The Motion Grammar: Linguistic Perception, Planning, and Control

Neil Dantam ⟨ntd@gatech.edu⟩ and Mike Stilman ⟨mstilman@cc.gatech.edu⟩
School of Interactive Computing, Center for Robotics and Intelligent Machines,
Georgia Institute of Technology, 801 Atlantic Dr. NW, Atlanta, GA 30332, USA

***Abstract*—We present and analyze the Motion Grammar: a novel unified representation for task decomposition, perception, planning, and control that provides both fast online control of robots in uncertain environments and the ability to guarantee completeness and correctness. The grammar represents a policy for the task which is parsed in real-time based on perceptual input. Branches of the syntax tree form the levels of a hierarchical decomposition, and the individual robot sensor readings are given by tokens. We implement this approach in the interactive game of Yamakuzushi on a physical robot resulting in a system that repeatably competes with a human opponent in sustained game-play for the roughly six minute duration of each match.**

I. Introduction

As robots come into increasing contact with humans, it is absolutely vital to *prove* that these potentially dangerous machines are *safe and reliable*. Furthermore, applying robots to increasingly complicated and varied tasks requires a tractable way to generate desired behaviors. Typical reactive strategies focus on efficiency rather than proving how the system will respond during complicated tasks with uncertain outcomes [29, 2]. Existing deliberative planners do provide guarantees but often simplify the control system or have prohibitive computational cost [31]. By representing perception, planning, and control of robotic systems using Context-Free Grammars (CFG), the *Motion Grammar* (MG) [9] enables robots to handle uncertainty in the outcomes of control actions through online parsing. Furthermore, MG makes it possible to prove that the system will correctly respond to all possible situations.

Imagine a robot searching for earthquake survivors. This robot must carefully remove pieces of rubble while ensuring that the larger structure does not collapse. It must also coordinate its efforts with other robots and human rescuers. We consider a simplified version of this scenario in the two-player game of Yamakuzushi. While the game is adversarial, the robot and human collaborate to safely disassemble a pile of Shogi pieces. The game contains many of the challenging elements of physical human-robot interaction. Both the robot and human make careful contact with pieces without causing them to fall. The robot slides pieces without losing contact. It also observes human actions and handles dangerous conditions. The key to all these interactions is that the robot must handle *uncertainty*. Our Motion Grammar guarantees that the robot responds to the complete range of uncertain outcomes online.

The Motion Grammar is a linguistic approach to robot control with significant benefits over existing techniques. The fast algorithms for Context-Free *parsing* enable the robot to quickly react online without lengthy deliberative planning. CFGs represent a balance between power and provability. This allows the system designer to tackle a broader class of problems and still prove desired response with model-checking. Most robot tasks can be recursively divided into a number of simpler subtasks. The hierarchical nature of grammatical *productions* and the corresponding *parse trees* are well suited to representing this hierarchical task decomposition. To our knowledge, these benefits are not found together in any prior methods for robot control

This paper discusses related work, the theory behind the Motion Grammar, and presents our experimental validation using the game of Yamakuzushi. Sect. III,IV formally define the Motion Grammar and explain the requirements for applying CFGs to robotic systems. Sect. V describes our application of the MG to the interactive game Yamakuzushi. Sect. VI analyzes provability and properties of the MG. Finally, Sect. VII introduces some of the numerous possible extensions to the MG approach.

II. Related Work

Literature on grammars from the Linguistic and Computer Science communities has a number applications related to robotics focusing primarily on image processing. [17, 19, 25, 34] use grammars to syntactically describe structural relations in images and point clouds. B. Stilman's Linguistic Geometry applies a syntactic approach to deliberative planning and search in adversarial games [33]. These works show that grammars are useful beyond their traditional role in the Linguistic, Theoretical, and Programming Language communities. Our approach applies grammars to online control of robotic systems.

Existing methods for planning and policy generation typically trade off efficiency for analytical properties such as completeness. Classical planners based on first-order logic [31] guarantee completeness. However, plan generation requires an NP-complete logical inference [31], and additionally symbolic methods do not explicitly address continuous domains. Partially Observable Markov Decision Processes explicitly represent domains with uncertainty, but their solvers are also NP-complete [27]. The MG provides a natural representation for hybrid continuous-discrete systems through parsing of Context-Free Languages (CFL). The online response for any event string has maximum $O(n^3)$ runtime [11]. Behavior-based methods are widely fielded approaches for efficient decision making [6, 2]. However, their primary validation has been experimental due to infeasible computation time requirements of earlier formal methods [29]. We argue that the common applications of behavior-based robotics including defense and personal assistance require formal verification. The MG provides an efficient control *policy*, and its structure can *guarantee* that robots will not cause accidental injury or damage.

There are many alternative formal control methods. Control of Discrete Event Systems is explained in [30, 7]. [22] describes a domain specific language for robot control whereas the MG describes the language produced by the robot itself. [23] uses graph grammars to organize many simple agents, while we use a CFG for one complex agent. We show that MG extends to correlated dimensions, human-robot interaction, and high-dimensional manipulation. Hybrid [7] and Maneuver [16] Automata switch continuous controllers in a Finite State Machine, whereas the MG uses CFGs which allow the controller to keep a memory of prior actions, build models, and improve decisions. Our language-theoretic approaches to analysis are complementary to existing methods for proving the stability of hybrid systems [35, 4]. The Motion Description Language (MDL) [5, 12, 13] describes robot operation with strings of symbols each representing continuous-valued controllers. These strings are generated off-line. Instead, the MG parses a string of tokenized sensor readings online. The string and associated parse tree evolve to represent the history of system execution. MDLe [28] also handles reactive planning; however, as shown in [21], it is not strictly more expressive than Hybrid Automata. More detailed relationships between MG and existing approaches are discussed in [10].

