Robotics: Science and Systems (RSS), 2013

Correct Software Synthesis for Stable Speed-Controlled Robotic Walking

Neil Dantam*, Ayonga Hereid†, Aaron Ames†, and Mike Stilman*
* Center for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA
† Mechanical Engineering, Texas A&M University, College Station, TX 77843-3123, USA

***Abstract*—We present a software synthesis method for speed-controlled robot walking based on supervisory control of a context-free Motion Grammar. First, we use Human-Inspired control to identify parameters for fixed speed walking and for transitions between fixed speeds, guaranteeing dynamic stability. Next, we build a Motion Grammar representing the discrete-time control for this set of speeds. Then, we synthesize C code from this grammar and generate supervisors[1] online to achieve desired walking speeds, guaranteeing correctness of discrete computation. Finally, we demonstrate this approach on the Aldebaran NAO, showing stable walking transitions with dynamically selected speeds.**

I. Introduction

Walking provides key advantages for robot locomotion. Walkers are holonomic, they move over rough and uneven terrain, and they adjust their center of mass to better manipulate their environment. Producing stable walking on bipedal robots, however, is a challenging control problem. To compose walking with more varied and autonomous robot actions, the robot must safely and reliably transition between different walking behaviors [25, 19]. In this work, we focus on one type of behavior: transitioning between different walking velocities to increase flexibility of walking locomotion. Reliable walking depends on both dynamic stability across a variety of modes and correctness of the discrete software implementation. We present a method to achieve both dynamic stability and discrete correctness in the domain of walking speed regulation by implementing Human-Inspired control of bipedal robots using a supervised Motion Grammar, automatically generating *stable and correct* hybrid control software.

The contribution of this paper is a method for automatic synthesis of software for speed-controlled bipedal robot walking and an experimental realization on the NAO robot. Using Human-Inspired control, we generate parameters for walking at fixed and transition speeds [20] (Sect. IV-A), *guaranteeing dynamic, continuous stability*. Then, we automatically convert this set of steps and transitions to a context-free Motion Grammar [8] (Sect. IV-B), which provides key benefits for control implementation. With a context-free grammar, we can efficiently specify a hierarchical, discrete time controller for step-taking within a single language framework (Fig. 6, Fig. 7). From this single language, we automatically generate the control software for the robot (Sect. V-A) without tedious hand-coding. Then, we apply supervisory control, *guaranteeing discrete correctness* (Sect. V-B). Finally, we demonstrate this approach on the Aldebaran NAO humanoid robot (Sect. VI).

[1]Software for this synthesis and control at http://golems.github.com/motion-grammar-kit

II. Related Work

Bipedal robotic walking is a well studied problem with a variety of control approaches, including Zero Moment Point control [23], passive walking [7], and hybrid zero dynamics [24], to name only a few. In this paper, we use Human-Inspired control [3], which aims for more human-like walking on bipedal robots [20, 4]. *Motion Transitions* in this method [19] model the transition between two different modes, such as flat ground walking and stair climbing. Work in [25] extended the idea of Motion Transitions to rough terrain walking and showed that control parameters for transition steps can be solved in closed form while maintaining hybrid invariance throughout the step. Here, we use Motion Transitions to stably go between a set of fixed-speed walking gaits, implementing the approach through online supervisory control of a Motion Grammar.

Linguistic models for hybrid systems are widely used. Language and Automata Theory [15] was first applied to Discrete Event Systems (DES) by [21]. The Motion Description Language describes a hybrid system switching though a sequence of continuously-valued input functions [5]. Hybrid Automata combine a Finite Automaton (FA) with differential equations for each FA control state. This is a widely studied and utilized model [2, 6, 12, 16, 18]. In this paper, we model hybrid systems using the Motion Grammar which represents continuous dynamics with differential equations and discrete dynamics using a *context-free grammar* (CFG) [8]. In the walking domain, CFGs naturally represent a hierarchical decomposition of step-taking, providing a more compact form than FA. In addition, the grammar notation simplifies some of the symbolic transformations in this paper. More generally, CFGs permit a broader class of discrete dynamics than FA [15]. The challenge posed by CFGs over FA is the need for a more advanced parsing algorithm; we present such an algorithm in Sect. V-A. Finally, CFGs, just like FA, are efficiently parsable and formally verifiable [15, 8]

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 [13]. For robot control, Linear Temporal Logic (LTL) is commonly used to specify behavior [22, 11]. LTL formula are representable as Büchi Automata, which define finite state languages over infinite length strings. We can also apply supervisory control to CFGs [15, 9]. In this work, we use regular expressions to design supervisors for walking. Regular expressions are a convenient notation for specifying the desired sequences of speeds, and they can be efficiently converted to FA [1] implementing the supervisors.

Parser generation is an established technique with many successes. Recursive Descent, LL(1), and LALR parsers are popular for compilers [1]. The Earley [10] and CYK [17, 26] algorithms produce parsers for any CFG. 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 (Sect. V-A) to synthesize robot software from the grammar.

III. Background

A. Formal Language and Motion Grammar

Formal language is a model for both discrete dynamics and computation [21, 15]. A formal language is a set of strings, and a string is a sequence of atomic symbols. For robots, these symbols may represent sensor readings, physical regions, or control laws. In this work, we represent language for robot operation using context-free grammars.

