 2|\ln q| \tag{217}$$

$$\hat{u} = \begin{cases} \theta \neq 0 & \frac{q_v}{\sin\frac{\theta}{2}} \\ \theta = 0 & 0 \end{cases} = \frac{\ln q}{|\ln q|} \tag{218}$$

$$v = 2\ln q \tag{219}$$

The rotation vector and quaternion derivatives are related as follows, substituting $y = \frac{v}{2}$, $\dot{y} = \frac{\dot{v}}{2}$, and $\phi = |y|$:

$$\dot{\phi} = \frac{y \cdot \dot{y}}{\phi} \tag{220}$$

$$\dot{q}_w = -\dot{\phi}\sin\phi = (y \cdot \dot{y})\frac{\sin\phi}{\phi} \tag{221}$$

$$\dot{q}_v = \frac{\sin\phi}{\phi}\dot{y} - \frac{\dot{\phi}\sin\phi}{\phi^2}y + \frac{\dot{\phi}\cos\phi}{\phi}y = \frac{\sin\phi}{\phi}\dot{y} + \left(\frac{\cos\phi - \frac{\sin\phi}{\phi}}{\phi^2}\right)(\dot{y} \cdot y)\,y \tag{222}$$

When ϕ goes to zero, we can approximate $\frac{\sin\phi}{\phi}$ with the series in (190) and the other singular factor as:

$$\frac{\cos\phi - \frac{\sin\phi}{\phi}}{\phi^2} = -\frac{1}{3} + \frac{\phi^2}{30} - \frac{\phi^4}{840} + \frac{\phi^6}{45360} + \dots \tag{223}$$

B.2.5 Spherical Linear Interpolation

Spherical Linear Interpolation, *SLERP*, interpolates between two quaternions. SLERP can be understood geometrically by considering a relative orientation in the axis-angle form.

Consider the relative quaternion q_r between two endpoints, $q_1 \otimes q_r = q_2$, given in axis angle form $(\hat{u}_r, \theta_r)$. To interpolate between q_1 and q_2, we apply the $q(\tau) = q_1 \otimes q_s(\tau)$, where q_s is a rotation about $\hat{u}_r$ with angle θ_s varying from 0 to θ_r as τ varies from 0 to 1. We can compute the rotation vector form of q_s from that of q_r as $v_s = \tau v_r$.

Composing definitions for quaternion and rotation vector conversion and quaternion exponents:

$$q(\tau) = q_1 \otimes \exp\left(\tau \ln\left({q_1}^* \otimes q_2\right)\right) = q_1 \otimes \left({q_1}^* \otimes q_2\right)^t \tag{224}$$

To interpolate in the shorter direction, e.g., $-\frac{\pi}{2}$ vs. $+\frac{3\pi}{2}$, scale ${q_1}^* \otimes q_2$ so it has a positive scalar element.

A more efficient computation for SLERP [160] is:

$$\phi = |\angle(\vec{q}_1, \vec{q}_2)| \tag{225}$$

$$\theta = \begin{cases} \phi > \frac{\pi}{2} & \pi - \phi \\ \phi \leq \frac{\pi}{2} & \phi \end{cases} \tag{226}$$

$$q(\tau) = \begin{cases} \phi > \frac{\pi}{2} & \frac{\sin\theta - \tau\theta}{\sin\theta} q_1 - \frac{\sin\tau\theta}{\sin\theta} q_2 \\ \phi \leq \frac{\pi}{2} & \frac{\sin\theta - \tau\theta}{\sin\theta} q_1 + \frac{\sin\tau\theta}{\sin\theta} q_2 \end{cases} \tag{227}$$

B.2.6 Integration

Euler or Runge-Kutta integration of quaternion derivatives would not preserve the unit constraint, introducing error. We can instead integrate a constant rotational velocity with:

$$q_1 = \exp\left(\frac{\omega \Delta t}{2}\right) \otimes q_0 \tag{228}$$

$$= \exp\left(\Delta t \, \dot{q} \otimes q_0^*\right) \otimes q_0 \tag{229}$$

B.2.7 Finite Difference

Based on (228), we can compute a finite difference velocity ω_Δ between two orientations:

$$\omega_\Delta = 2\ln\left(q_1 \otimes q_0^*\right) \tag{230}$$

$$\dot{q}_\Delta = \ln\left(q_1 \otimes q_0^*\right) \otimes q_0 \tag{231}$$

B.3 Dual Quaternions and Euclidean Transforms

Dual quaternions are convenient for representing Euclidean transformations. Formally, dual quaternions are the generalization of quaternions to dual number.

B.3.1 Dual Numbers

Dual numbers are similar to complex numbers, but the square of the dual element ε is zero:

$$\tilde{z} = a + b\varepsilon \tag{232}$$

$$\varepsilon \neq 0 \tag{233}$$

$$\varepsilon^2 = 0 \tag{234}$$

If we consider the Taylor series of $f(a+b\varepsilon)$ at point a, we obtain the following property:

$$f(a+b\varepsilon) = f(a) + bf'(a)\varepsilon \tag{235}$$

This lets us define a few functions for dual numbers:

$$\cos a + b\varepsilon = \cos a - \sin ab\varepsilon \tag{236}$$

$$\sin a + b\varepsilon = \sin a + \cos ab\varepsilon \tag{237}$$

$$\exp\left(a + b\varepsilon\right) = e^a + e^a b\varepsilon \tag{238}$$

$$\sqrt{a+b\varepsilon} = \sqrt{a} + \frac{b}{2\sqrt{a}}\varepsilon \tag{239}$$

B.3.2 Representation

Dual quaternions are quaternions with dual numbers for elements.

$$\begin{aligned}\mathcal{S} &= \\ \tilde{x}\imath + \tilde{y}\jmath + \tilde{z}k + \tilde{w} &= \\ (r_x + d_x\varepsilon)\imath + (r_y + d_y\varepsilon)\jmath + (r_z + d_z\varepsilon)k + (r_w + d_w\varepsilon) &= \\ (r_x\imath + r_y\jmath + r_z k + r_w) + (d_x\imath + d_y\jmath + d_z k + d_w)\varepsilon &= \\ r + d\varepsilon \end{aligned} \tag{240}$$

For computation, it is convenient to represent dual quaternion $\mathcal{S}$ factored into the separate real and dual parts r and d:

$$\begin{aligned}\mathcal{S} &= r + d\varepsilon \\ &= \Big(r,\ d\Big)\end{aligned} \tag{241}$$

B.3.3 Construction

We can produce a dual quaternion for some transformation represented by the rotational quaternion q, and the translation vector v as follows:

$$r = q \tag{242}$$

$$d = \frac{1}{2} v \otimes r \tag{243}$$

Translation v is augmented with 0 as the scalar element for the quaternion multiply. The real part r represents orientation, and the dual part d represents translation. Note that the real part r will be a unit quaternion while the dual part d has no such restriction.