Model-checking approaches [3] are widely used to develop high-reliability systems. [15, 26, 24] use Linear Temporal Logic to formally describe uncertain multi-agent robotics by a finite state partitioning of the 2D environment. We adopt a discrete representation more suitable to high dimensional spaces; our manipulation task uses a 7-DOF robot and 40 movable objects making complete discretization computationally infeasible. Typically, model-checking uses a finite state model of the system. However, there are algorithms to check Context-Free systems as well [14]; we describe the specific language classes for which this is possible in Sect. VI-B.

III. The Motion Grammar

The Motion Grammar (MG) is a tool for designing and analyzing robot controllers through formal language. Results from language and automata theory are directly applicable to MG making it possible to prove correctness and completeness. This paper introduces an implementation of MG and analyzes those guarantees. First, we review formal language.

A. Review of Grammars and Automata

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 1 (Context Free Grammar, CFG):
$\Pi = (Z, V, P, S)$ where Z is an alphabet of symbols called *tokens*, V is a set of symbols called *nonterminals*, P is the set of mappings $V \mapsto (V \cup Z)^*$ called *productions*, and $S \in P$ is the *starting production*.

The productions of a CFG are conventionally written in Backus-Naur form, $A \to X_1 X_2 \ldots X_n$, where A is some nonterminal and $X_1 \ldots X_n$ is a sequence of tokens and nonterminals. This indicates that A may *expand* to all strings represented by the right-hand side of the production. For additional clarity, nonterminals may be represented between angle brackets $\langle\rangle$ and tokens between floor brackets $\lfloor\rfloor$.

Grammars have equivalent representations as *automata*. In the case of a Regular Grammar – where all productions are of the form $\langle \mathrm{A} \rangle \to \lfloor \mathrm{a} \rfloor \langle \mathrm{B} \rangle$, $\langle \mathrm{A} \rangle \to \lfloor \mathrm{a} \rfloor$, or $\langle \mathrm{A} \rangle \to \varepsilon$ – 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 to permit counting.

Definition 2 (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 3 (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, L_M, and M is said to *recognize* L_M.

Any string in a formal language can be represented as a *parse tree*. The root of the tree is the start symbol. 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. A production $A \to X_1 \ldots X_n$ will produce a piece of the parse tree with parent A and children $X_1 \ldots X_n$. The children of each node in the parse tree indicate which nonterminals or tokens that node *expands* to in a given string. The internal tree nodes are nonterminals, and tree leaves are tokens. The parse tree conveys the full syntactic structure of the string.

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).

B. Motion Grammar Definition

The Motion Grammar (MG) is a Syntax-Directed Definition (SDD) expressing the language of interaction between agents and real-world uncertain environments. In this paper, the agent is a robot and the example language represents the game of Yamakuzushi (Sect. V). Like SDDs for programming languages, the MG must have two components: *syntax* and *semantics*. This paper focuses on the *s*yntax of the MG, its expressivity, and formal analysis of MG languages. MG tokens are system states or discretized sensor readings. MG strings are histories of states during system execution. The syntax thus represents the ordering in which system events and states may occur. The semantics defines the response to those events. The

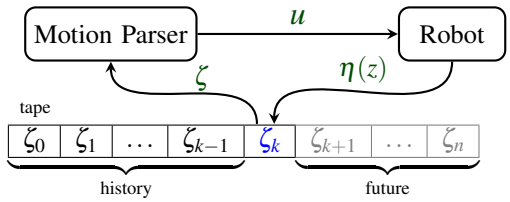


Fig. 1. Operation of the Motion Parser.

MG uses syntax to decide from the set of system behavior and semantics to interpret the state and make continuous control decisions. This SDD given by the MG is used to generate the *Motion Parser* which drives the robot as in Fig. 1.

Definition 4: The Motion Grammar (MG), $\mathscr{G}_M$, is a tuple $\mathscr{G}_M = (Z, V, P, S, \mathscr{Z}, \mathscr{U}, \eta, K)$:

Z	set of tokens representing robot state
V	set of nonterminals
P	set of productions
S	starting nonterminal, $S \in V$
$\mathscr{Z}$	space of robot sensor readings
$\mathscr{U}$	space of robot inputs
η	tokenizing function, $\eta : \mathscr{Z} \mapsto Z$
K	set of semantic rules, each associated with one and only one production.

Definition 5: The Motion Parser (MP) is a program that recognizes the language specified by the Motion Grammar and executes the corresponding semantic rules for each production.

Consider the illustration in Fig. 1. The output of the robot z is discretized into a stream of tokens ζ for the parser to read. Based on the sequence of tokens seen so far, 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 MG Grammar represents the language of robot sensor readings. The MP for that grammar is a *transducer* that translates the language of tokenized sensor readings into the language of controllers or actuator inputs.

C. Robot and Controller

The robot and the controller are both mappings between the input and sensor spaces.

Definition 6: The *robot* is a function $f : \mathscr{U} \mapsto \mathscr{Z}$ with internal state (q_f, x_f) where q_f is a discrete valued vector and $x_f \in \Re^m$.

Definition 7: The *controller* is a function $g : \mathscr{Z} \mapsto \mathscr{U}$ with internal state (q_g, x_g) where q_g is a discrete valued vector and $x_g \in \Re^n$.

The MG represents a language that is produced by the robot and consumed by the controller. The physical robotic system, coupled with the tokenizer η produces a string of tokenized sensor readings. The Motion Parser is the controller which uses semantic rules, K, to determine continuous input. Both discrete and continuous information are passed between the robot and the controller. Discrete events are generated by η, becoming part of the string. The continuous portion of controller state x_g comes from sensor readings $z \in \mathscr{Z}$ and semantic rules K. It is stored by the parser in the attributes of tokens and nonterminals.

```
1: procedure A
2:     Choose a production for A, A → X1 ... Xn
3:     for i = 1,n do
4:         if nonterminal? Xi  then call Xi
5:         else if Xi = η(z(t)) then continue
6:         else error
7:         end if
8:     end for
9:     Execute semantic rule for A → X1 ... Xn
10: end procedure
```

Fig. 2. Procedure for ⟨A⟩ in a recursive parser for $\mathscr{G}_M$

We emphasize that the MG is not a Domain Specific Language or Robot Programming Language [8, p339] but rather the direct application of linguistic theory to robot control. Furthermore, the MP does not require the Markov Assumption. It maintains a history or *memory* in parse tree nodes and node attributes or equivalently as the stack the pushdown automata.