*Definition 1 (*Context-Free Grammar, CFG): *$G = (Z, V, P, S)$ where Z is a finite alphabet of symbols called* terminals *or* tokens*, V is a finite set of symbols called* nonterminals*, P is a finite set of mappings from a nonterminal to a sequence of terminals and nonterminals, $V \mapsto (Z \cup V)^*$, called* productions*, and $S \in V$ is the* start symbol*.*

The productions of a CFG are conventionally written in Backus-Naur form, $A \rightarrow X_1 X_2 \ldots X_n$, 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 clarity, nonterminals may be represented between angle brackets $\langle\rangle$ and terminals between square brackets $[]$.

Context-free languages (CFL) generalize the regular languages while still maintaining useful properties for robot control. Regular languages – often represented with Finite Automata – are limited to finite state. CFLs have infinite state in the form of a pushdown stack, increasing the representative power. This pushdown stack and the grammar notation allow natural and efficient expression of hierarchies of action (Sect. IV-B). CFLs provide these advantages while maintaining the verifiability and efficiency of the regular set [15, 8]. For these reasons, we adopt a context-free model for our system.

To represent the hybrid system dynamics, we use a context-free *Motion Grammar* [8]. The Motion Grammar augments a CFG with continuous system dynamics, represented as *semantic rules* within the grammar. We define the Motion Grammar as,

*Definition 2 (*Motion Grammar): The tuple $\mathcal{G}_M = (Z, V, P, S, \mathcal{X}, \mathcal{Z}, \mathcal{U}, \eta, K)$ where,

- Z — *set of terminals;*
- V — *set of nonterminals;*
- $P \subset V \times (Z \cup V \cup K)^*$ — *set of productions;*
- $S \in V$ — *start symbol;*
- $\mathcal{X} \subseteq \Re^m$ — *cont. state space;*
- $\mathcal{Z} \subseteq \Re^n$ — *cont. observation space;*
- $\mathcal{U} \subseteq \Re^p$ — *cont. 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 3 (*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 software for the robot.*

In this model, discrete control corresponds to *parsing*. The continuous 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 pushdown automaton. 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 states, events, and control modes [8].

We represent specifications for correct operation as regular languages. In the general case, theoretical restrictions on decidablity and computational performance [8] limit specifications to the regular set. To represent regular language specifications, we use *regular expressions*. Regular expressions define a language based on the operators *concatenation* ("ab"), which appends two symbols or subexpressions, *union* ("$a|b$"), which covers all strings defined by each subexpression, and *kleene-closure* or *free-monoid* ("a^*"), which permits zero or more repetitions of the subexpression. A thorough coverage of regular expressions is given in [15].

Supervisory control operates on system G and specification S by restricting G to only those transitions allowed in S. The result G' is,

$$G' = G \cap S \tag{1}$$

The controlled system G' is thus restricted to the only correct transitions as specified by S. This is how we guarantee correctness of the system.

B. Human-Inspired Bipedal Walking

In this section, we briefly introduce the method of Human-Inspired Control which we use to produce stable walking gaits and transitions on the NAO [20, 4].

Robot Model Bipedal walking is well represented as a hybrid system, exhibiting hybrid dynamics within a single step [4]. We model bipedal robots with a Motion Grammar,

$$\mathcal{G}_R = (Z, V, P, S, \mathcal{X}, \mathcal{Z}, \mathcal{U}, \eta, K) \tag{2}$$

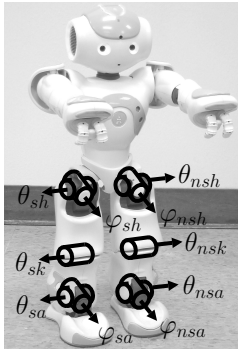


Fig. 1. Angle conventions for NAO.

where $\mathcal{X} = \mathcal{Z}$ is the fully observable *domain* representing the physically allowable state or configuration space of the system $\mathcal{X}, \mathcal{Z} \subset \mathcal{Q}$, $\mathcal{U} \subseteq \Re^{10}$ is the set of *admissible controls*, η defines a set of a *guards* or *switching surfaces* $\mathcal{S}^R \subset Z$ which represent the edge of the domain, and K defines both a smooth map called the *reset map* Δ^R for the guards and the *control system* (f^R, g^R) on $\mathcal{X}$, which is obtained by constructing the Lagrangian mechanical model through the Euler-Lagrange equation [24]. The configuration space, $\mathcal{Q}$, of the 3D version of the conventional seven-link biped model [20] is given by coordinates,

$$q = (\varphi_{sa}, \theta_{sa}, \theta_{sk}, \theta_{sh}, \varphi_{sh}, \varphi_{nsh}, \theta_{nsh}, \theta_{nsk}, \theta_{nsa}, \varphi_{nsa})^T$$

as illustrated in Fig. 1.

Human-Inspired Control The approach of Human-Inspired Control achieves human-like locomotion on bipedal robots. This method identifies a set of outputs from human locomotion data, i.e., distance from hip to ankle. Then, control laws for the robot to track these outputs are produced through optimization.

Previous work shows that the outputs of human locomotion can be expressed as either a linear function of time or a specific form termed the *canonical walking function (CWF)* [3, 4, 20],

$$y_H(t, \alpha_i) = e^{-\alpha_{i,4} t}(\alpha_{i,1} \cos(\alpha_{i,2} t) + \alpha_{i,3} \sin(\alpha_{i,2} t)) + \alpha_{i,5} \tag{3}$$

with parameters $\alpha_i \in \Re^5$, generally represented as row vectors in a parameter matrix $\boldsymbol{\alpha}^*$. The *CWF* thus encodes walking trajectories as a function of time. To achieve human-like walking, the Human-Inspired controller drives the outputs of the robot to the outputs of the human as given by the *CWF*.

