Proceedings of the ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference
IDETC/CIE 2017
August 6-9, 2017, Cleveland, Ohio, USA

DETC2017-68049

CORRECT-BY-CONSTRUCTION APPROACH FOR SELF-EVOLVABLE ROBOTS

Gang Chen
Department of Mechanical and Aerospace Engineering
University of California, Davis
One Shields Avenue,Davis, CA
ggchen@ucdavis.edu

Zhaodan Kong
Department of Mechanical and Aerospace Engineering
University of California, Davis
One Shields Avenue,Davis, CA
zdkong@ucdavis.edu

ABSTRACT

The paper presents a new formal way of modeling and designing reconfigurable robots, in which case the robots are allowed to reconfigure not only structurally but also functionally. We call such kind of robots “self-evolvable”, which have the potential to be more flexible to be used in a wider range of tasks, in a wider range of environments, and with a wider range of users. To accommodate such a concept, i.e., allowing a self-evovable robot to be configured and reconfigured, we present a series of formal constructs, e.g., structural reconfigurable grammar and functional reconfigurable grammar. Furthermore, we present a correct-by-construction strategy, which, given the description of a workspace, the formula specifying a task, and a set of available modules, is capable of constructing during the design phase a robot that is guaranteed to perform the task satisfactorily. We use a planar multi-link manipulator as an example throughout the paper to demonstrate the proposed modeling and designing procedures.

1 Introduction

Reconfigurable robots are a family of robots that are capable of adjusting their shapes and functions to changing environments and tasks [1, 2]. They are posed to meet the increasing demands of providing personal robots to adjust to individual needs and physical characteristics [3, 4] as well as industrial robots to adapt to changes in the market [5]. Over the past three decades, the field of reconfigurable robots has advanced from proofs-of-concept to physical implementations. However, even with their potential versatility and robustness over conventional robots, reconfigurable robots still suffers from inferior performance, one of the main factors impeding them from practical adoption. Furthermore, existing reconfigurable robots are rarely capable of functional adaption. In this paper, we propose a formal modeling framework of reconfigurable robots that are capable of both structural and functional reconfigurations. We will also explore a design philosophy called “correct-by-construction” to guarantee the performance of the robots during the design phase.

Formally the approaches of studying reconfigurable robots can be roughly divided into three categories, those based on graph theory, those based on optimization, and those based on dynamic analysis. Graph-theory-based approaches are mostly suitable to study how modules are put together structurally [1, 6–8]. Modules are represented as vertices while connections between the modules are represented as edges. Then tools from graph theory can be used to solve problems related to reconfigurable robots, such as configuration recognition [9] and motion planning [8]. Optimization-based approaches cast the design of a reconfigurable robot as an optimization problem with a objective function over the vector of design variables [10, 11]. The design variables, either discrete or continuous, are subjected to equality and/or inequality constraints. The optimization-based approaches are suitable to address trade-offs among multiple competing objectives. The detailed kinematics/dynamics of the designed robots are generally either ignored or simplified in the first two types of approaches, while the last type of approaches, dynamic-analysis-based, puts kinematics/dynamics as the main focus [12, 13]. Currently papers employing dynamic-analysis-based approaches mostly deal with arm robots [12,14] with some exceptions dealing with mobile robots [13]. One issue with the aforementioned approaches is their inability to allow for simultaneously structural and functional reconfigurations, thus greatly restrict the potential of their robots. In this paper, we will develop a formal framework incorporating both types of reconfigurations. We will call such type of robots as self-evolvable robots. Notice that in existing literature, reconfiguration generally refers to structural changes, i.e., units/modules change the way they connect to each other mechanically. In this paper, we will adopt a rather broader definition of reconfiguration to include functional changes within each unit (we will focus on using changes in dynamics due to some physical parameters, e.g., the length of a link, as an example of functional changes in this paper). This is inspired by natural evolution, i.e., a biological mechanism (analogous to a robot) gradually changes its shapes and functions to adapt to changes in the environment (analogous to changes in missions). For the rest of the paper, we will use self-evolvable robots and reconfigurable robots interchangeably.

This paper is organized along the line of modeling and design as follows. Section 2 discusses the modeling of self-evolvable robots. Section 3 formally defines the design problem. Section 4 presents the method to solve the design problem. Section 5 provides a case study to demonstrate our method. Section 6 concludes the paper.