IV. Grammars for Robotic Systems

A MG for any given task is developed based on the task specification and the robot hardware to be used. The spaces $\mathscr{U}$ and $\mathscr{Z}$ are the inputs and sensors that the robot possesses. The token set Z should be designed as the collection of events that may occur during task execution. The system designer must create the tokenizing function η to map from $\mathscr{Z}$ to Z. Then, the nonterminals V and productions P can be created by hierarchically decomposing the task into progressively simpler subtasks until finally bottoming out in a continuously valued control-loop. Once the productions have been created, the semantic rules K for each production can be developed. These rules will compute the input to the robot based on continuous values stored as node attributes. Finally, the starting nonterminal S is selected from V as the top level of the hierarchical decomposition and the grammar is complete.

A. Tokenizing

Tokens are events. These may be changes in a discrete state, timeouts, thresholds, or entry into regions where the continuous dynamics change. Whenever one of these events occurs, the parser is given the next token.

B. Parsing

Once the MG for the task is developed, it must be transformed into the Motion Parser. There are many different parsing techniques, however the online nature of the MG constrains this choice. First, the parser must expand productions left to right as that is the order in which it receives the tokens. Second, the input to give the robot must never be ambiguous; this limits the ability of the parser to lookahead and backtrack. For our proof-of-concept application, we used a hand-written *recursive descent parser*, an approach also employed by GCC [18]. A recursive descent parser is written as a set of mutually-recursive procedures, one for each nonterminal in the grammar. An example of one of these procedures is shown in Fig. 2, based on [1, p219]. Each procedure will fully expand its nonterminals via a top-down, left-to-right derivation. This approach is a good match for the MG's top-down task decomposition and its left-to-right temporal progression.

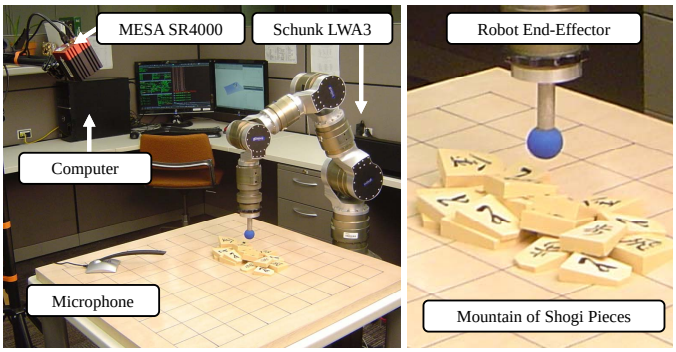


Fig. 3. Our experimental environment for linguistic physical human-robot games of Yamakuzushi.

Token	Description
$\lfloor \alpha \le t < \beta \rfloor$	Within time Range
$\lfloor$contact$\rfloor$	E.E. touching piece
$\lfloor$no contact$\rfloor$	not touching piece
$\lfloor$destination$\rfloor$	at traj. end
$\lfloor$human piece$\rfloor$	removed by human
$\lfloor$robot piece$\rfloor$	removed by robot
$\lfloor$clear$\rfloor$	board cleared
$\lfloor$sound$\rfloor$	noise removing piece
$\lfloor$quiet$\rfloor$	no noise made
$\lfloor$in space$\rfloor$	human in workspace
$\lfloor \neg$in space$\rfloor$	not in workspace
$\lfloor$point$\rfloor$	element of point cloud

(a) Tokens

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

(b) Attributes

Fig. 4. Tokens, Attributes for the Yama Motion Grammar

PRODUCTIONS	SEMANTIC RULES
$\langle g \rangle \rightarrow \langle g' \rangle$	$u = \dot{x}_r - K_p(x - x_r) - K_f(f - f_r)$
$\langle g' \rangle \rightarrow \lfloor t < 0 \rfloor$	$x_r = 0, \dot{x}_r = 0$
$\langle g' \rangle \rightarrow \lfloor 0 \le t < t_1 \rfloor$	$x_r = x_0 + \frac{1}{2}\ddot{x}_m t^2, \dot{x}_r = t\ddot{x}_m$
$\langle g' \rangle \rightarrow \lfloor t_1 \le t < t_2 \rfloor$	$x_r = x_0 + \frac{1}{2}\ddot{x}_m t_1^2 + \dot{x}_m(t - t_1), \dot{x}_r = \dot{x}_m$
$\langle g' \rangle \rightarrow \lfloor t_2 \le t < t_n \rfloor$	$x_r = x_n - \frac{1}{2}\ddot{x}_m(t_n - t)^2, \dot{x}_r = \dot{x}_m + \ddot{x}_m(t_2 - t)$
$\langle g' \rangle \rightarrow \lfloor t_n < t \rfloor$	$x_r = 0, \dot{x}_r = 0$

Fig. 5. Syntax-Directed Definition that encodes impedance control over trapezoidal velocity profiles.

C. Semantics

Semantic rules are procedures that are executed when the parser expands a production. They represent the continuous domain dynamics as state-space differential equations by computing *attribute* values. While the token type encodes a discrete event, attributes associated with tokens and nonterminals represent the continuous system state. Some attributes are obtained directly from the sensors: positions from encoders, force from load cells, time from the clock. The rest are computed according to the semantic rules. 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. Attributes are thus passed between the tokens and nonterminals in the grammar to implement the continuous domain semantics. Hybrid control is achieved by combining the discrete decisions of the Motion Parser's syntax with continuous functions defined by the semantics.