To extract the translation, we do:

$$v = 2d \otimes r^* \tag{244}$$

B.3.4 Multiplication

Multiplication is defined in terms of the standard quaternion multiply, performed over both real and dual parts:

$$\mathcal{A} \otimes \mathcal{B} = \left(a_r \otimes b_r, \quad a_r \otimes b_d + a_d \otimes b_r \right) \tag{245}$$

B.3.5 Matrix Form

We can also represent the dual quaternion multiplication as a matrix multiply. Based on (178):

$$\begin{aligned} \mathcal{A} \otimes \mathcal{B} = \begin{pmatrix} a_r \otimes b_r \\ a_r \otimes b_d + a_d \otimes b_r \end{pmatrix} &= \\ \mathbf{A_L}\vec{B} = \begin{bmatrix} \mathbf{A_{r,L}} & 0 \\ \mathbf{A_{d,L}} & \mathbf{A_{r,L}} \end{bmatrix} \vec{B} &= \\ \mathbf{B_R}\vec{A} = \begin{bmatrix} \mathbf{B_{r,R}} & 0 \\ \mathbf{B_{d,R}} & \mathbf{B_{r,R}} \end{bmatrix} \vec{A} & \end{aligned} \tag{246}$$

B.3.6 Conjugate

$$\mathcal{S}^* = \left(s_r{}^*, \; s_d{}^* \right) \tag{247}$$

B.3.7 Exponential

We derive the dual quaternion exponential by expanding (189) using dual arithmetic:

$$\phi = |r_v| \tag{248}$$

$$k = r_v \cdot d_v \tag{249}$$

$$e^{\mathcal{S}} = e^{\tilde{w}} \left(\left(\frac{s}{\phi} r_v, \; c \right), \; \left(\frac{s}{\phi} d_v + \frac{c - \frac{s}{\phi}}{\phi^2} k r_v, \; -\frac{s}{\phi} k \right) \right) \tag{250}$$

where $\tilde{w} = r_w + d_w \varepsilon$.

Then, to handle the singularity at $\phi = 0$, we use (190) and:

$$\frac{\cos\phi - \frac{\sin\phi}{\phi}}{\phi^2} = -\frac{1}{3} + \frac{\phi^2}{30} - \frac{\phi^4}{840} + \frac{\phi^6}{45360} + \dots \tag{251}$$

B.3.8 Logarithm

We derive the dual quaternion logarithm by expanding (193) using dual arithmetic:

$$\phi = \operatorname{atan2}\left(|r_v|, r_w\right) \tag{252}$$

$$k = r_v \cdot d_v \tag{253}$$

$$\alpha = \frac{r_w - \frac{\phi}{|r_v|}|r|^2}{|r_v|^2} \tag{254}$$

$$(\ln S)_r = \left(\frac{\phi}{|r_v|} r_v,\ \ln|r|\right) \tag{255}$$

$$(\ln S)_d = \left(\frac{k\alpha - d_w}{|r|^2} r_v + \frac{\phi}{|r_v|} d_v,\ k + \frac{r_w d_w}{|r|^2}\right) \tag{256}$$

To handle the singularity at $|r_v| = 0$, we apply (194) and (195) to handle $\frac{\phi}{|r_v|}$. Then, we rewrite α as:

$$\begin{aligned}
\frac{r_w - \frac{\phi}{|r_v|}|r|^2}{|r_v|^2} &= \\
\frac{r_w}{|r_v|^2} - \frac{\phi|r|^2}{|r_v|^3} &= \\
\frac{r_w|r|^2}{|r_v|^2|r|^2} - \frac{\phi|r|^3}{|r_v|^3|r|} &= \\
\frac{1}{|r|}\left(\frac{r_w}{|r|}\frac{|r|^2}{|r_v|^2} - \phi\frac{|r|^3}{|r_v|^3}\right) &= \\
\frac{1}{|r|}\left(\frac{\cos\phi}{\sin^2(\phi)} - \frac{\phi}{\sin^3(\phi)}\right) &
\end{aligned} \tag{257}$$

This gives the Taylor series:

$$\frac{c}{s^2} - \frac{\phi}{s^3} = -\frac{2}{3} - \frac{1}{5}\phi^2 - \frac{17}{420}\phi^4 - \frac{29}{4200}\phi^6 + \dots \tag{258}$$

B.3.9 Chaining Transforms

Transforms are chained by multiplying the dual quaternions.

B.3.10 Transforming a point

We can transform a point v by constructing a dual quaternion for translation v and identity rotation, and chaining it onto the transform, then extracting the resulting translation:

$$\mathcal{S}' = \mathcal{S} \otimes \left((0,\ 1),\ \frac{1}{2}v \right) \tag{259}$$

$$v' = 2s'_d \otimes s'^*_r \tag{260}$$

This reduces to:

$$v' = (2s_d + s_r \otimes v) \otimes s^*_r \tag{261}$$

B.3.11 Derivatives

B.3.11.1 Product Rule

Because dual quaternion multiplication is a linear operation, the product rule applies:

$$\frac{d}{dt}(\mathcal{S}_1 \otimes \mathcal{S}_2) = \dot{\mathcal{S}}_1 \otimes \mathcal{S}_2 + \mathcal{S}_1 \otimes \dot{\mathcal{S}}_2 \tag{262}$$

B.3.11.2 Angular Velocity

Angular velocity computation is identical to the single unit quaternion case:

$$\dot{r} = \frac{1}{2}\omega \otimes r \tag{263}$$

$$\omega = 2\dot{r} \otimes r^* \tag{264}$$

B.3.11.3 Translational Velocity

We find the equation for the derivative of the dual part by differentiating (243),

$$\dot{d} = \frac{1}{2}(\dot{v} \otimes r + v \otimes \dot{r}) \tag{265}$$

Translational velocity comes from differentiating (244):

$$\dot{v} = 2(\dot{d} \otimes r^* + d \otimes (\dot{r})^*) \tag{266}$$

B.3.12 Integration

To integrate dual quaternions, we first introduce the *twist*, Ω:

$$\Omega = \Big((\omega,\ 0),\ (\dot{v} + v \times \omega,\ 0) \Big) \tag{267}$$

where ω is angular velocity, v is translation, and $\dot{v}$ is translational velocity.