Using this function for controller design results in nonautonomous control, i.e, the parameters vary over time. However, autonomous state feedback, i.e., the parameters are time-invariant, is generally more robust to disturbances than nonautonomous control. One common procedure is to parameterize time with a state-dependent map [24]. As shown in [4], the linearized position of the hip, $\delta p^R_{\text{hip}}(q)$, monotonically increases over the course of a step. Thus, define the following parameterization,

$$\tau(q) := (\delta p^R_{\text{hip}}(q) - \delta p^R_{\text{hip}}(q^+))/v_{\text{hip}} \tag{4}$$

where v_{hip} is the desired linearized hip velocity and $\delta p^R_{\text{hip}}(q^+)$ is the linearized position of the hip at the beginning of a step. This parameterization importantly allows for control over walking speed through the parameter v_{hip}. For an arbitrary v_{hip}, one can obtain autonomous controller output functions which depend on parameters $\boldsymbol{\alpha}^*$ from the *CWF* (3). The specific construction of the human outputs and the Human-Inspired Controller can be found in [20]. Notably, for a control gain $\varepsilon > 0$, the controller drives the system to a specific surface which is termed a *zero dynamics surface*.

For the *continuous* dynamics of the hybrid system, the control law renders the *full zero dynamics surface* ($\mathbf{FZ}_\alpha$) exponentially stable. Yet the surface will not be invariant through impact. Therefore, hybrid invariance is enforced only for the relative degree two outputs. The corresponding surface is referred to as the *partial zero dynamics surface* ($\mathbf{PZ}_\alpha$) and is fundamental to Human-Inspired Control. In [3], it was shown that stable walking can be achieved when the system operates on $\mathbf{PZ}_\alpha$.

The next step is to use optimization, imposing the constraints of hybrid zero dynamics, to find the parameters $\boldsymbol{\alpha}^*$. Thus the *human-inspired partial hybrid zero dynamics (PHZD) optimization* problem of [20] was employed to obtain the parameters $\boldsymbol{\alpha}^*$ which minimize the least squares fit of the sagittal plane outputs to the corresponding mean human data. By the nature of the optimization, these parameters will satisfy *PHZD* conditions, resulting in an exponentially stable limit cycle for robot walking at a specific speed v_{hip}. To achieve physically realistic robotic walking, additional physical constraints are enforced by the optimization (see [20]). The optimization also produces a fixed point $(\vartheta(\alpha), \dot{\vartheta}(\alpha)) \in \mathcal{S}^R \cap \mathbf{FZ}_\alpha$ on the *PHZD* surface and the guard, that can be explicitly computed in terms of the parameters $\boldsymbol{\alpha}^*$ (which will later be used to compute transitions between two different *PHZD* surfaces.

An important feature of Human-Inspired Control is that gaits with nearby speeds are similar, thus implying some degree of continuity. By perturbing and fixing v^*_{hip} and then solving an optimization problem by searching in the neighborhood of $\boldsymbol{\alpha}^*$, subject to the same constraints, we can produce control parameters for different walking speeds [20]. Choosing a small perturbation on v^*_{hip} and using $\boldsymbol{\alpha}^*$ as the seed to the *speed regulation optimization* results in rapid convergence. Thus, we begin with a nominal speed $v^0_{\text{hip}} = v^*_{\text{hip}}$, iteratively perturb by $\pm\delta$, and use the optimization result as the seed to find controller parameters for nearby speeds. For our experiments on the NAO (Sect. VI), we obtain a set of *stable* control parameters for 29 different walking speeds from 10cm/s to 38cm/s with a perturbation value of 1cm/s.

IV. Modeling

A. Computing Speed Transitions

To regulate speed of the robot, we identify new control parameters to stably transition between fixed initial and final speeds. This process involves identifying and connecting stable surfaces in the state space of the system. The surfaces for the initial and final speeds are connected by a surface for the speed transition.

We connect these initial and final PHZD surfaces by producing a new zero dynamics surface between them. Each control parameter α^l, obtained from the *speed regulation optimization*, defines a unique PHZD surface $\mathbf{PZ}_{\alpha^l}$ for walking speed v^l_{hip}. To switch between two walking speeds, i.e., to switch between the two different *PHZD* surfaces for these speeds, we identify a new control parameter α^e defining the *intermediate partial zero dynamics surface*, $\mathbf{PZ}^e_\alpha$, which smoothly connects these two *PHZD surfaces* and ensures that partial hybrid zero dynamics is maintained.

To determine the control parameters, α^e, we use the fixed points $((\vartheta(\alpha^l), \dot{\vartheta}(\alpha^l)))$, corresponding to each walking speed. Let α^{l-1} and α^l be the parameters of the CWF associated with walking at two different successive speeds. Associated

with these parameters are the linearized position of the hip, ξ_1, at the beginning and end of a step: $\xi_1^{0,l} = \delta p_{\text{hip}}(\Delta_q \vartheta(\alpha^l))$ and $\xi_1^{f,l} = \delta p_{\text{hip}}(\vartheta(\alpha^l))$. Moreover, since ξ_1 is used to parameterize time (see (4)), we can write the desired output $y_H(\tau, \alpha) = y_H(\xi_1, \alpha)$, which is now viewed as a function of ξ_1. Since the desired outputs of α^e at the beginning of a step associate to α^{l-1} and the end of a step associate with α^l, the goal is to find an α^e, which satisfies the following equations,