V. HUMAN-ROBOT GAME APPLICATION

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. 3. 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.

In our implementation, a human plays against the Schunk LWA3 7-DOF robot arm. The robot has a 6-axis force-torque sensor at the wrist that we used for force control. A Mesa Swiss Ranger allowed the robot to locate the Shogi pieces, and a microphone detected sounds indicating turn loss. 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. The lowest levels of our grammatical controller operated at 1kHz.

A. Tokens and Semantics

The tokens and attributes used by the yama grammar encode the hybrid system dynamics. Both are summarized in Fig. 4. The tokenizer η produces each token ζ by thresholding the current sensor reading. The Motion Parser uses tokens to determine syntactically correct expansions of productions according to the grammar.

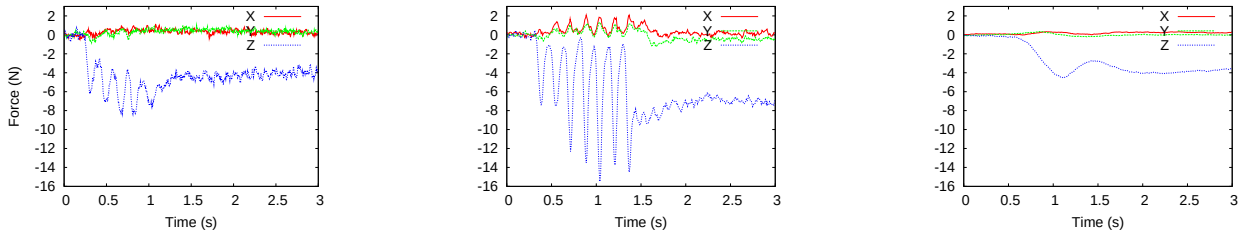
The semantic rules in yama assign updated sensor readings to attributes, maintain previously computed attributes, determine new targets for the controller, and send control input. In this paper, we give one key example of robot control through semantic rules.

The SDD presented in Fig. 5 illustrates the semantics of trapezoidal velocity profiles. Expanding $\langle g' \rangle$ yields distinct semantic rules depending on t, the time attribute of the current nonterminal. For each stage of the trajectory we target distinct reference positions and velocities. The semantic rule for $\langle g \rangle$ defines the control output as a target velocity, u, based on the references provided by expanding $\langle g' \rangle$ and the current force attribute. This demonstrates how the continuous domain control of physical systems can be encoded in the semantics of a discrete grammar.

B. Touching Pieces

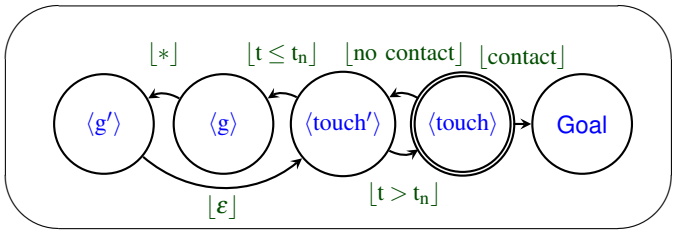
The Finite State Machine in Fig. 6(a) could be used to make the robot touch a Shogi piece. This state machine is equivalent to the grammar in Fig. 6(b). In the grammar, the tokenizing function η applies a threshold to the force-torque sensors and produces $\lfloor$contact$\rfloor$ if the end-effector forces exceed the threshold or $\lfloor$no contact$\rfloor$ otherwise. To expand the $\langle$touch$\rangle$ nonterminal, the parser consumes a $\lfloor$contact$\rfloor$ and returns, or it consumes a $\lfloor$no contact$\rfloor$, moves down a small increment using the trapezoidal velocity profile in $\langle$touch$'\rangle$ and $\langle g \rangle$, and recurses on $\langle$touch$\rangle$. This behavior is mirrored by the state transitions in Fig. 6(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 ap-



(a) Impedance (b) High Gain Impedance (c) Grammatical

Fig. 7. Piece Touching with Impedance and Discrete Control Strategies



(a) Finite State Machine Representation

$$
\begin{aligned}
\langle \text{touch} \rangle &\rightarrow \lfloor \text{contact} \rfloor \\
&\mid \lfloor \text{no contact} \rfloor \langle \text{touch}' \rangle \langle \text{touch} \rangle \\
\langle \text{touch}' \rangle &\rightarrow \lfloor t > t_n \rfloor \mid \lfloor t \leq t_n \rfloor \langle g \rangle \langle \text{touch}' \rangle
\end{aligned}
$$

(b) Equivalent MG Fragment

Fig. 6. Illustration of control for piece touching.

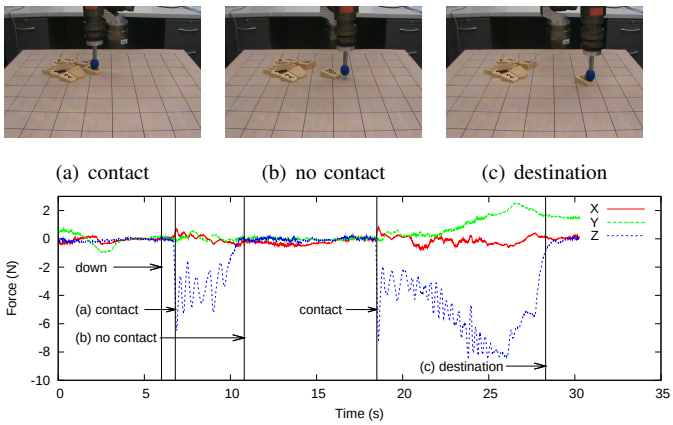
$$
\begin{aligned}
\langle \text{slide} \rangle &\rightarrow \lfloor \text{contact} \rfloor \langle g \rangle \langle \text{slide} \rangle \\
&\mid \lfloor \text{no contact} \rfloor \langle g \rangle \langle \text{slide} \rangle \\
&\mid \lfloor \text{destination} \rfloor \\
&\mid \langle \text{reacquire} \rangle \lfloor \text{contact} \rfloor \langle g \rangle \langle \text{slide} \rangle \\
\langle \text{reacquire} \rangle &\rightarrow \lfloor \text{no contact} \rfloor \langle \text{touch} \rangle
\end{aligned}
$$

Fig. 8. Grammar fragment to reacquire lost pieces.

proach 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 Fig. 7(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 Fig. 7(b) has severe oscillation and very poor performance. The grammatical controller in Fig. 7(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.