Then, integration of a constant velocity is given by:

$$\mathcal{S}_1 = \exp\left(\frac{\Omega \Delta t}{2}\right) \otimes \mathcal{S}_0 \tag{268}$$

B.4 Implicit Dual Quaternions

We can implicitly represent the dual quaternion for a Euclidean transform by storing orientation quaternion r and translation vector v:

$$E = (r,\ v) \tag{269}$$

This form allows more efficient computation for some operations.

B.4.1 Chaining transforms

From dual quaternion multiplication (245), we derive the multiplication formula for the implicit form:

$$\begin{aligned} C_v = 2C_d \otimes C_r^* = \\ 2\left(A_r \otimes B_d + A_d \otimes B_r\right) \otimes \left(A_r \otimes B_r\right)^* = \\ 2\left(A_r \otimes \frac{B_v \otimes B_r}{2} + \frac{A_v \otimes A_r}{2} \otimes B_r\right) \otimes B_r^* \otimes A_r^* = \\ \left(A_r \otimes B_v + A_v \otimes A_r\right) \otimes A_r^* = \\ A_r \otimes B_v \otimes A_r^* + A_v \end{aligned}$$

This is equivalent to rotating B_v by A_r, then adding A_v. Thus, we chain transforms with:

$$C_r = A_r \otimes B_r \tag{270}$$

$$C_v = \operatorname{rot}(A_r, B_v) + A_v \tag{271}$$

B.4.2 Transforming points

To transform point p, we first rotate it by the given orientation r, then add the translation v

$$p' = \operatorname{rot}(r, p) + v \tag{272}$$

B.4.3 Conjugate

From the dual quaternion conjugate (247) for $S = (r, d)$:

$$\begin{aligned}(S^*)_v = 2(S^*)_d \otimes ((S^*)_r)^* =\\ 2d^* \otimes (r^*)^* =\\ 2(\frac{1}{2}v \otimes r)^* \otimes r =\\ (v \otimes r)^* \otimes r =\\ r^* \otimes v^* \otimes r =\\ -\operatorname{rot}(r^*, v)\end{aligned}$$

Thus, to find the conjugate translation, we rotate v by r^* and negate.

B.4.4 Derivatives

The transform chaining in (271) is not linear, so we cannot apply the product rule. Instead, we directly differentiate (271):

$$\frac{d}{dt}\left(\mathscr{S}_{\mathrm{i}}\begin{pmatrix} r_1 \\ v_1 \end{pmatrix} \otimes \mathscr{S}_{\mathrm{i}}\begin{pmatrix} r_2 \\ v_2 \end{pmatrix}\right) =$$

$$\mathscr{S}_{\mathrm{i}}\begin{pmatrix} \dot{r}_1 \otimes r_2 + r_1 \otimes \dot{r}_2 \\ \dot{v}_1 + \dot{q}_1 \otimes v_2 \otimes q_1^* + q_1 \otimes \dot{v}_2 \otimes q_1^* + q_1 \otimes v_2 \otimes \dot{q}_1^* \end{pmatrix} \tag{273}$$

B.5 History

Quaternions were invented in the mid-nineteenth century by William Rowan Hamilton, who spent the rest of his life exploring their properties. They quickly found use among physicists; Maxwell's equations were originally formulated using quaternions.

Around the turn of the twentieth century, Josiah Gibbs published his *Vector Analysis*, presented as a simplification over quaternions. The chief distinction was the invention of the dot and cross product operators, splitting quaternion multiplication into two separate operations. Eventually, Gibbs's notation overtook quaternions as the representation of choice among physicists and engineers.

Though quaternions may have lost the overall popularity contest to Gibbs's vector analysis, their useful numerical properties mean quaternions still have some role to play.

REFERENCES

[1] ABB, *Operating manual, RobotStudio*, 5.15k ed., 2013. Document ID: 3HAC032104-001.

[2] AGÜERO, C., CAÑAS, J., MARTÍN, F., and PERDICES, E., "Behavior-based iterative component architecture for soccer applications with the NAO humanoid," in *5th Workshop on Humanoids Soccer Robots. Nashville, TN, USA*, 2010.

[3] AHN, J.-S., CHUNG, W.-J., and JUNG, C.-D., "Realization of orientation interpolation of 6-axis articulated robot using quaternion," *Journal of Central South University*, vol. 19, no. 12, pp. 3407–3414, 2012.

[4] AHO, A., LAM, M., SETHI, R., and ULLMAN, J., *Compilers: Principles, Techniques, & Tools.* Pearson, 2nd ed., 2007.

[5] ALLEN, J. F. and FERGUSON, G., "Actions and events in interval temporal logic," *Journal of logic and computation*, vol. 4, no. 5, pp. 531–579, 1994.

[6] ALUR, R., COURCOUBETIS, C., HENZINGER, T., and HO, P., "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Hybrid Systems*, pp. 209–229, 1993.

[7] ALUR, R., DANG, T., ESPOSITO, J., HUR, Y., IVANCIC, F., KUMAR, V., MISHRA, P., PAPPAS, G., and SOKOLSKY, O., "Hierarchical modeling and analysis of embedded systems," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 11–28, 2003.

[8] AMES, A. D., "First steps toward automatically generating bipedal robotic walking from human data," in *Robotic Motion and Control 2011*, vol. 422 of *LNICS*, pp. 89–116, Springer, 2012.