$$y_H(\xi_1^{0,l}, \alpha_i^{l-1}) = y_H(\xi_1^{0,l}, \alpha_i^e) \quad (5)$$

$$y_H(\xi_1^{f,l}, \alpha_i^l) = y_H(\xi_1^{f,l}, \alpha_i^e) \quad (6)$$

$$\frac{d}{d\xi_1} y_H(\xi_1, \alpha_i^{l-1})\Big|_{\xi_1=\xi_1^{0,l-1}} = \frac{d}{d\xi_1} y_H(\xi_1, \alpha_i^e)\Big|_{\xi_1=\xi_1^{0,l-1}} \quad (7)$$

$$\frac{d}{d\xi_1} y_H(\xi_1, \alpha_i^l)\Big|_{\xi_1=\xi_1^{f,l}} = \frac{d}{d\xi_1} y_H(\xi_1, \alpha_i^e)\Big|_{\xi_1=\xi_1^{f,l}} \quad (8)$$

for i as each output. However, solving these nonlinear complex equations will be time-consuming and solutions may not exist. Thus we use the *extended canonical walking function (ECWF)* [19] to describe more complex walking motions, such as going up and down stairs. We can use this same form for Motion Transitions [25, 20]. Specifically, the *ECWF* is given by the time solution to a linear mass-spring-damper system subject to sinusoidal excitation,

$$\begin{aligned} y_H^e(t, \alpha_i^e) =& e^{-\alpha_{i,4}^e t}\big(\alpha_{i,1}^e \cos(\alpha_{i,2}^e t) + \alpha_{i,3}^e \sin(\alpha_{i,2}^e t)\big) \\ &+ \alpha_{i,5}^e \cos(\alpha_{i,6}^e t) + \kappa(\alpha) \sin(\alpha_{i,6}^e t) + \alpha_{i,7}^e, \end{aligned} \quad (9)$$

where $\kappa(\alpha_i^e) = (2\alpha_{i,4}^e \alpha_{i,5}^e \alpha_{i,6}^e)/((\alpha_{i,4}^e)^2 + (\alpha_{i,2}^e)^2 - (\alpha_{i,6}^e)^2)$ for each output i defined in [20]. Note that due to the linearity of the parameters $\alpha_{i,1}^e$, $\alpha_{i,3}^e$, $\alpha_{i,5}^e$ and $\alpha_{i,7}^e$ in (9), we can write:

$$y_H^e(t, \alpha_i^e) = Y_H^e(t, \alpha_{i,2}^e, \alpha_{i,4}^e, \alpha_{i,6}^e) \begin{bmatrix} \alpha_{i,1}^e \\ \alpha_{i,3}^e \\ \alpha_{i,5}^e \\ \alpha_{i,7}^e \end{bmatrix} \quad (10)$$

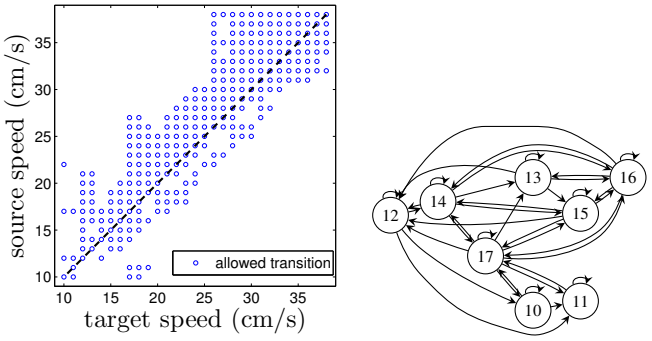
where $Y_H^e(t, \alpha_i^e) \in \Re^{1\times 4}$ only depends on the parameters $\alpha_{i,2}^e$, $\alpha_{i,4}^e$, $\alpha_{i,6}^e$.

Consider the *PHZD* surface for the *ECWF*, denoted by $\mathbf{PZ}_{\alpha^e}$. The advantage to the *ECWF* is that, given any two *PHZD* surfaces these surfaces can be connected via the $\mathbf{PZ}_{\alpha^e}$ to ensure that partial hybrid zero dynamics is maintained, and the corresponding parameters α^e can be computed in closed form; this is not possible with the *CWF* as there are not enough parameters present.

To achieve the goal of determining the parameters α_i^e, we utilize (10) to form the following matrix,

$$\mathbb{Y} = \begin{bmatrix} Y_H^e(\xi_1^{0,l-1}, \alpha_{i,2}^e, \alpha_{i,4}^e, \alpha_{i,6}^e) \\ \frac{d}{d\xi_1} Y_H^e(\xi_1, \alpha_{i,2}^e, \alpha_{i,4}^e, \alpha_{i,6}^e)\Big|_{\xi_1=\xi_1^{0,l-1}} \\ Y_H^e(\xi_1^{f,l}, \alpha_{i,2}^e, \alpha_{i,4}^e, \alpha_{i,6}^e) \\ \frac{d}{d\xi_1} Y_H^e(\xi_1, \alpha_{i,2}^e, \alpha_{i,4}^e, \alpha_{i,6}^e)\Big|_{\xi_1=\xi_1^{f,l}} \end{bmatrix}$$

Picking $\alpha_{i,2}^e = \alpha_{i,2}^l$, $\alpha_{i,4}^e = \alpha_{i,4}^l$, $\alpha_{i,6}^e > 0$ and $v_{\text{hip}}^e = v_{\text{hip}}^l$, yields $\mathbb{Y}$ is nonsingular. Therefore, the final four parameters