C. Sliding and Reacquiring Lost Pieces

The grammar in Fig. 8 describes how the robot slides pieces and how it can reacquire pieces it has lost. This grammar again uses the trapezoidal velocity profile $\langle g \rangle$. The tokenizer η supplies $\lfloor \text{destination} \rfloor$ when robot has moved the piece to the desired location. If the robot momentarily loses contact



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

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

Fig. 9. Application of sliding grammar in Fig. 8 when contact is lost.

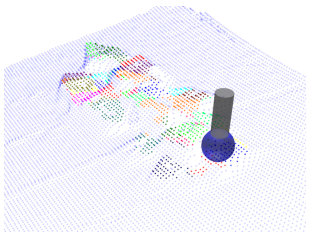
with the piece, it will continue expanding $\langle \text{slide} \rangle$; however, when the contact loss is long enough for the robot to move past the piece, the robot must backtrack to the last contact position to reacquire the piece. This action is performed by $\langle \text{reacquire} \rangle$. 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. 9, it uses the end-effector forces shown in Fig. 9(d) to make its decisions regarding piece contact. At 6s, the robot begins moving down to touch the piece. It acquires the pieces at 6.8s, Fig. 9(a) and begins sliding. At 10.8s, Fig. 9(b), it loses contact with the piece. Recognizing this, the robot backtracks, and again makes contact with the piece at 18.5s. It then continues sliding the piece, reaching the destination at 28.3s, Fig. 9(c).

D. Selecting Target Pieces

The grammar in Fig. V-D describes how the robot chooses a target piece to move. The Swiss Ranger provides a point cloud from the stack of pieces, Fig. 10(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, Fig. 10(b).

(a) Pieces



(b) Planes

$$
\begin{aligned}
\langle \text{act} \rangle &\rightarrow \langle \text{target} \rangle \langle \text{touch} \rangle \langle \text{slide} \rangle \\
\langle \text{target} \rangle &\rightarrow \langle \text{plane} \rangle_0 \mid \ldots \mid \langle \text{plane} \rangle_n \\
\langle \text{plane}_i \rangle &\rightarrow \lfloor \text{point} \rfloor_j \mid \lfloor \text{point} \rfloor_j \langle \text{plane}_i \rangle
\end{aligned}
$$

(c) MG Fragment

Fig. 10. Deciding target piece and direction.

$$
\begin{aligned}
\langle \text{winner} \rangle &\rightarrow \langle \text{draw} \rangle \mid \langle \text{robot} \rangle \mid \langle \text{human} \rangle \\
\langle \text{draw} \rangle &\rightarrow \varepsilon \\
&\mid \lfloor \text{robot piece} \rfloor \langle \text{draw} \rangle \lfloor \text{human piece} \rfloor \langle \text{draw} \rangle \\
&\mid \lfloor \text{human piece} \rfloor \langle \text{draw} \rangle \lfloor \text{robot piece} \rfloor \langle \text{draw} \rangle \\
\langle \text{robot} \rangle &\rightarrow \langle \text{draw} \rangle \lfloor \text{robot piece} \rfloor \langle \text{draw} \rangle \\
&\mid \langle \text{draw} \rangle \lfloor \text{robot piece} \rfloor \langle \text{robot} \rangle \\
\langle \text{human} \rangle &\rightarrow \langle \text{draw} \rangle \lfloor \text{human piece} \rfloor \langle \text{draw} \rangle \\
&\mid \langle \text{draw} \rangle \lfloor \text{human piece} \rfloor \langle \text{human} \rangle
\end{aligned}
$$

Fig. 11. Grammar fragment to decide winner

E. 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. 11. This grammar will count the number of pieces removed by the human $\lfloor$human piece$\rfloor$ and the robot $\lfloor$robot piece$\rfloor$. The $\langle$draw$\rangle$ nonterminal serves to match up a piece removed by the human and a piece removed by the robot. The $\langle$robot$\rangle$ and $\langle$human$\rangle$ 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. 12. This parse tree demonstrates how each $\langle$draw$\rangle$ matches one $\lfloor$r$\rfloor$ and one $\lfloor$h$\rfloor$ token which requires a CFL [20, p125]. This solution to the counting problem for deciding the winner demonstrates the advantage of using a Context-Free model for the MG.

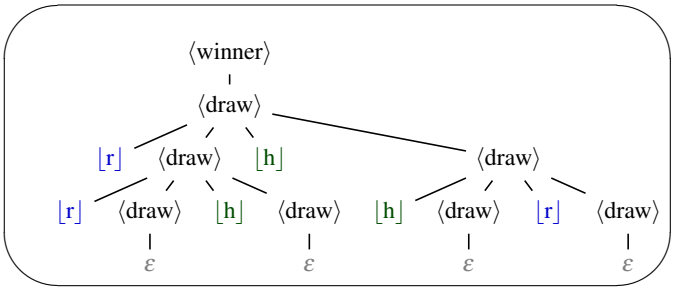


Fig. 12. Parse tree for winner decision problem in draw case: $\lfloor$h$\rfloor \equiv \lfloor$human piece$\rfloor$, $\lfloor$r$\rfloor \equiv \lfloor$robot piece$\rfloor$