2 Modeling of Self-Evolvable Reconfigurable Robots

In this section, we will first describe a list of modules that will be used in this paper to construct self-evolvable robots. The list is not meant to be exhaustive but mainly serves as a running example for the rest of the paper. Next, we will introduce two definitions, structural reconfiguration grammar (SRG) and structural reconfiguration automaton (SRA), which formally characterize the way the modules are mechanically/structurally connected to each other to form a robot. Then we will introduce dynamic models of modules. Finally, we will introduce two additional definitions, functional reconfiguration grammar (FRG) and functional reconfiguration automaton (FRA), which formally characterize the way to (re)configure a robot not only structurally but also functionally.

2.1 Modules

Reconfigurable robots have the capacity to deliberately change their own structures by adaptively rearranging the connectivity of their components according to the environments and/or task scenarios [1]. The repeatable building components of a reconfigurable robot are called modules or mechanical units. They usually have uniform docking interfaces allowing different modules to connect to each other mechanically and electronically.

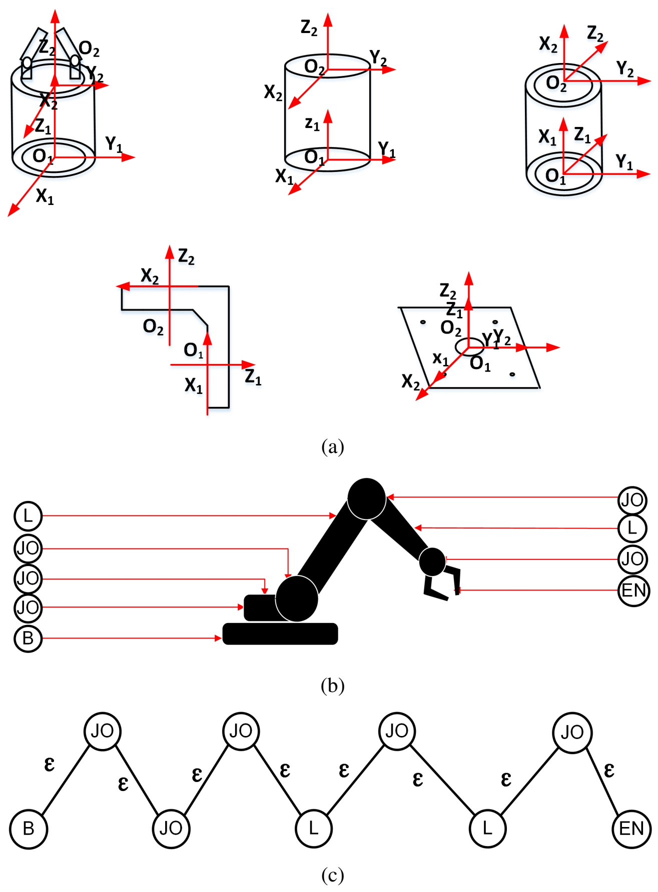


FIGURE 1. (a) The modules that will be used in this paper to construct reconfigurable robots. They are (starting from the upper left corner in the clockwise direction) end-effector module, joint module, cylindrical link module, base module, and L-shaped link module. (b) An illustrative example of a robot built from the modules. (c) The (Σ,Γ) labeled graph representation of the robot shown in (b). The sequence of symbols representing the configuration is $B\varepsilon JO\varepsilon JO\varepsilon JO\varepsilon L\varepsilon JO\varepsilon L\varepsilon JO\varepsilon EN$.

In the following text, we will describe four types of modules as shown in Fig. 1(a). Each module has an input end denoted by subscript 1 and an output end denoted by subscript 2. Information comes into the module via the input end and gets out of the module via the output end. Two coordinate frames are attached to the two ends of the module for the purpose of characterizing the dynamics of various parts of the robot. To illustrate the functional reconfigurability, some module is associated with a design parameter, which can be adjusted thus changing the functionality of the module.

Joint Module: As shown by the upper middle sub-figure of Fig. 1(a), the joint module F is modeled as a cylinder with an axis of rotation O_1O_2. An input frame JO_1 is attached to the input connector/end at point O_1 and an output frame JO_2 is attached to the output connector/end at point O_2. The z-axes of the two frames both coincide with the line O_1O_2 while their x and y axes define the two end planes.