(a) full: 10 to 38 cm/s (b) partial: 10 to 17 cm/s

Fig. 3. Graph of permissible speed transitions

of α_i^e can be determined from (5) – (8),

$$\begin{bmatrix} \alpha_{i,1}^e \\ \alpha_{i,3}^e \\ \alpha_{i,5}^e \\ \alpha_{i,7}^e \end{bmatrix} = \mathbb{Y}^{-1} \begin{bmatrix} y_H(\xi_1^{0,l}, \alpha_i^{l-1}) \\ \frac{d}{d\xi_1} y_H(\xi_1, \alpha_i^{l-1})\Big|_{\xi_1=\xi_1^{0,l-1}} \\ y_H(\xi_1^{f,l}, \alpha_i^l) \\ \frac{d}{d\xi_1} y_H(\xi_1, \alpha_i^l)\Big|_{\xi_1=\xi_1^{f,l}} \end{bmatrix} \quad (11)$$

The end result of solving for α^e in this manner is that any solution starting in $\mathbf{PZ}_{\alpha^{l-1}}$ which transitions through $\mathbf{PZ}_{\alpha^e}$ for one step will begin the subsequent step on $\mathbf{PZ}_{\alpha^l}$. In other words, we will connect the *PHZD* surfaces $\mathbf{PZ}_{\alpha^{l-1}}$ and $\mathbf{PZ}_{\alpha^l}$ through $\mathbf{PZ}_{\alpha^e}$, and thus the control laws are valid and stable even as the robot transitions between different speeds.

We test the derived control laws in simulation. The desired and actual value of first five outputs, denoted as $y_{H,i}$ and **Oi** respectively, and the limit cycles of both pitch and roll angles are shown in the Fig. 2. The simulation was started from the speed of 14 cm/s, then sped up to 17 cm/s at the 2nd step, and transitioned back to 14 cm/s at the 4th step. As shown in Fig. 2, the actual outputs, except the first output, which is hip velocity, converge to desired outputs on each step including the two transition steps, which shows that the hybrid invariance was maintained through the transition between two *PHZD* surfaces.

Permissible Speed Transition Graph Given this set of smooth transitions between different walking speeds, we now model this system of fixed speeds and transitions as a directed graph. Mathematically, we can connect any two partial zero dynamics surface through the *ECWF* [19] to enforce hybrid invariance and guarantee dynamic stability. However, not all the control parameters, α^e, solved from Eq. 11 are physically permissible, i.e., non-colliding, walking gaits.

To construct the permissible speed transition graph, physical constraints are checked over the course of the transition step. Fig. 3 shows the permissible speed transition graph for the multiple walking speeds. Fig. 3(a) shows the full graph of walking speeds from 10 to 38 cm/s, where the blue circles indicate the transition from source speed to target speed is physically allowed. Fig. 3(b) figure shows a partial graph for speeds between 10 and 17 cm/s in the conventional directed graph form.

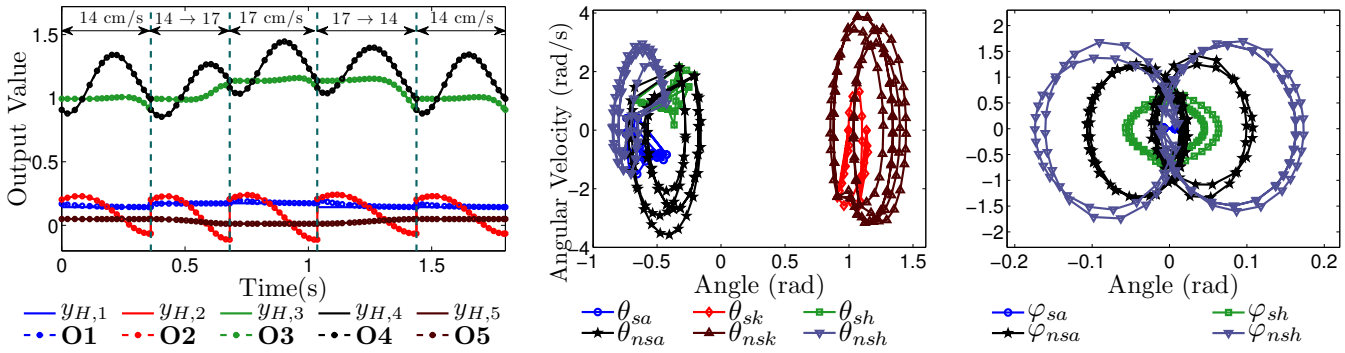


Fig. 2. Simulation results for speed regulation: controller outputs (left), phase portraits for pitch (middle) and roll (right)

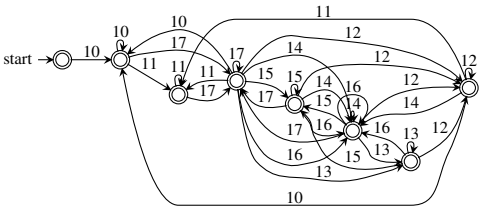


Fig. 4. Minimum state Finite Automaton of speed transitions. Edge labels are speed in cm/s.

B. Grammar for Walking

Given this the graph of permissible speed transitions in Fig. 3, we proceed to construct the Motion Grammar for the system, from which we will synthesize the control software. First, we convert the speed graph (Fig. 3) to a Finite Automaton (Fig. 4). Then we add symbols for transitions steps between different speeds. Finally, we incorporate a grammar for discrete-time control of the individual steps. The result is a grammar 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, i.e., speeds for this walking domain, from the nodes in the graph to the edges in the FA. Fig. 4 shows the FA for transitions between 10 and 17 cm/s. The corresponding FA for the full system with transitions between 10 and 38 cm/s has 26 states, 41 terminals, and 230 edges.

Rewriting the graph as an FA has a few benefits. First, we can apply many existing algorithms for FA such as Hopcroft's Algorithm for state minimization [14]. 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.