$$
\begin{aligned}
\langle \text{game} \rangle &\rightarrow \langle \text{robot turn} \rangle \lfloor \text{clear} \rfloor \langle \text{winner} \rangle \\
&\mid \langle \text{robot turn} \rangle \langle \text{human turn} \rangle \lfloor \text{clear} \rfloor \langle \text{winner} \rangle \\
&\mid \langle \text{robot turn} \rangle \langle \text{human turn} \rangle \langle \text{game} \rangle \\
\langle \text{robot turn} \rangle &\rightarrow \langle \text{act} \rangle \lfloor \text{quiet} \rfloor \langle \text{robot turn} \rangle \\
&\mid \langle \text{act} \rangle \lfloor \text{sound} \rfloor \\
&\mid \lfloor \text{clear} \rfloor \\
\langle \text{human turn} \rangle &\rightarrow \langle \text{wait sound} \rangle \langle \text{wait safe} \rangle \\
\langle \text{wait sound} \rangle &\rightarrow \lfloor \text{sound} \rfloor \\
&\mid \lfloor \text{clear} \rfloor \\
&\mid \lfloor \text{quiet} \rfloor \langle \text{wait sound} \rangle \\
\langle \text{wait safe} \rangle &\rightarrow \lfloor \text{in space} \rfloor \langle \text{wait safe} \rangle \\
&\mid \lfloor \neg \text{in space} \rfloor
\end{aligned}
$$

Fig. 13. Complete Yama Grammar. This is the remaining set of productions used in the game.

F. Complete Game

The remaining productions to implement a full game of Yamakuzushi are given in Fig. 13. A game consists of alternating robot and human turns until the board is clear. During the $\langle$robot turn$\rangle$, it will repeatedly $\langle$act$\rangle$ to remove pieces until it causes $\lfloor$sound$\rfloor$ by making a noise exceeding the preset threshold or until it clears the board. During the $\langle$human turn$\rangle$, the robot will simply wait until it detects a $\lfloor$sound$\rfloor$ or sees that the board has been cleared. After the human makes a $\lfloor$sound$\rfloor$, the robot will wait until the human is out of the workspace before beginning its turn.

VI. Analysis

In this section, we investigate analytical properties of the Motion Grammar for *correctness* and *completeness* of the resulting system. First Sect. VI-A demonstrates grammar completeness ensuring that the robot will respond to all situations. Then Sect. VI-B presents one approach to proving correctness, ensuring that the grammatically-controlled system satisfies a target constraint.

One key element of our analysis is that Motion Grammars use a Context-Free system model. This representation provides an appropriate balance between power of the computational model and provability of the resulting system. Regular languages are a simpler representation whose response can be just as easily proven, but they are very limited in what they can represent. Context-Sensitive Languages are somewhat more powerful than Context-Free, but Context-Sensitive parsing is PSPACE-Complete. Recursively-enumerable languages are more powerful, but by Rice's Theorem, any nontrivial property of a Turing Machine is unprovable [20, p188]. The online parsing of a CFL is polynomial time, and the offline verification is still decidable. Thus, CFLs are an appropriate representation for robotic systems that must operate in real-time and whose behavior should be verified.

A. Completeness

By formalizing the hybrid control problem as a grammar that recognizes the language of the robotic system, we can precisely determine what situations we are able to handle. This allows us to guarantee that the robot will *never give up*. We

define a *complete* MG as one that will direct the robot to take some action in all possible circumstances.

Definition 8: Let $L_{\mathscr{G}_M}$ be the set of all strings accepted by Motion Grammar $\mathscr{G}_M$. Let L_g be the set of all strings produced by the robotic system. Then $\mathsf{complete}\{\mathscr{G}_M\} \iff L_g \subseteq L_{\mathscr{G}_M}$.

Now we give one method for proving completeness of a MG. Though the general subset relation between two CFLs is undecidable [20, p203], we can take advantage of the constraints imposed by continuous-domain dynamics to decide $\mathsf{complete}\{\mathscr{G}_M\}$. This approach combines the First and Follow sets used in compiler theory [1, p220] and the Invariant Set Theorem from Lyapunov theory [32, p68] to show that the system will always remain in a region that is defined by the grammar. We use a variation of the conventional $\mathsf{follow}(A)$, which is based on unions, and instead base our $\mathsf{xfollow}(A)$ on the intersections of tokens that may follow A.

Definition 9: $\mathsf{xfollow}(A)$ is the intersection of all tokens that may follow nonterminal A in the grammar. Specifically, $\mathsf{xfollow}(S) = \$$ where $\$$ indicates the end of the input string and S is not a child in any production. For all $A \neq S$, $\mathsf{xfollow}(A)$ is the intersection of $\mathsf{first}(\beta)$ for all productions $B \to \alpha A \beta$ and of $\mathsf{xfollow}(B)$ for all productions $B \to \alpha A$.

Lemma 10: If a top-down parser receives token ζ after expanding some production $A \to X_1 \ldots X_n$ and $\zeta \in \mathsf{xfollow}(A)$, then ζ will not generate a syntax error.

Proof: Since $\zeta \in \mathsf{xfollow}(A)$, all parent productions of A must be able to accept ζ following their expansion of A. Thus ζ cannot cause a syntax error. ∎

Theorem 11: For Motion Grammar $\mathscr{G}_M$ in Chomsky normal form with productions $p = (A \to \chi) \in P$, let

- S be the starting nonterminal
- $\mathscr{X} = \Re^n$ be the continuous, finite-dimensional, state of the robot and $x \in \mathscr{X}$
- $h_p(x) : \mathscr{X} \mapsto \mathscr{Z}$
- $\mathscr{R}_0(p) = \{x \mid \eta(h_p(x)) \in \mathsf{first}(\chi)\}$
- $\mathscr{R}_1(p) = \{x \mid \eta(h_p(x)) \in \mathsf{xfollow}(A)\}$
- ζ_0 be the first token received by the Motion Parser.