Link Module: In this paper, we define two different types of link modules, the cylindrical link module as shown at the upper right corner of Fig. 1(a) and the L-shaped link module at the lower left corner of Fig. 1(a). The cylindrical link module is modeled similar to the link module. It is a cylinder with an axis of rotation O_1O_2. An input frame L_1 and an output frame L_2 are attached to the two ends at point O_1 and point O_2, respectively, with their Z-axes having the same direction as O_1O_2 and their X- and Y-axes defining the two end planes. The L-shaped link, on the other hand, has its input end and out end perpendicular to each other. An input frame L_1 is attached to the input end of the module with its z-axis perpendicular to the input end plane and its X- and Y-axes defining the input end plane. An output frame L_2 is defined similarly with respect to the output end. Each link module has a designing parameter p_L, which is the length of the module.

End-Effector Module: The end-effector module is the functional component of the robot. It has a variety of forms, i.e., a mechanical gripper and a machine tool base. In this paper, as shown at the upper left corner of Fig. 1(a), we will use a gripper as an example of the end-effector module. An input frame EN_1 is attached to its input end with its Z-axis perpendicular to the end plane and its X- and Y-axes defining the end plane. An output frame EN_2 is defined in such a way that its origin is at the grasping center of the fingers.

Base Module: The base module serves as the base for other modules. As shown at the lower right corner of Fig. 1(a), an input frame B_1 is attached to the input end of the module, which is attached to the ground, while an output frame B_2 is attached to the output end of the module in such a way that the origin of the frame B_2 is located at the center of the base, its Z-axis is perpendicular to the output end plane, and its X- and Y-axes define the output end plane.

A module can be connected to another one as long as the input framework of one module coincide with the output framework of the other. Of course, in order to build a functional robot, some further requirements need to be taken into consideration, e.g., the base module must be attached to the ground and there must be at least one end-effector module. An example of such a robot built from the modules is illustrated in Fig. 1(b).

2.2 Structural (Re)Configuration

In this subsection, we will present two definitions, structural reconfiguration grammar (SRG) and structural reconfiguration automaton (SRA). Both of them can characterize the way to reconfigure a robot structurally/mechanically.

2.2.1 Structural Reconfiguration Grammar (SRG)

Let's first define (Σ,Γ) labeled graph, modified from a definition called Σ-labeled Γ-graph in [15].

Definition 1. ((Σ,Γ) labeled graph) [15]: Let Σ and Γ be two finite nonempty sets of node labels and edge labels, respectively. Let $G=(V,E)$ be a directed graph, where V is the set of nodes and E is the set of directed edges. The graph G can be labeled by a function $l: E \to (\Sigma,\Gamma)$ with the node labeling $l_V: V \to \Sigma$ and the edge labeling $l_E: E \to \Gamma$. The tuple $\langle G,\Sigma,\Gamma\rangle$ is called a (Σ,Γ) labeled graph (or simply labeled graph) and denoted by $G_{(\Sigma,\Gamma)}$.

Next, we will define the structural reconfiguration grammar (SRG).

Definition 2. *(Structural Reconfiguration Grammar, SRG)*: A reconfiguration graph grammar SRG is a tuple $SRG = (\Sigma,\Gamma,N,P,I)$, where Σ is a finite alphabet of node symbols or tokens, Γ is a finite alphabet of edge symbols or tokens, N is a finite set of symbols called non-terminals, P is a finite set of mappings $N \to (\Sigma\cup\Gamma\cup N)^*$ called production rules with superscript $(\cdot)^*$ as a notation for the set of all strings over an alphabet $(\cdot)$, and $I \in \Sigma$ is the initial node symbol.

The production rules can be conveniently written in Backus-Naur form [16], $N \to X_1X_2...X_n$, where N is some non-terminal and $X_1X_2...X_n$ is a sequence of node/edge symbols and non-terminals. A production rule indicates that N may expand to all strings represented by the right hand side of the rule. The collection of all sequences of *terminal* symbols/tokens, i.e., those in Σ or Γ, generated by the SRG is called the language of the SRG, denoted by $L(SRG) \subset (\Sigma\cup\Gamma)^*$.

Example 1. *(SRG for a reconfigurable, planar, multi-link manipulator robot)* Given the four types of modules described in Sec. 2.1, JO, the joint module, L, the link module, EN, the end-effector module, and B, the base module, a SRG for a reconfigurable, planar, multi-link robot is $SRG = (\Sigma,\Gamma,N,P,I)$ with (1) $\Sigma = \{JO,L,EN,B\}$, the collection of modules; (2) $\Gamma = \{\varepsilon\}$ meaning that there is no restriction on the way one module is connected to another one; (3) N is the collection of (Σ,Γ) labeled graphs with each element corresponding to a structural configuration of the robot; (4) $P: N \to B|N\varepsilon JO|N\varepsilon L|N\varepsilon EN$ characterizing how the robot is configured; and (5) $I = B$. An illustration is shown in Fig. 1(c).