[9] AMES, A. D., COUSINEAU, E. A., and POWELL, M. J., “Dynamically stable bipedal robotic walking with NAO via human-inspired hybrid zero dynamics,” in *Hybrid Systems: Computation and Control*, (Beijing), pp. 135–44, April 2012.

[10] ANDO, N., SUEHIRO, T., KITAGAKI, K., KOTOKU, T., and YOON, W., “RT-middleware: distributed component middleware for RT (robot technology),” in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 3933–3938, IEEE, 2005.

[11] ARGALL, B., CHERNOVA, S., VELOSO, M., and BROWNING, B., “A survey of robot learning from demonstration,” *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[12] BACCHUS, F. and YANG, Q., “Downward refinement and the efficiency of hierarchical problem solving,” *Artificial Intelligence*, vol. 71, pp. 43–100, 1993.

[13] BAIER, C., KATOEN, J., and OTHERS, *Principles of Model Checking*. MIT Press, Cambridge, MA, 2008.

[14] BELTA, C. and KUMAR, V., “Abstraction and control for groups of robots,” *IEEE Transactions on Robotics*, vol. 20, no. 5, pp. 865–875, 2004.

[15] BEN AMOR, H., KROEMER, O., HILLENBRAND, U., NEUMANN, G., and PETERS, J., “Generalization of human grasping for multi-fingered robot hands,” in *Proceedings of the International Conference on Robot Systems (IROS)*, 2012.

[16] BERENSON, D., SRINIVASA, S. S., FERGUSON, D., and KUFFNER, J. J., “Manipulation planning on constraint manifolds,” in *Intl. Conf. on Robotics and Automation*, pp. 625–632, IEEE, 2009.

[17] BILLARD, A., CALINON, S., DILLMANN, R., and SCHAAL, S., *Handbook of Robotics Chapter 59: Robot Programming by Demonstration*. Springer, 2007.

[18] BLUM, A. L. and FURST, M. L., "Fast planning through planning graph analysis," *Artificial intelligence*, vol. 90, no. 1, pp. 281–300, 1997.

[19] BROCKETT, R., "Formal languages for motion description and map making," *Robotics*, vol. 41, pp. 181–191, 1990.

[20] BROOKS, R., "Elephants don't play chess," *Robotics and autonomous systems*, vol. 6, no. 1-2, pp. 3–15, 1990.

[21] BROWN, J. H. and MARTIN, B., "How fast is fast enough? Choosing between Xenomai and Linux for real-time applications," tech. rep., Rep Invariant Systems, 2010. `https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf`.

[22] BRUYNINCKX, H., SOETENS, P., and KONINCKX, B., "The real-time motion control core of the Orocos project," in *Intl. Conference on Robotics and Automation*, vol. 2, pp. 2766–2771, IEEE, 2003.

[23] BRZOZOWSKI, J., "Canonical regular expressions and minimal state graphs for definite events," *Mathematical theory of Automata*, vol. 12, pp. 529–561, 1962.

[24] CABALAR, P., OTERO, R. P., CABARCOS, M., and BARREIRO, A., "Introducing planning in discrete event systems," in *Computer Aided Systems Theory*, pp. 146–159, Springer, 1997.

[25] CASSANDRAS, C. and LAFORTUNE, S., *Introduction to Discrete-Event Systems*. Springer, 2nd ed., 2008.

[26] CHAMPARNAUD, J., KHORSI, A., and PARANTHOËN, T., "Split and join for minimizing: Brzozowskis algorithm," *Proc. of PSC*, vol. 2, pp. 96–104, 2002.

[27] CHAO, C., CAKMAK, M., and THOMAZ, A., "Towards grounding concepts for transfer in goal learning from demonstration," in *Intl. Conf. on Development and Learning*, 2011.

[28] CHAUMETTE, F. and HUTCHINSON, S., "Visual servo control, part I: Basic approaches," *Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[29] CHAUMETTE, F. and HUTCHINSON, S., "Visual servo control, part II: Advanced approaches," *Robotics and Automation Magazine*, vol. 14, no. 1, pp. 109–118, 2007.

[30] CHO, B.-K., PARK, S.-S., and HO OH, J., "Controllers for running in the humanoid robot, Hubo," in *Intl. Conf. on Humanoid Robots*, 2009.

[31] CHOI, C. and CHRISTENSEN, H. I., "Robust 3D visual tracking using particle filtering on the special Euclidean group: A combined approach of keypoint and edge features," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 498–519, 2012.

[32] CHOUKROUN, D., BAR-ITZHACK, I. Y., and OSHMAN, Y., "Novel quaternion Kalman filter," *Trans. on Aerospace and Electronic Systems*, vol. 42, no. 1, pp. 174–190, 2006.

[33] CIMATTI, A., CLARKE, E., GIUNCHIGLIA, E., GIUNCHIGLIA, F., PISTORE, M., ROVERI, M., SEBASTIANI, R., and TACCHELLA, A., "NuSMV 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, pp. 359–364, Springer, 2002.

[34] September 2012. http://clang.llvm.org/features.html.

[35] CORKE, P. I., *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer, 2011.

[36] Cormen, T., Leiserson, C., Rivest, R., and Stein, C., *Introduction to algorithms*. MIT press, 2001.

[37] Craig, J., *Introduction to Robotics: Mechanics and Control*. Pearson, 3rd ed., 2005.

[38] Cresswell, S. and Coddington, A. M., "Compilation of LTL goal formulas into PDDL," in *European Conference on Artificial Intelligence*, 2004.

[39] Dam, E. B., Koch, M., and Lillholm, M., *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998.

[40] Dantam, N., Amor, H. B., Christensen, H., and Stilman, M., "Online camera registration for robot manipulation (presented)," in *International Symposium on Experimental Robotics*, 2014.

[41] Dantam, N., Hereid, A., Ames, A., and Stilman, M., "Correct software synthesis for stable speed-controlled walking," in *RSS*, IEEE, 2013.

[42] Dantam, N., Kolhe, P., and Stilman, M., "The motion grammar for physical human-robot games," in *ICRA*, IEEE, 2011.

[43] Dantam, N. and Stilman, M., "The motion grammar: Linguistic perception, planning, and control," in *RSS*, IEEE, 2011.

[44] Dantam, N. and Stilman, M., "The motion grammar calculus for context-free hybrid systems," in *ACC*, 2012.