If, $\zeta_0 \in \mathsf{first}(S)$ and for all $p \in P$, $\mathscr{R}_0(p) \subseteq \mathscr{R}_1(p)$ and $\mathscr{R}_0(p)$ or $\mathscr{R}_1(p)$ is an Invariant Set, then $\mathsf{complete}\{\mathscr{G}_M\}$.

Proof: We prove this theorem inductively. For the inductive case, the Motion Parser expanding some $p \in P$ must start with $x \in \mathscr{R}_0(p)$. If $\mathscr{R}_0(p) \subseteq \mathscr{R}_1(p)$ and $\mathscr{R}_0(p)$ or $\mathscr{R}_1(p)$ is an Invariant Set, then at the end of expanding p, the system will have $x \in \mathscr{R}_1(p)$. This will generate a subsequent token $\zeta \in \mathsf{xfollow}(A)$ and Lemma 10 shows that this will not cause a syntax error. For the base case, the Motion Parser is given ζ_0 as the first token. Since ζ_0 is in $\mathsf{first}(S)$, it will not generate a syntax error. ∎

Theorem 11 shows that a system is fully represented by a MG. If our MG is not complete, we can fix this by modifying semantic rules K to create the invariant set property or by creating more productions in P to respond to the additional cases. When the grammar is complete, it guarantees a response to all situations and additionally lets us use the discrete *syntax* to represent the system. We use the syntax to guarantee that the system is correct.

	$L_r \in \mathscr{L}_R$	$L_r \in \mathscr{L}_D$	$L_r \in \mathscr{L}_C$
$L_{\mathscr{G}_M} \in \mathscr{L}_R$	yes	yes	no
$L_{\mathscr{G}_M} \in \mathscr{L}_D$	yes	no	no
$L_{\mathscr{G}_M} \in \mathscr{L}_C$	yes	no	no

Fig. 14. Decidability of $\mathsf{correct}\{\mathscr{G}_M, L_\mathsf{r}\}$ by language class.

B. Correctness

In addition to ensuring that the robot takes *some* action for all circumstances, it is also important to evaluate the correctness of the action. We define the correctness of a language specified by the MG, $L_{\mathscr{G}_M}$, by relating it to a *constraint language*, L_r. While $L_{\mathscr{G}_M}$ for a given problem integrates all problem subtasks, as shown in Sect. V, 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 during execution are also part of language L_r.

Definition 12: A Motion Grammar $\mathscr{G}_M$ is correct with respect to some constraint language L_r when all strings in the language of $\mathscr{G}_M$ are also in L_r: $\mathsf{correct}\{\mathscr{G}_M, L_\mathsf{r}\} \iff L_{\mathscr{G}_M} \subseteq L_r$.

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

Lemma 13: Let $\mathscr{L}_R$ be the Regular set, $\mathscr{L}_D$ be the Deterministic Context-Free set, and $\mathscr{L}_C$ be the Context-Free set. $R \in \mathscr{L}_R$, $D \in \mathscr{L}_D$, and $C \in \mathscr{L}_C$. Then,

1) $C \subseteq C'$ is undecidable. [20, p203]
2) $R \subseteq C$ is undecidable. [20, p203]
3) $C \subseteq R$ is decidable. [20, p204]
4) $R \subseteq D$ is decidable. [20, p246]
5) $D \subseteq D'$ is undecidable. [20, p247]

Corollary 14: Based on $\mathscr{L}_R \subset \mathscr{L}_D \subset \mathscr{L}_C$, the results from [20] extend to the following statements on decidability:

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

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

Theorem 15: The decidability of $\mathsf{correct}\{\mathscr{G}_M, L_\mathsf{r}\}$ for Regular, Deterministic Context-Free, and Context-Free Languages is specified by Fig. 14.

Proof: Each entry in Fig. 14 combines a result from Lemma 13 or Corollary 14 with Definition 12. The algorithms that perform each subset evaluation and therefore the evaluation of $\mathsf{correct}\{\mathscr{G}_M, L_\mathsf{r}\}$ are given in [20]. ∎

Theorem 15 ensures that we can prove the correctness of a MG 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. Furthermore,

since we can determine the complement of any Regular or Deterministic Context-Free language, we can specify constraints in terms of events that should *never* happen, $L_{\mathscr{G}_M} \subseteq \bar{L}_e$. Both positive and negative constraints allow existing algorithms to guarantee that a MG-based system is safe and reliable.

This approach to proving correctness is very similar to the model checking of [3, 14]. We note that while model-checking is a computationally expensive operation, it is one that we can perform *offline*. Since the MG represents a *policy*, once we have verified the grammar, all online operation is performed as efficient parsing. Verifying an efficiently implementable policy permits guarantees while maintaining fast online performance and response to uncertain events.

C. Example Correctness Verification

We present a simple correctness verification to demonstrate the approach.

Claim 16: Let $\mathscr{G}$ be the grammar in Fig. 8, and let L_r be the language represented by regular expression $(.^* \lfloor \text{destination} \rfloor)$. Then $\mathsf{correct}\{\mathscr{G}, \mathsf{L_r}\}$.

Proof: We prove this claim by induction. For the base case, the start symbol of $\mathscr{G}$, $\langle \text{slide} \rangle$, expands immediately to $\lfloor \text{destination} \rfloor$. For the inductive step, $\langle \text{slide} \rangle$ otherwise expands with a rightmost recursive call to $\langle \text{slide} \rangle$. Thus, we will see recursive calls to $\langle \text{slide} \rangle$ until $\lfloor \text{destination} \rfloor$, so $\lfloor \text{destination} \rfloor$ must at some point occur and end the string. ■

VII. Conclusions and Future Work

In this paper we presented a novel approach to perception, planning, and control using grammars. Our Motion Grammar demonstrates that you can have both formal guarantees and computationally efficient performance, crucial properties in the development of robots that are safe and reliable. We showed how the MG can be used to prove a system will be robust to uncertainty, responding to every possible situation, and we described how to develop such a complete grammar. We also showed how to provide guarantees on the correct operation of the system. Finally, we have demonstrated the efficacy of this approach by developing a physical robotic system to play the game Yamakuzushi against a human opponent.