The structural reconfiguration grammar (SRG) can be defined alternatively as follows:

Definition 3. *(Structural Reconfiguration Grammar, SRG, Alternative Definition)*: A reconfiguration graph grammar SRG is a tuple $SRG = (Z,N,P,I)$, where $Z = \{\Sigma,\Gamma\}$ is a finite alphabet of symbols or tokens, P is a finite set of mappings $N \to (Z\cup N)^*$

called production rules, and the others have the same meanings as in Definition 2.

2.2.2 Structural Reconfiguration Automaton (SRA)

Context-free grammars (CFGs), such as those in Definition 2 and Definition 3, have equivalent representations as pushdown automata (PDA) which recognize the language of the grammar [16]. A pushdown automaton is a automaton with a stack, which provides the automaton with memory. The automaton corresponding to SRG, called structural reconfiguration automata (SRA), can be defined as follows:

Definition 4. *(Structural Reconfiguration Automaton, SRA)*: A reconfiguration graph automaton SRA is a tuple $SRA = (Q,Z,\delta,Q_0,A)$, where Q is a finite set of states, Z is a finite alphabet of symbols/tokens, $\delta : Q \times Z \to Q$ is the transition function, Q_0 is the initial state, $A \in Q$ is the set of accept states.

Let $SRA = (Q,Z,\delta,Q_0,A)$ be a SRA and $\omega = z_1 \ldots z_n \in Z^*$ a finite word. A run for ω in SRA is a finite sequence of states $q_0 q_1 \ldots q_n$ such that: (i) $q_0 \in Q_0$; (ii) $q_i \to_{z_{i+1}} q_{i+1}$ for all $0 \le i < n$ where $\to_\cdot$ is defined by the transition δ as $q \to_z q'$ if and only if $q' \in \delta(q,z)$. Runs $q_0 q_1 \ldots q_n$ is called accepting if $q_n \in A$. A finite word $\omega \in Z^*$ is called accepted by SRA is there exists an accepting run for ω. The accepted language of SRA, denoted by $L(SRA)$, is the set of finite words in Z^* accepted by SRA, i.e., $L(SRA) = \{\omega \in Z^* |$ there exists an accepting run forω in $SRA\}$.

The idea behind the construction of a PDA from a CFG is to have the PDA simulate the sequence of left- or right-sentential forms that the grammar uses to generate a given terminal string ω [16–18]. For a reconfiguration graph grammar SRG, there is a unique equivalent reconfiguration graph automaton SRA such that $L(SRG) = L(SRA)$. For a given SRG $SRG = (Z,N,P,I)$, its equivalent SRA $SRA = (Q,Z,\delta,Q_0,A)$ is constructed as follows: (1) $Q = (N \cup Z)^*$; (2) they share the same Z; (3) $q' \in \delta(q,z)$ if $q \to q' = qz$ is a production rule with $q \in N$, $z \in Z$, and $q' \in (N \cup Z)^*$; (4) $Q_0 = I$; and (5) $A = Z^*$. Even though SRA and SRG are equivalent, they can be used for different purposes. For instance, SRG, given its constructive form, is more intuitive, while SRA, given its automaton format, is easier to be integrated with other formal verification and synthesis techniques, such as model checking [17].

Example 2. *(SRA of the reconfigurable robot in Example 1)* Part of the SRA of the reconfigurable robot illustrated in Example 1 and Fig. 1(a) is shown in Fig. 2. A state of the SRA corresponds to a labeled graph representation of a structural configuration with only one initial state $q_0 = B$, i.e., the the base module. A transition between two states represents the addition or removal of a module. For instance, the transition from q_0 to q_1 represents to connection of a joint module JO to the base module B. To specify the requirement that a functional robot must have an end-effector, we restrict the set of accept states to $A = \{\omega \in Z^* | \exists i$, such that $\omega_i = EN\}$, i.e., at least one of the modules need to be an end-effector module. An example of such accept states is q_f in Fig. 2. The steps to construct a robot structurally can then be represented as an accepted run of the corresponding SRA as shown in the figure.