[45] Dantam, N. and Stilman, M., "The motion grammar calculus for context-free hybrid systems," in *ACC*, IEEE, 2012.

[46] Dantam, N. and Stilman, M., "Robustness and efficient communication for real-time multi-process robot software," in *Humanoids*, IEEE, 2012.

[47] DANTAM, N. and STILMAN, M., "The motion grammar: Analysis of a linguistic method for robot control," *Transactions on Robotics*, vol. 29, no. 3, pp. 704–718, 2013.

[48] DANTAM, N., LOFARO, D., HEREID, A., OH, P., AMES, A., and STILMAN, M., "Multiprocess communication and control software for humanoid robots (accepted)," *Robotics and Automation Magazine*, 2014.

[49] DANTAM, N. T., HEREID, A., AMES, A., and STILMAN, M., "Correct software synthesis for stable speed-controlled robotic walking," in *Robotics: Science and Systems*, June 2013.

[50] DE GIACOMO, G. and VARDI, M. Y., "Automata-theoretic approach to planning for temporally extended goals," in *Recent Advances in AI Planning*, pp. 226–238, Springer, 2000.

[51] DE LA HIGUERA, C., *Grammatical Inference*. Cambridge University Press, 2010.

[52] DE LUCA, A., ALBU-SCHAFFER, A., HADDADIN, S., and HIRZINGER, G., "Collision detection and safe reaction with the DLR-III lightweight manipulator arm," in *IROS*, pp. 1623–1630, IEEE/RSJ, 2006.

[53] DEMPSTER, A. P., LAIRD, N. M., and RUBIN, D. B., "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[54] DEREMER, F., "Simple LR(k) grammars," *Communications of the ACM*, vol. 14, no. 7, pp. 453–460, 1971.

[55] DEREMER, F. and MAC., M. I. O. T. C. P., *Practical translators for LR(k) languages*. PhD thesis, Massachusetts Institute of Technology, 1969.

[56] DIEBEL, J., "Representing attitude: Euler angles, unit quaternions, and rotation vectors," tech. rep., Stanford University, 2006. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.5134`.

[57] DING, J., GILLULA, J., HUANG, H., VITUS, M., ZHANG, W., TOMLIN, C., GILLULA, J., HOFFMANN, G., HUANG, H., VITUS, M., and OTHERS, "Toward reachability-based controller design for hybrid systems in robotics," in *9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, vol. 45, pp. 2526–2536, 2011.

[58] DOYLE, J. C., "Guaranteed margins for lqg regulators," *IEEE Trans. on Automatic Control*, vol. 23, no. 4, pp. 756–757, 1978.

[59] EARLEY, J., "An efficient context-free parsing algorithm," *Communications of the ACM*, vol. 13, no. 2, pp. 94–102, 1970.

[60] EKVALL, S. and KRAGIC, D., "Learning task models from multiple human demonstrations," in *ROMAN*, pp. 358–363, IEEE, 2006.

[61] Electronic Industries Alliance, *Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines*, February 1979.

[62] EROL, K., HENDLER, J., and NAU, D. S., "Htn planning: Complexity and expressivity," in *AAAI*, vol. 94, pp. 1123–1128, 1994.

[63] ESPARZA, J., KUCERA, A., and SCHWOON, S., "Model checking LTL with regular valuations for pushdown systems," *Information and Computation*, vol. 186, no. 2, pp. 355–376, 2003.

[64] FAINEKOS, G., GIRARD, A., KRESS-GAZIT, H., and PAPPAS, G., "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[65] Fikes, R. E. and Nilsson, N. J., "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[66] Finkel, A., Willems, B., and Wolper, P., "A direct symbolic approach to model checking pushdown systems," *Electronic Notes in Theoretical Computer Science*, vol. 9, pp. 27–37, 1997.

[67] Folkesson, J. and Christensen, H., "Graphical SLAM - a self-correcting map," *Intl. Conf. on Robotics and Automation*, 2004.

[68] Frazzoli, E., Dahleh, M., and Feron, E., "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.

[69] Fu, K., *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1981.

[70] July 2010. http://gcc.gnu.org/gcc-3.4/changes.html.

[71] Gibson, J. J., *The senses considered as perceptual systems*. Boston: Houghton Mifflin, 1966.

[72] GmbH, S., "Dextrous lightweight arm LWA 4D, technical data." `http://mobile.schunk-microsite.com/en/produkte/produkte/dextrous-lightweight-arm-lwa-4d.html`.

[73] Gropp, W., Lusk, E., and Skjellum, A., *Using MPI: portable parallel programming with the message passing interface*. MIT Press, Cambridge, MA, 1999.

[74] Haghverdi, E., Tabuada, P., and Pappas, G., "Bisimulation relations for dynamical, control, and hybrid systems," *Theoretical Computer Science*, vol. 342, no. 2-3, pp. 229–261, 2005.

[75] HAMPEL, F. R., RONCHETTI, E. M., ROUSSEEUW, P. J., and STAHEL, W. A., *Robust statistics: the approach based on influence functions*, vol. 114. John Wiley & Sons, 2011.

[76] HAN, F. and ZHU, S., "Bottom-up/top-down image parsing by attribute graph grammar," in *ICCV*, vol. 2, 2005.

[77] HAREL, D., "On visual formalisms," *Communications of the ACM*, vol. 31, no. 5, pp. 514–530, 1988.

[78] HAUSER, K., "Fast interpolation and time-optimization on implicit contact submanifolds," in *Robotics: Science and Systems*, (Berlin, Germany), June 2013.

[79] HEBERT, P., HUDSON, N., MA, J., and BURDICK, J. W., "Dual arm estimation for coordinated bimanual manipulation," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 120–125, IEEE, 2013.

[80] HELMERT, M., RÖGER, G., and KARPAS, E., "Fast downward stone soup: A baseline for building planner portfolios," in *ICAPS 2011 Workshop on Planning and Learning*, pp. 28–35, 2011.

[81] HENZINGER, T., "The theory of hybrid automata," in *Logic in Computer Science*, pp. 278–292, IEEE, 1996.