Now, we add to Fig. 4 the transition steps to go between different fixed walking speeds. Fig. 5 shows a fragment of the resulting FA. The full transition-step FA for 10-38 cm/s has 58 states, 279 terminals, and 308 edges.

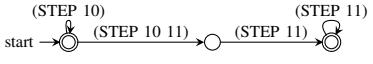


Fig. 5. Finite Automaton fragment showing transition step between 10 and 11 cm/s

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

Fig. 6. Parameter and Step Grammar

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

Fig. 7. Step Grammar. Thresholds on weight $[\text{weight} \geq \varepsilon_\text{w}]$ detect foot placement, and thresholds on cycle count $[\text{count} \geq \varepsilon_\text{c}]$ and time in the current step $[\tau \geq \varepsilon_\tau]$ avoid erroneous transitions due to bounce during placement.

Next, we replace each of the unique $[(\text{step } x)]$ and $[(\text{step } y\ z)]$ 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 in Fig. 6 for speeds of 10 and 11 cm/s. The full grammar for 10-38 cm/s has 340 productions.

Finally, we take each $\langle \text{step} \rangle$ symbol and decompose it with a discrete-time hybrid controller to take a single step. This step controller is shown as the grammar in Fig. 7. The $\{\kappa\}$ production in this grammar computes the inputs for the robot based on the current parameter matrix for the current control cycle. See [20] for the detailed algorithm.

The result of these algorithmic transformations is a grammar with 291 terminals, 68 nonterminals, and 354 productions, which represents all step sequences through speeds of 10-38 cm/s that the robot may take.

V. CONTROL

A. Code Generation

Based on this grammatical model for walking, we synthesize control software as a *Motion Parser* (Def. 3) for this Motion Grammar. For efficiency and portability, we generate standard C code. However, online parsing for real-time control presents a few challenges compared to translating parsers such as compilers:

- Compilers can look forward and backward in the input, while a Motion Parser must provide immediate input to the system without seeing the future.
- For programming languages, parse trees representing the structure of the input have limited depth, while parse trees for a Motion Grammar may be arbitrarily large since the system may need to run for an arbitrarily long time.

To handle these constraints, we place some restrictions on the grammar and perform some optimizations when generating the parser.

While a compiler is typically given its input as a file, a Motion Parser must act token-by-token while continually driving the system. This temporal constraint restricts the ability of the Motion Parser to *lookahead* to future tokens or *backtrack* to a previous point in the parse. First, at each timestep the parser must immediately provide some input to the robot without lookahead to future timesteps, which have not yet occurred. Second, the parser cannot backtrack to previous timesteps, since these have already occurred. We can conservatively satisfy these two parsing requirements with the LL(1) class of grammars [1, 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 [1, p.223] and amply powerful for the walking grammars in this work. Thus, we employ an LL(1) parser generation approach.

The deeply recursive nature of grammatical productions is another issue in online parsing. Notice in these grammars (Fig. 7) that concatenation and looping are implemented recursively, as nonterminals at the end of some parent production. Representing such a parse tree, either explicitly or implicitly via recursive function calls, during a long running operation would consume excessive memory; therefore, we must avoid building such a large structure in memory.

We can avoid this arbitrarily large memory use with *tail call optimization*, as used in functional programming languages like Scheme and ML. Tail call optimization reduces memory use when one function immediately returns the result of another. The optimization reuses the stack frame of the parent function for the tail function, typically replacing a `call` machine instruction with a `jump`. Similarly in our parser generator, whenever some parent production has a nonterminal in the final position of its body, we jump, e.g, `goto` in C, to the code for that nonterminal rather than recursively expanding it. This important optimization limits memory usage for the deeply recursive Motion Parser.

```
int super_mgparse                                          1
( mg_context_t* context,                                   2
  mg_supervisor_table_t *table, int i )                    3
{ //...                                                    4
case 424:                                                  5
nonterm__lp_0_sp__dot__sp_g1_rp_:                          6
  // (STEP TIME-ZERO STEP-1)                               7
  if ( ((mg_supervisor_allow(table, 5)) &&                 8
        (0 == (time_zero(context)))) )                     9
  {                                                        10
    (table->state) =                                       11
      (mg_supervisor_next_state(table, 5));                12
  case 425:                                                13
  prod__lp__lp_0_sp__dot__sp\                              14
  _g1_rp__sp_time_zero_rp_:                                15
    goto nonterm__lp_1_sp__dot__sp_g1_rp_;                 16
  }                                                        17
  return -1;                                               18
  //...                                                    19
}                                                          20
```

Fig. 8. Example of parsing code

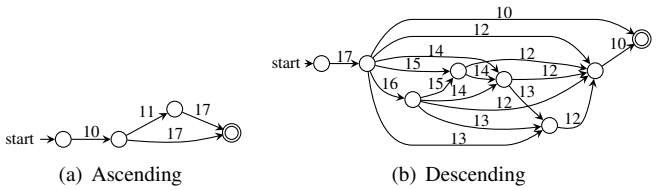
We now implement LL(1) parser generation to construct our Motion Parser. To generate standard C, we need to optimize tail calls to `goto`. Since C `goto` can only target a label in the same function, we must implement our parser as a single function. As a design choice, we use the C call stack for the 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.

Fig. 8 shows a fragment of the generated parser, corresponding to the first production of Fig. 7. This parser first calls `time_zero`, line 9. Notice the `goto` in line 16, implementing the tail recursive expansion. The full generated code for the 354 production grammar amounts to 3488 lines of C code. When compiled with gcc 4.4.5 `-O2`, this produces 8914 instructions in AMD64 assembly.