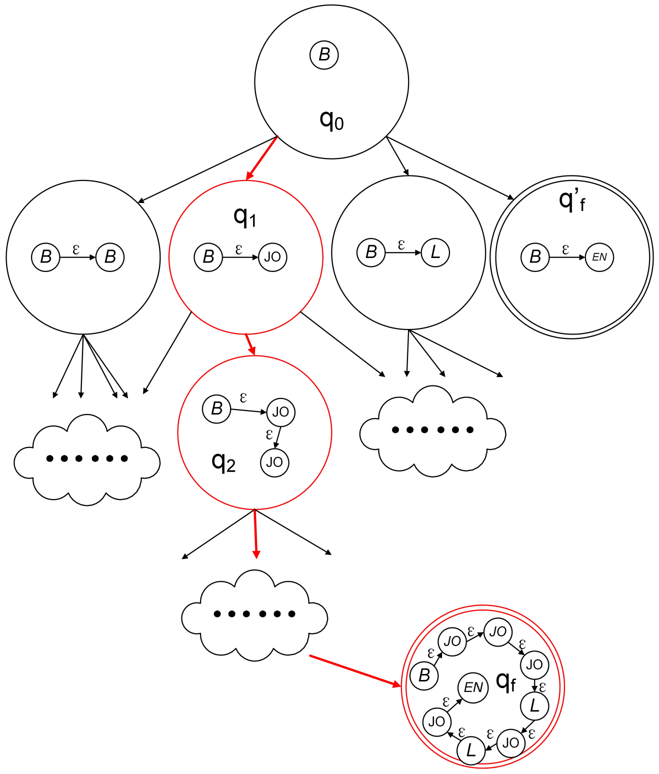


FIGURE 2. Part of the reconfiguration graph automaton (SRA) of the reconfigurable robot illustrated in Example 1 and Fig. 1(c). The initial state q_0 is indicated by having an incoming arrow without source. The accept states are indicated by double circles. The red nodes and edges indicate the accepted run to construct the robot in Fig. 1(c).

2.3 Models of Modules

Each module of a reconfigurable robot has a unique function, which depends on some continuous or discrete design parameters, e.g., the dimension of the module. For instance, the length of a link module of a robot determines the configuration space of the robot, i.e., whether a certain position and orientation can be achieved. Here we introduce a definition of modules as one of the modeling bases to allow a robot to reconfigure not only its structure but only its functions, i.e., the set of design parameters.

Definition 5. *(Model of Modules)*: The function of a modules $\sigma \in \Sigma$ is defined as a parametric controlled dynamical system

$F_\sigma = (X_\sigma, \Xi_\sigma, U_\sigma, f_\sigma)$, where $X_\sigma \subset \mathbb{R}^n$ is the state space, $\Xi_\sigma \subset \mathbb{R}^p$ is the parameter space, $U_\sigma \subset \mathbb{R}^m$ is the control space, $f_\sigma : X_\sigma \times \Xi_\sigma \times U_\sigma \to X_\sigma$ is an analytic vector field, assumed to be sufficiently smooth, and n, p, m are the dimensions of X_σ, Ξ_σ and U_σ, respectively.

Notice that the modules are controlled not autonomous, meaning that a designer or the robot itself has the freedom to specify a control policy $u_\sigma \in U_\sigma$ for a module σ.

2.4 Functional (Re)Configuration

The models of modules, combined with the concept of SRG, lead naturally to the following concept called Functional Reconfiguration Grammar (FRG):

Definition 6. *(Functional Reconfiguration Grammar, FRG)*: A functional reconfiguration grammar FRG is a tuple $FRG = (N, Z, P, F, I)$, where N, Z, P and I are defined the same as in Definition 3 and $F := \{F_\sigma, \sigma \in \Sigma\}$ with each F_σ defined the same as in Definition 5.

Given a set of modules with their dynamics described by parametric dynamical systems (Definition 5) and as a grammar, structural reconfiguration grammar (SRG) in our case, describing how these modules can be structurally connected, the above definition gives rise to a range of dynamics that can emerge from the whole robots. Such dynamics can be considered as the results of the semantic interpretation of the syntax of the functional reconfiguration grammar (FRG), i.e., given a production rule in the corresponding SRG, a semantic rule can be generated by a parser; applying a sequence of production rules in SRG gives the structural configuration of the robot, while applying the corresponding sequence of semantic rules of the corresponding FRG gives the functional configuration (the dynamics in our case) of the robot. The aforementioned points can be best understood with an example.