[82] HOARE, C., "Report on the elliott algol translator," *The Computer Journal*, vol. 5, no. 2, pp. 127–129, 1962.

[83] HOFFMANN, J. and NEBEL, B., "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[84] HOLTZMAN, G., *The Spin Model Checker*. Addison Wesley, Boston, MA, 2004.

[85] HOPCROFT, J., "An n log n algorithm for minimizing states in a finite automaton," *Reproduction*, pp. 189–196, 1971.

[86] HOPCROFT, J. and ULLMAN, J., *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, MA, 1979.

[87] HRISTU-VARSAKELIS, D., EGERSTEDT, M., and KRISHNAPRASAD, P., "On the structural complexity of the motion description language mdle," in *CDC*, pp. 3360–3365, IEEE, 2003.

[88] HRISTU-VARSAKELIS, D. and LEVINE, W., eds., *Handbook of Networked and Embedded Control Systems*. Birkhauser, 2005.

[89] HUANG, A. S., OLSON, E., and MOORE, D. C., "LCM: Lightweight communications and marshalling," in *Intelligent Robots and Systems*, pp. 4057–4062, IEEE, 2010.

[90] HUBER, P. J. and OTHERS, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[91] HYATT, R., "CRAFTY–chess program," *ftp://ftp.cis.uab.edu/pub/hyatt*, 1996.

[92] IROBOT CORPORATION, "Aware 2.0." http://www.irobot.com/gi/developers/Aware/.

[93] IVANOV, Y. and BOBICK, A., "Recognition of visual activities and interactions by stochastic parsing," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 852–872, 2000.

[94] JOHNSON, S. and BELL TELEPHONE LABORATORIES, I., *Yacc: Yet another compiler-compiler*. Bell Laboratories, 1975.

[95] JOO, S. and GREY, M., "DRC-Hubo retrospective," January 2014. Personal Communication.

[96] KAHAN, W., "How futile are mindless assessments of roundoff in floating-point computation," tech. rep., U.C. Berkeley, 2006. `http://www.cs.berkeley.edu/~wkahan/Mindless.pdf`.

[97] KANG, I. and PARK, F., "Cubic spline algorithms for orientation interpolation," *International journal for numerical methods in engineering*, vol. 46, no. 1, pp. 45–64, 1999.

[98] KARAMAN, S. and FRAZZOLI, E., "Sampling-based motion planning with deterministic μ-calculus specifications," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pp. 2222–2229, IEEE, 2009.

[99] KASAMI, T., "An efficient recognition and syntax analysis algorithm for context-free languages," Tech. Rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

[100] KAUTZ, H. and SELMAN, B., "Blackbox: A new approach to the application of theorem proving to problem solving," in *AIPS98 Workshop on Planning as Combinatorial Search*, vol. 58260, pp. 58–60, 1998.

[101] KELLY, T., WANG, Y., LAFORTUNE, S., and MAHLKE, S., "Eliminating concurrency bugs with control engineering," *Computer*, vol. 42, no. 12, pp. 52–60, 2009.

[102] KIENTZ, J. A., PATEL, S. N., JONES, B., PRICE, E., MYNATT, E. D., and ABOWD, G. D., "The Georgia Tech aware home," in *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, (New York, NY, USA), ACM, 2008.

[103] KIM, M.-J., KIM, M.-S., and SHIN, S. Y., "A general construction scheme for unit quaternion curves with simple high order derivatives," in *Computer Graphics and Interactive Techniques*, pp. 369–376, ACM, 1995.

[104] Klavins, E., "A language for modeling and programming cooperative control systems," in *ICRA*, vol. 4, pp. 3403–3410, IEEE, 2004.

[105] Klingensmith, M., Galluzzo, T., Dellin, C., Kazemi, M., Bagnell, J. A. D., and Pollard, N., "Closed-loop servoing using real-time markerless arm tracking," in *International Conference on Robotics And Automation (Humanoids Workshop)*, May 2013.

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

[107] Knoblock, C. A., "Automatically generating abstractions for planning," *Artificial intelligence*, vol. 68, no. 2, pp. 243–302, 1994.

[108] Knuth, D., "On the translation of languages from left to right," *Information and control*, vol. 8, no. 6, pp. 607–639, 1965.

[109] Košecká, J. and Bajcsy, R., "Discrete event systems for autonomous mobile agents," *Robotics and Autonomous Systems*, vol. 12, no. 3, pp. 187–198, 1994.

[110] Koutsourakis, P., Simon, L., Teboul, O., Tziritas, G., and Paragios, N., "Single view reconstruction using shape grammars for urban environments," in *ICCV*, 2009.

[111] Kress-Gazit, H., Fainekos, G., and Pappas, G., "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[112] Kröger, T., *On-Line Trajectory Generation in Robotic Systems*, vol. 58 of *Springer Tracts in Advanced Robotics*. Berlin, Heidelberg, Germany: Springer, January 2010.

[113] KUNZ, T. and STILMAN, M., "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Robotics: Science and Systems*, pp. 09–13, July 2012.

[114] LAVIOLA, J. J., "A comparison of unscented and extended Kalman filtering for estimating quaternion motion," in *American Control Conference*, vol. 3, pp. 2435–2440, IEEE, 2003.

[115] LEE, E. A., "Computing needs time," *Communications of the ACM*, vol. 52, pp. 70–79, May 2009.

[116] LEFFERTS, E. J., MARKLEY, F. L., and SHUSTER, M. D., "Kalman filtering for spacecraft attitude estimation," *Journal of Guidance, Control, and Dynamics*, vol. 5, no. 5, pp. 417–429, 1982.

[117] LEKAVỲ, M. and NÁVRAT, P., "Expressivity of strips-like and htn-like planning," in *Agent and Multi-Agent Systems: Technologies and Applications*, pp. 121–130, Springer, 2007.

[118] LEWIS II, P. and STEARNS, R., "Syntax-directed transduction," *Journal of the ACM (JACM)*, vol. 15, no. 3, pp. 465–488, 1968.

[119] LIVINGSTON, S. C., PRABHAKAR, P., JOSE, A. B., and MURRAY, R. M., "Patching task-level robot controllers based on a local μ-calculus formula," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 4588–4595, IEEE, 2013.