B. Supervision

We perform speed control on the robot using supervisory control. We algorithmically derive specifications for a supervisor to take the NAO between any two speeds with a minimum number 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. 4 and transform it to an FA with monotonically ascending or descending speeds, Fig. 9. This ensures that the robot will continually increase or decrease in speed. We produce these monotonic FA by repeatedly intersecting the speed



(a) Ascending (b) Descending

Fig. 9. Ascending and Descending Speed Finite Automata

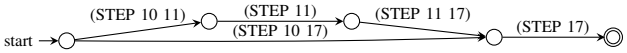


Fig. 10. Ascending FA with transition steps

FA G_s (Fig. 4) with a language S_m to enforce ordering for each terminal symbol. For terminal y, this ordering language S_m is given by the regular expression,

$$S_m = \{x : x < y\}^* \, y \, \{x : x > y\}^* \tag{12}$$

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 would enforce a descending constraint. By applying an S_m for each speed, we produce the monotonic FA in Fig. 9.

Next, we insert the transition steps into these monotonic FA as in Fig. 5. The result for the ascending case is shown in Fig. 10.

Now, we find the desired sequence of steps by generating the shortest string σ in the monotonic transition step FA, Fig. 10. This sequence is obtained using a breadth-first search of the FA states until the accept-state is found. For Fig. 10, this gives the following two steps

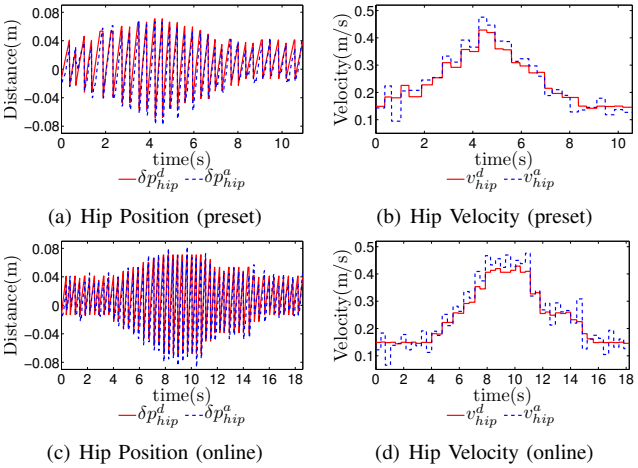
$$\sigma = [\text{step 10 17}] \;\; [\text{step 17}] \tag{13}$$

From the string σ in (13), we generate a regular expression S for the supervisor. Initially, let S be $\langle \text{step} \rangle^*$, which here denotes the union of all terminal symbols in Fig. 7. For each [step a b] in σ, we concatenate to S the expression [setparam a b] $\langle \text{step} \rangle^*$. For the last symbol in σ, indicating the target speed, concatenate $([\text{setparam a}] \;\; \langle \text{step} \rangle)^*$ [HALT]. The result for (13) is the following regular expression, shown as an FA in Fig. 11.

$$S = \langle \text{step} \rangle^* \, [\text{setparam 10 17}] \, \langle \text{step} \rangle^* \, [\text{setparam 17}] \\ ([\text{setparam 17}] \, \langle \text{step} \rangle^*)^* \, [\text{HALT}] \tag{14}$$

Given the supervisor in Fig. 11, 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 $\mathcal{G}$ and the supervisory Finite Automaton S in parallel, transitioning only when both allow it [15, p.135]. Before the parser checks any terminal symbol or executes any semantic rule, it first ensures that the action is allowed from

Fig. 11. Supervisor for transitioning from 10 to 17 cm/s. Here, $\langle \text{step} \rangle$ corresponds to the union of all terminals in Fig. 7



(a) Hip Position (preset) (b) Hip Velocity (preset)

(c) Hip Position (online) (d) Hip Velocity (online)

Fig. 12. Hip position and velocity for preset and online supervisors

the current state of supervisor S. After reading the terminal or executing the semantic rule, the parser updates the state of S for the transition on that symbol. The result implements the supervised system, $\mathcal{G} \cap S$.

VI. Experimental Results

Our experimental platform is the Aldebaran NAO, Fig. 1. The NAO is a 0.5m, 5kg bipedal robot with 25 degrees-of-freedom (DOF). We focus on controlling the NAO's legs, each of which has five DOF and Force Sensitive Resistors (FSR) on the bottom of the feet to detect the reactive force with the ground. We use this platform to validate our approach for speed-controlled walking.

We conduct two different experiments to test the effectiveness of the proposed supervisory control approach for speed control: one with a predetermined supervisor, in which the target speeds were set prior to the experiment, and one with the supervisor generated online, which allows the target speeds to be chosen and set during walking.

In the first experiment, the robot starts from an initial speed of 14cm/s, speeds up to 38cm/s, slows to 32cm/s, 28cm/s, 25cm/s and 20cm/s sequentially, then finally returns to the initial speed of 14cm/s. The whole experiment takes 30 steps to achieve these three changes in walking speeds, including several steps of steady state walking at 14cm/s at the end. The result are plotted in Fig. 12. Fig. 12(a) figure shows the actual linearized hip position $\delta p_{\text{hip}}(q)$, which is monotonically increasing over each step (see Fig. III-B) and closely tracks the desired references throughout all steps. This plot shows relative hip position, reset after each step, to reduce total plot size. Fig. 12(b) shows the changes in both desired hip velocity v^d_{hip} and actual mean hip velocity v^a_{hip}. These velocities are computed as the slope of a linear fit of hip position $\delta p_{\text{hip}}(q)$ over one step. We note that the robot achieved stable walking in many trials with multiple walking speeds and transition between them in a few steps without falling. Compared to the result of [20], the robot achieves greater speed in fewer steps using a supervised Motion Parser.