Example 3. *(FRG of the reconfigurable robot in Example 2)* The linearized dynamic model of a reconfigurable robot constructed in Example 2 can be described as follows:

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{T\} - \{F\} \tag{1}$$

where

$$\begin{aligned}
&\{x\} = \{\triangle q\}^T; \quad [M] = [E_2]^T [E_2] \\
&[C] = [E_2]^T [E_1] + [\dot{E}_2]^T [E_2] + [E_2]^T [\dot{E}_2] - [E_1]^T [E_2] \\
&[K] = [\dot{E}_2]^T [E_1] + [E_2]^T [\dot{E}_1] - [E_1]^T [E_1] \\
&\{T\} = \{\triangle \tau\}^T \\
&\{F\} = [\dot{E}_2]^T [E_0] + [E_2]^T [\dot{E}_0] - [E_1]^T [E_0]
\end{aligned}$$

with $\triangle q$ as the vector of perturbed link poses and $\triangle \tau$ as the vector of perturbed torques [19].

Here a set of semantic rules can be introduced to construct the matrix E for the set of production rules in structural reconfiguration grammar (SRG). Specifically, if the production rule is $N \to N\varepsilon JO$, i.e., the newly added module is a joint, then E will be kept the same; if the production rule is $N \to N\varepsilon L$, i.e., the newly added module is a link; let's call the new link as the n-th link and index existing ones as 1-st link, 2-nd link and so on, according to the order they are added, then E will be updated as follows:

$$\begin{aligned}
[E_{i0}] &= \begin{Bmatrix} \sum_{l=1}^{i} -L_l \dot{q}_{0l} \sin q_{0l} \\ \sum_{l=1}^{i} L_l \dot{q}_{0l} \cos q_{0l} \end{Bmatrix} \\
[E_{i1}] &= \begin{bmatrix} -L_1 \dot{q}_{01} \cos q_{01} & -L_2 \dot{q}_{02} \cos q_{02} & \cdots & -L_i \dot{q}_{0i} \cos q_{0i} \\ -L_1 \dot{q}_{01} \sin q_{01} & -L_2 \dot{q}_{02} \sin q_{02} & \cdots & -L_i \dot{q}_{0i} \sin q_{0i} \end{bmatrix} \\
[E_{i2}] &= \begin{bmatrix} -L_1 \sin q_{01} & L_2 \cos q_{02} & \cdots & -L_i \sin q_{0i} \\ L_1 \cos q_{01} & L_2 \cos q_{02} & \cdots & L_i \cos q_{0i} \end{bmatrix}
\end{aligned}$$

$$\left([E_k]^T [E_l]\right)_n = \left([E_k]^T [E_l]\right)_{n-1} + L_n [E_{nk}]^T [E_{nl}]$$

where L_i is the length of the i-th link with $i = 1, ..., n$ and $k, l = 0, 1, 2$, $k \geq l$.

An automaton, called functional reconfiguration automaton (FRA), that is equivalent to a functional reconfiguration grammar (FRG), can be constructed similar to the way that a structural reconfiguration automaton (SRA) is constructed from a structural reconfiguration grammar (SRG). We are going to omit the definition here to save space.

3 Design Problem Statement

The robot's workspace can be represented by a set of polytopes $P = \{P_i, i = 1, ..., p\}$. Each polytope P_i is assigned with an atomic proposition $\pi_i \in \Pi = \{\pi_t, \pi_o, \pi_f\}$, where π_t, π_o and π_f stand for "target region", "obstacle region" and "free region", respectively. The adjacency relationship among the polytopes can be encoded by an adjacency matrix $N = [N_{i,j}, i, j = 1, ..., p]$ with $N_{i,j}$ as one if polytope i and polytope j are neighboring regions, zero otherwise. Finally, there is a projection function $\mathscr{H} : X \to \Pi$ which maps a robot's state to its corresponding atomic proposition.

Problem 1. Given a functional reconfiguration grammar $FRG = (N, Z, P, F, I)$ and a workspace description $\mathscr{W} = (P, \mathscr{H}, N)$, find a finite sequence of symbols $\omega = z_1 ... z_n \in Z^*$ and a finite sequence of parameters $\theta_\omega = \theta_{z_1} ... \theta_{z_n}$ with $\theta z_i \in \Theta_{z_i}$ such that: (i) ω is accepted by the corresponding *SRG*; (ii) there exists a trajectory $x_0, ..., x_k$ of the robot built in accordance with ω and θ_ω, satisfying the following formula ϕ:

$$\mathscr{H}(x_k) = \pi_t \wedge_{i=0}^{k-1} \mathscr{H}(x_i) = \pi_f \wedge_{i=0}^{k-1} N(\mathscr{H}(x_i), \mathscr{H}(x_{i+1})) = 1. \tag{2}$$

Remark 1. The above formula essentially specifies a motion planning (pick-and-place) problem, i.e., the constructed robot should be able to move from a starting region to the target region while in the meantime avoiding all obstacles. Such a way of specifying the problem may seem awkward. There are two reasons for such a choice: one is to enable us to use off-the-shelf solvers to find a feasible path, and the other one is to keep the option open for future extensions. For instance, we are interested in using richer logic specifications, such as linear temporal logic [17] and signal temporal logic [20, 21], in the future.