[120] LOFARO, D., *Unified Algorithmic Framework for High Degree of Freedom Complex Systems and Humanoid Robots*. PhD thesis, Drexel University, College of Engineering, May 2013.

[121] LOZANO-PÉREZ, T. and BROOKS, R., "An approach to automatic robot programming," Tech. Rep. A.I. Memo 842, Massachusetts Intitute of Technology, 1985.

[122] LYGEROS, J., JOHANSSON, K., SIMIC, S., ZHANG, J., and SASTRY, S., "Dynamical properties of hybrid automata," *IEEE Trans. on Automatic Control*, vol. 48, no. 1, pp. 2–17, 2003.

[123] LYONS, D. M. and ARBIB, M. A., "A formal model of computation for sensory-based robotics," *Robotics and Automation, IEEE Transactions on*, vol. 5, no. 3, pp. 280–293, 1989.

[124] MALI, A. D. and KAMBHAMPATI, S., "Encoding htn planning in propositional logic.," in *AIPS*, pp. 190–198, 1998.

[125] MALY, M., LAHIJANIAN, M., KAVRAKI, L. E., KRESS-GAZIT, H., and VARDI, M. Y., "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, (Philadelphia, PA, USA), pp. 353–362, ACM, ACM, 08/04/2013 2013.

[126] MARKLEY, F. L., CHENG, Y., CRASSIDIS, J. L., and OSHMAN, Y., "Averaging quaternions," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 4, pp. 1193–1197, 2007.

[127] MCCONNELL, S., *Software estimation: demystifying the black art*. Microsoft press, March 2006.

[128] MICROSOFT CORPORATION, "Microsoft robotics studio." http://www.microsoft.com/robotics/.

[129] MINNEN, D., ESSA, I., and STARNER, T., "Expectation grammars: Leveraging high-level expectations for activity recognition," in *Computer Vision and Pattern Recognition*, vol. 2, pp. II–626, IEEE, 2003.

[130] MITCHELL, I., BAYEN, A., and TOMLIN, C., "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *IEEE Trans. on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[131] MOORE, D. and ESSA, I., "Recognizing multitasked activities from video using stochastic context-free grammar," in *National Conf. on Artificial Intelligence*, pp. 770–776, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[132] MORRISETT, G., HARPER, R., and OTHERS, "Semantics of memory management for polymorphic languages," *Higher-Order Operational Techniques in Semantics*, pp. 175–226, 1996.

[133] NAKAMURA, Y. and HANAFUSA, H., "Inverse kinematics solutions with singularity robustness for robot manipulator control," *Journal of Dynamic Systems, Measurement, and Control*, no. 108, pp. 163–171, 1986.

[134] NIETO-GRANDA, C., ROGERS III, J. G., TREVOR, A. J. B., and CHRISTENSEN, H. I., "Semantic map partitioning in indoor environments using regional analysis," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, (Taiwan), pp. 1451–1456, IEEE, Oct 2010.

[135] NILSSON, N. J., *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998.

[136] The Object Management Group, *Data Distribution Service for Real-time Systems*, 1.2 ed., January 2007. `http://www.omg.org/spec/DDS/1.2/`.

[137] The Object Management Group, *Common Object Request Broker Architecture (CORBA/IIOP)*, 3.1.1 ed., August 2011. `http://www.omg.org/spec/CORBA/3.1.1/`.

[138] The Object Management Group, *Systems Modeling Language (SysML)*, June 2012. http://www.omg.org/spec/SysML/1.3/.

[139] O'CALLAGHAN, S., RAMOS, F. T., and DURRANT-WHYTE, H. F., "Contextual occupancy maps using gaussian processes.," in *IEEE Intl. Conf. on Robotics and Automation*, pp. 1054–1060, IEEE, 2009.

[140] OGALE, A. S., KARAPURKAR, A., and ALOIMONOS, Y., "View-invariant modeling and recognition of human actions using grammars," in *Dynamical vision*, pp. 115–126, Springer, 2007.

[141] *OpenCV API Reference*. `http://docs.opencv.org/master/modules/refman.html`.

[142] PARR, T. and FISHER, K., "LL (*): the foundation of the ANTLR parser generator," in *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, pp. 425–436, ACM, 2011.

[143] PETERSON, J. L., *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[144] PLAKU, E., KAVRAKI, L., and VARDI, M., "Falsification of ltl safety properties in hybrid systems," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 4, pp. 305–320, 2013.

[145] POWELL, M. J., HEREID, A., and AMES, A. D., "Speed regulation in 3D robotic walking through motion transitions between human-inspired partial hybrid zero dynamics," in *IEEE Intl. Conf. Robotics and Automation*, (Karlsruhe), 2013.

[146] POWELL, M. J., SINNET, R. W., and AMES, A. D., “3D human-inspired robotic walking: Optimization, speed regulation and implementation,” *Intl. J. of Adv. Robotic Systems*, 2012. Submitted Aug. 2012, available upon request.

[147] PRADEEP, V., KONOLIGE, K., and BERGER, E., “Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach,” in *Experimental Robotics*, pp. 211–225, Springer, 2014.

[148] PRAJNA, S. and JADBABAIE, A., “Safety verification of hybrid systems using barrier certificates,” *HSCC*, pp. 271–274, 2004.

[149] QUIGLEY, M., GERKEY, B., CONLEY, K., FAUST, J., FOOTE, T., LEIBS, J., BERGER, E., WHEELER, R., and NG, A., “ROS: an open-source robot operating system,” in *Intl. Conf. on Robotics and Automation, Workshop on Open Source Robotics*, IEEE, 2009.

[150] RAINIO, K. and BOYER, A., *ALVAR – A Library for Virtual and Augmented Reality User’s Manual*. VTT Augmented Reality Team, December 2013.

[151] RAMADGE, P. J. and WONHAM, W. M., “Supervisory control of a class of discrete event processes,” *Analysis and Optimization of Systems*, vol. 25, pp. 206–230, January 1987.

[152] RATHER, E. D., COLBURN, D. R., and MOORE, C. H., “The evolution of forth,” in *ACM Sigplan Notices*, vol. 28, pp. 177–199, ACM, 1993.