There are many avenues to enhance the guarantees, expressiveness, and power of our MG. Firstly, a parser generator to produce a Motion Parser from the MG would automate much of the implementation details. Applying type theory could provide for stricter definitions and guarantees. We will continue exploring these approaches to improve the capabilities and guarantees of the resulting system.

References

[1] Aho, A., Lam, M., Sethi, R., and Ullman, J. *Compilers: Principles, Techniques, & Tools*, 2nd ed. Pearson, 2007.

[2] Arkin, R. *Behavior-Based Robotics*. MIT press, 1999.

[3] Baier, C., Katoen, J., et al. *Principles of Model Checking*. MIT Press, Cambridge, MA, 2008.

[4] Branicky, M. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on automatic control 43*, 4 (1998), 475–482.

[5] Brockett, R. Formal Languages for Motion Description and Map Making. *Robotics 41* (1990), 181–191.

[6] Brooks, R. A robust layered control system for a mobile robot. *IEEE journal of robotics and automation 2*, 1 (1986), 14–23.

[7] Cassandras, C. *Discrete-Event Systems*, 2nd ed. Springer, 2008.

[8] Craig, J. *Introduction to Robotics: Mechanics and Control*, 3rd ed. Pearson, 2005.

[9] Dantam, N., Kolhe, P., and Stilman, M. The motion grammar for physical human-robot games. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2011), IEEE.

[10] Dantam, N., and Stilman, M. The motion grammar. Tech. Rep. GT-GOLEM-2010-001, Georgia Institute of Technology, 2010. http://smartech.gatech.edu/handle/1853/36485.

[11] Earley, J. An efficient context-free parsing algorithm. *Communications of the ACM 13*, 2 (1970), 94–102.

[12] Egerstedt, M. Motion Description Languages for Multi-Modal Control in Robotics. *Control Problems in Robotics* (2002), 74–90.

[13] Egerstedt, M., Murphey, T., and Ludwig, J. Motion Programs for Puppet Choreography and Control. *Hybrid Systems: Computation and Control* (2007), 190–202.

[14] Esparza, J., Kucera, A., and Schwoon, S. Model checking ltl with regular valuations for pushdown systems* 1. *Information and Computation 186*, 2 (2003), 355–376.

[15] Fainekos, G., Kress-Gazit, H., and Pappas, G. Hybrid controllers for path planning: a temporal logic approach. *Proceedings of the 2005 IEEE Conference on Decision and Control* (2005).

[16] Frazzoli, E., Dahleh, M., and Feron, E. Maneuver-Based Motion Planning for Nonlinear Systems with Symmetries. *IEEE Transactions on Robotics 21*, 6 (2005), 1077–1091.

[17] Fu, K. *Syntactic Pattern Recognition and Applications*. Prentice Hall, 1981.

[18] GCC 3.4 Release, July 2010. http://gcc.gnu.org/gcc-3.4/changes.html.

[19] Han, F., and Zhu, S. Bottom-up/top-down image parsing by attribute graph grammar. *International Conference on Computer Vision 2* (2005).

[20] Hopcroft, J., and Ullman, J. *Introduction to automata theory, languages, and computation*. Addison-wesley Reading, MA, 1979.

[21] Hristu-Varsakelis, D., Egerstedt, M., and Krishnaprasad, P. On the structural complexity of the motion description language MDLe. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on* (2004), vol. 4, IEEE, pp. 3360–3365.

[22] Klavins, E. A language for modeling and programming cooperative control systems. In *IEEE international conference on robotics and automation* (2004), vol. 4, IEEE; 1999, pp. 3403–3410.

[23] Klavins, E., Ghrist, R., and Lipsky, D. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control 51*, 6 (2006), 949–962.

[24] Kloetzer, M., and Belta, C. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics 26*, 1 (2010), 48–61.

[25] Koutsourakis, P., Simon, L., Teboul, O., Tziritas, G., and Paragios, N. Single View Reconstruction Using Shape Grammars for Urban Environments. *ICCV* (2009).

[26] Kress-Gazit, H., Fainekos, G., and Pappas, G. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE transactions on robotics 25*, 6 (2009), 1370–1381.

[27] Littman, M. *Algorithms for sequential decision making*. PhD thesis, Brown University, 1996.

[28] Manikonda, V., Krishnaprasad, P., and Hendler, J. Languages, Behaviors, Hybrid Architectures and Motion Control. *Mathematical Control Theory* (1998), 199–226.

[29] Pack, R. *IMA: The Intelligent Machine Architecture*. PhD thesis, Vanderbilt University, 2003.

[30] Ramadge, P. J., and Wonham, W. M. Supervisory control of a class of discrete event processes. *Analysis and Optimization of Systems 25*, 1 (January 1987), 206–230.

[31] Russell, S., and Norvig, P. *Artificial intelligence: a modern approach*, 2nd ed. Prentice Hall, 2002.

[32] Slotine, J., Li, W., et al. *Applied nonlinear control*. Prentice Hall Englewood Cliffs, NJ, 1991.

[33] Stilman, B. *Linguistic geometry: from search to construction*. Kluwer Academic Publishers, 2000.

[34] Toshev, A., Mordohai, P., and Taskar, B. Detecting and Parsing Architecture at City Scale from Range Data. *International Conference on Computer Vision and Pattern Recognition* (2010).

[35] Ye, H., Michel, A., and Hou, L. Stability theory for hybrid dynamical systems. *Automatic Control, IEEE Transactions on 43*, 4 (2002), 461–474.