Remark 2. Solving the problem requires solving the following two sub-problems. The first one is a structural synthesis problem. We need find an ω such that it is accepted by the corresponding *SRG*, meaning that we need to build a robot that is structurally feasible, e.g., it must start from a base and end up with an effector. The solution of this sub-problem is a robot with its structural configuration fixed, i.e., the set of selected modules and the way they are connected to each other are determined. The second sub-problem is a functional synthesis problems, involving selecting a parameter θ_{z_i} for each module F_{z_i} in such a way that a feasible trajectory can be generated by the constructed robot. Notice that since each module is modeled as a parametric controlled dynamical system, even after the parameters have been chosen for all the modules, we still need to to check whether there exists a control policy to solve the problem. The first sub-problem is an easy one, given the formulation of the definition of SRG or SRA. So next we are going to focus on solving the second sub-problem.

4 Functional Synthesis

Before embarking upon presenting the solution to the functional synthesis problem, let's first introduce a concept called configuration robustness.

4.1 Configuration Robustness

Once the structural (encoded by ω, see Problem 1) and functional (encoded by θ_ω, see Problem 1) configuration of a robot has been determined, the dynamics of the robot will be determined as well, as demonstrated by the Example 3. The equation describing such dynamics, e.g., Eqn. (1), can be written in its state space form as follows:

$$x_{i+1} = A(x_i)x_i + B(x_i)u_i. \tag{3}$$

Definition 7. *(Configuration Robustness)*: Given a configuration (ω, θ_ω), a workspace description $\mathscr{W} = (P, \mathscr{H}, N)$, a finite trajectory of the corresponding robot $\bar{x} = x_0, \ldots, x_k$, and a formula ϕ, e.g., Eqn. (2), the configuration robustness ρ is defined as follows:

$$\rho(\omega, \theta_\omega, \mathscr{W}, \bar{x}, \phi) = \max_{\substack{u_0, \cdots, u_{k-1} \in \mathbb{R}^m \\ v_0, \cdots, v_{k-1} \in \mathbb{R}^m}} \left(-\max_{\substack{s^u_0, \cdots, s^u_{k-1} \in \mathbb{R} \\ s^v_0, \cdots, s^v_{k-1} \in \mathbb{R}}} (s^u_0 + s^v_0, s^u_1 + s^v_1, \cdots, s^u_{k-1} + s^v_{k-1})\right)$$

subject to

$$\begin{array}{ll}
(C.1) & \mathscr{H}(\bar{x}) \models \phi; \\
(C.2) & x_{i+1} = A(x_i)x_i + B(x_i)u_i + B^{'} v_i, \quad i = 0, \cdots, k-1; \\
(C.3) & \| u_i \| \leq \bar{u} + s^u_i, \quad i = 0, \cdots, k; \\
(C.4) & \| v_i \| \leq s^v_i, \quad i = 0, \cdots, k; \\
(C.5) & 0 \leq s^v_i, \quad i = 0, \cdots, k; \\
(C.6) & \varepsilon \left(\sum_{l=0}^{i-1} s^u_l + s^v_l \right) \leq s^u_i + s^v_i, \quad i = 1, \cdots, k.
\end{array}$$

C.1 says that the trajectory $\bar{x}$ must satisfy the specification ϕ. *C.2* says that $\bar{x}$ is a feasible trajectory of the robot. $B^{'}$ is a matrix to make $[B(x_i), B^{'}]$ surjective. v_i is an additional control input. *C.3*, *C.4*, and *C.5* constrain the input u_i and the additional input v_i by slack variables s^u and s^v. These slack variables are added to relax the dynamics constraints. $\bar{u}$ is a bound on the magnitude of the control input. Finally, *CR.6* provides a user specified bound ε on the slack variables.

Theorem 1. *Given two structural configurations ω_1 and ω_2 with $\omega_1 = \omega_2 z$, i.e., ω_2 is a prefix of ω_1, then the following relationship holds:*

$$\rho(\omega_1, \theta^*_{\omega_1}, \mathscr{W}, \bar{x}, \phi) \geq \rho(\omega_2, \theta^*_{\omega_2}, \mathscr{W}, \bar{x}, \phi)$$

where $\theta^*_{\omega_1}$ and $\theta^*_{\omega_2}$ are the optimal parameters for the two structural configurations ω_1 and ω_2, respectively, in term of configuration robustness.