The second experiment examines online supervisory control. The NAO walks 50 steps for each trial of this experiment,

and an operator can choose and set the target walking speed during the experiment. The software will then generate a new supervisor to take the NAO from its present speed to the speed selected by the operator. For these trials, the robot starts from same $14\mathrm{cm/s}$ initial speed as before. After a few steps, we set the target speed to $38\mathrm{cm/s}$, then again to $25\mathrm{cm/s}$, and finally back to the initial speed of $14\mathrm{cm/s}$ to perform a full speed-up and slow-down cycle. The mean hip velocity and the linearized hip position for this experiment are shown in Fig. 12. *These results show that by using online supervisory control with the Motion Grammar, we can dynamically select and change the speed of the robot while it is walking, all with model-based guarantees on stability.*

We note two performance issues in the experiments. First, the NAO slightly changes direction during walking due to a small, unmodeled rolling on the lateral edge of the robot's foot. Extending the walking model to include the heel roll should reduce these direction changes. Second, there is an initial step down of hip velocity caused by foot scuffing before the robot reaches steady-state walking. Since we model from non-zero initial velocity rather than from rest, the robot should take a few steps to adjust its gait before reaching stable walking. A transition step from rest to initial walking speed should improve stability of walking, which we will address in future work. These issues illustrate the caveat of model-based guarantees: the guarantee is only as good as the model.

VII. Conclusion

In this paper, we presented an approach to develop software for speed-controlled robot walking with model-based guarantees on *stability and correctness*, and we demonstrated this method on the Aldebaran NAO. Human-Inspired walking produces *stable* controllers for fixed speeds and transitions between speeds. Framing these transitions as a formal language with the Motion Grammar enables automatic synthesis of *correct* control software through parser generation and supervisory control. Our experimental results on the NAO show that this technique allows supervisors to be updated online with dynamically selected walking speeds and achieves greater speed in fewer steps than previous implementations. With this linguistic hybrid method for speed control, we have guaranteed both continuous domain stability through stable walking and discrete-domain software correctness through proper transitioning of dynamically selected behaviors.

References

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

[2] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, pages 209–229, 1993.

[3] A. D. Ames. First steps toward automatically generating bipedal robotic walking from human data. In *Robotic Motion and Control 2011*, volume 422 of *LNICS*, pages 89–116. Springer, 2012.

[4] A. D. Ames, E. A. Cousineau, and M. J. Powell. Dynamically stable bipedal robotic walking with NAO via human-inspired hybrid zero dynamics. In *Hybrid Systems: Computation and Control*, pages 135–44, Beijing, April 2012.

[5] RW Brockett. Formal languages for motion description and map making. *Robotics*, 41:181–191, 1990.

[6] C.G. Cassandras and Stéphane Lafortune. *Introduction to Discrete-Event Systems*. Springer, 2nd edition, 2008.

[7] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307: 1082–5, February 2005.

[8] N. Dantam and M. Stilman. The motion grammar: Analysis of a linguistic method for robot control (accepted). *Trans. on Robotics*, 2012.

[9] N. Dantam, C. Nieto-Granda, H. Christensen, and M. Stilman. Linguistic composition of semantic mapping and hybrid control. In *ISER*, 2012.

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

[11] Georgios Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using s-taliro. In *Proceedings of the American Control Conference*, 2012.

[12] T.A. Henzinger. The theory of hybrid automata. In *Symposium on Logic in Computer Science*, pages 278–292. IEEE, 1996.

[13] G.J. Holtzman. *The Spin Model Checker*. Addison Wesley, Boston, MA, 2004.

[14] J. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. *Reproduction*, pages 189–196, 1971.

[15] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, MA, 1979.

[16] D. Hristu-Varsakelis and W.S. Levine, editors. *Handbook of Networked and Embedded Control Systems*. Birkhauser, 2005. ISBN 0817632395.

[17] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

[18] J. Lygeros, K.H. Johansson, S.N. Simic, J. Zhang, and S.S. Sastry. Dynamical properties of hybrid automata. *IEEE Trans. on Automatic Control*, 48(1):2–17, 2003.

[19] M. J. Powell, H. Zhao, and A. .D. Ames. Motion primitives for human-inspired bipedal robotic locomotion: Walking and stair climbing. In *IEEE Intl. Conf. Robotics and Automation*, pages 543–549, St.Paul, May 2012.

[20] M. J. Powell, A. Hereid, and A. .D. Ames. 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.

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

[22] S. Sarid, B. Xu, and H. Kress-Gazit. Guaranteeing high-level behaviors while exploring partially known maps. In *RSS*. IEEE, 2012.

[23] M. Vukobratović and B. Borovac. Zero-moment point—thirty-five years of its life. *Intl. J. of Humanoid Robotics*, 1(1):157–73, March 2005.

[24] E. R. Westervelt, J. W. Grizzle, C. Chevallereau, J. H. Choi, and B. Morris. *Feedback Control of Dynamic Bipedal Robot Locomotion*. CRC Press, Boca Raton, 2007.

[25] S. Nadubettu Yadukumar and A. D. Ames. Achieving bipedal locomotion on rough terrain through human-inspired control. In *IEEE Intl. Symposium on Safety, Security, and Rescue Robotics*, College Station, November 2012.

[26] D.H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189–208, 1967.