[153] RAWAL, C., TANNER, H. G., and HEINZ, J., “(sub)regular robotic languages,” in *Mediterranean Conf. on Control and Automation*, IEEE, 2011.

[154] ROTEM-GAL-OZ, A., “Fallacies of distributed computing explained,” tech. rep., Sun Microsystems, 2006. `http://www.rgoarchitects.com/Files/fallacies.pdf`.

[155] Şahin, E., Çakmak, M., Doğar, M., Uğur, E., and Üçoluk, G., "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, 2007.

[156] Schaal, S., Ijspeert, A., and Billard, A., "Computational approaches to motor learning by imitation," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 358, no. 1431, pp. 537–547, 2003.

[157] Schewe, S., "Beyond hyper-minimisation—minimising dbas and dpas is np-complete.," in *FSTTCS*, pp. 400–411, 2010.

[158] Schmidt, D. C., Levine, D. L., and Mungee, S., "The design of the TAO real-time object request broker," *Computer Communications*, vol. 21, no. 4, pp. 294–324, 1998.

[159] Shin, K. and McKay, N., "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Trans. on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.

[160] Shoemake, K., "Animating rotation with quaternion curves," *ACM SIGGRAPH computer graphics*, vol. 19, no. 3, pp. 245–254, 1985.

[161] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G., *Robotics: modelling, planning and control.* Springer Verlag, 2009.

[162] Silberschatz, A., Galvin, P. B., and Gagne, G., *Operating system concepts.* J. Wiley & Sons, 2009.

[163] Smith, R. and Cheeseman, P., "On the representation and estimation of spatial uncertainty," *Intl. Journal of Robotics Research*, vol. 5, pp. 56–68, Winter 1987.

[164] Srinivasan, R., *RPC: Remote Procedure Call Protocol Specification Version 2.* Internet Engineering Task Force, August 1995. http://www.ietf.org/rfc/rfc1831.txt.

[165] Srivastava, S., Immerman, N., and Zilberstein, S., “Applicability conditions for plans with loops: Computability results and algorithms,” *Artificial Intelligence*, vol. 191, pp. 1–19, 2012.

[166] Staiger, L., “Finite-state ω-languages,” *Journal of Computer and System Sciences*, vol. 27, no. 3, pp. 434–448, 1983.

[167] Stevens, W. R. and Rago, S. A., *Advanced Programming in the UNIX Environment*. Addison Wesley, Boston, MA, 2 ed., 2005.

[168] Stilman, B., *Linguistic Geometry: From Search to Construction*. Kluwer Academic Publishers, 2000.

[169] Stilman, M., Olson, J., and Gloss, W., “Golem Krang: Dynamically stable humanoid robot for mobile manipulation,” in *Intl. Conference on Robotics and Automation*, pp. 3304–3309, IEEE, 2010.

[170] Stilman, M., “Task constrained motion planning in robot joint space,” in *Intl. Conf. on Intelligent Robots and Systems*, pp. 3074–3081, October 2007.

[171] Stilman, M., “Global manipulation planing in robot joint space with task constraints,” *Trans. on Robotics*, vol. 26, no. 3, pp. 576–584, 2010.

[172] Tabuada, P. and Pappas, G. J., “Linear time logic control of discrete-time linear systems,” *Automatic Control, IEEE Transactions on*, vol. 51, no. 12, pp. 1862–1877, 2006.

[173] Teichman, A., Miller, S., and Thrun, S., “Unsupervised intrinsic calibration of depth sensors via SLAM,” in *Robotics: Science and Systems (RSS)*, 2013.

[174] Tellex, S., Knepper, R. A., Li, A., Roy, N., , and Rus, D., “Asking for help using inverse semantics,” in *Robotics: Science and Systems*, June 2014.

[175] TOPP, E. A. and CHRISTENSEN, H. I., "Detecting region transitions for human-augmented mapping," *IEEE Transactions on Robotics*, pp. 1–5, 2010.

[176] TOPP, E. A. and CHRISTENSEN, H. I., "Topological modelling for human augmented mapping," in *IEEE/RSJ Intl.er Conf. on Intelligent Robots and Systems*, pp. 2257–2263, Oct. 2006.

[177] TOSHEV, A., MORDOHAI, P., and TASKAR, B., "Detecting and parsing architecture at city scale from range data," in *CVPR*, IEEE, 2010.

[178] TREVOR, A. J. B., ROGERS III, J. G., and CHRISTENSEN, H., "Planar surface slam with 3d and 2d sensors," in *IEEE Intl. Conf. on Robotics and Automation*, 2012.

[179] TREVOR, A. J. B., ROGERS III, J. G., NIETO-GRANDA, C., and CHRISTENSEN, H., "Tables, counters, and shelves: Semantic mapping of surfaces in 3d," in *IROS Workshop on Semantic Mapping and Autonomous Knowledge Acquisition*, 2010.

[180] ULUSOY, A., WONGPIROMSARN, T., and BELTA, C., "Incremental controller synthesis in probabilistic environments with temporal logic constraints," *The International Journal of Robotics Research*, p. 0278364913519000, 2014.

[181] UMEYAMA, S., "Least-squares estimation of transformation parameters between two point patterns," *Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991.

[182] US FOOD AND DRUG ADMINISTRATION, "Medical device recall report, FY2003 to FY2012."

[183] VIJAYKUMAR, R., VENKATARAMAN, S., DAKIN, G., and LYONS, D., "A task grammar approach to the structure and analysis of robot programs," in *Workshop on Languages for Automation*, IEEE, 1987.

[184] VINOSKI, S., “Advanced message queuing protocol,” *Internet Computing, IEEE*, vol. 10, no. 6, pp. 87–89, 2006.

[185] WANG, Y., KELLY, T., KUDLUR, M., LAFORTUNE, S., and MAHLKE, S., “Gadara: Dynamic deadlock avoidance for multithreaded programs,” in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pp. 281–294, USENIX Association, 2008.

[186] WINTER, D., “Cyber physical systems in aerospace – challenges and opportunities,” in *Safe and Secure Systems and Software Symposium*, Air Force Research Laboratory, June 2011. Keynote.

[187] YOUNGER, D., “Recognition and parsing of context-free languages in time n3*,” *Information and control*, vol. 10, no. 2, pp. 189–208, 1967.