Proof. The proof of this theorem can be found in the Arxiv version of the paper [22].

Theorem 2. *Given a functional reconfiguration grammar $FRG = (N, Z, P, F, I)$ and a workspace description $\mathscr{W} = (P, \mathscr{H}, N)$, there is a solution to Problem (1) if and only if there exists an ω, a θ_ω, and a trajectory $\bar{x}$ generated by the robot built in accordance with ω and θ_ω, such that*

$$\rho(\omega, \theta_\omega, \mathscr{W}, \bar{x}, \phi) \geq 0$$

Proof. The proof of this theorem can be found in the Arxiv version of the paper [22].

Algorithm 1: Correct-by-Construction for Self-Evolvable Reconfigurable Robots

```
Input:
Workspace description $\mathscr{W} = (P, \mathscr{H}, N)$, a functional
reconfiguration grammar $FRG$, an initial configuration $q_0$
of the FRG
 1: $\omega = q_0$
 2: $\mathscr{W}^* = WS.Abstraction(\mathscr{W})$;
 3: while No feasible trajectory has been found do
 4:     $\kappa = SAT(P^*, \mathscr{H}^*, N^*)$ ($P.Planning$);
 5:     $\theta^*_\omega = argmax[\rho(\omega, \theta_\omega, \mathscr{W}^*, \bar{x}, \kappa)]$ ($P.Synthesis$);
 6:     if $\rho(\omega, \theta^*_\omega, \mathscr{W}, \bar{x}, \kappa) < 0$ then
 7:     $\phi_c = CounterExample$;
 8:     $\kappa = \kappa \wedge \phi_c$;
 9:         if $\kappa := \emptyset$ then
10:             $\omega = S.Synthesis(\omega)$;
11:     else
12:             break;
13: return Configuration $\omega, \theta_\omega$.
```

Our proposed algorithm to solve Problem 1 is briefly outlined in Algorithm 1. It involves solving three main subproblems. The first problem is a path planning problem (*P.Planning*): given certain abstraction of the workspace, it finds a path $\kappa = (\kappa_0, ..., \kappa_k)$ satisfying

$$\kappa_0 = \bar{\kappa} \wedge \kappa_k = \pi_t \wedge_{i=1}^{k-1} \kappa_i = \pi_f \tag{4}$$

where $\bar{\kappa}$ is the starting region, π_t is the target region, and π_f is a free region. Essentially the problem entails finding a path from the starting region to the target region while avoiding all obstacles. The second problem is a parameter synthesis problem (*P.Synthesis*): given a path κ and the current structural configuration ω of the robot, it finds an optimal parameter θ_ω to optimize the configuration robustness (refer to Theorem 2 for the rationale). Finally, we need to solve a structural synthesis problem (*S.Synthesis*), i.e., to find the next structural configuration to be considered. The last problem is an easy one as mentioned. So we are going to focus on solving the other two problems.

4.2 Path Planning

Before solving the path planning problem, we first abstract the description of the workspace $\mathscr{W} = (P, \mathscr{H}, N)$ further by using the technique described in [23]. The corresponding function in Algorithm (1) is *WS.Abstraction*. The end result is a coarse abstraction of the workspace. Each region $\mathscr{W}_i$ is a polytope, mathematically specified by a set of linear constraints $C_{\mathscr{W}_i} x \leq b_{\mathscr{W}_i}$. Notice that such an abstraction is a refinement of the original description of the workspace. Thus the original proposition of a region should be inherited, i.e., if $\mathscr{W}_j$ is a refinement of P_i, then $\pi_{\mathscr{W}_j} = \pi_{P_i}$.

Notice that propositions attached to the regions are atomic. Furthermore, the path planning specification, Eqn. (4), is written in propositional logic. Thus given the abstraction of the workspace, $\mathscr{W}^*$, off-the-shelf SAT solvers can be used to efficiently solve the path planning problem [24]. The corresponding function in Algorithm 1 is SAT. In the future, we are planning to replace the specification with richer ones, such as those written in linear temporal logic [17]. In that case, SMT solvers are needed [25].

4.3 Parameter Synthesis

According to Theorem 2, the parameter synthesis problem (*P.Synthesis* in Algorithm 1) entails to an optimization problem, i.e., given the current structural configuration ω of the robot, a workspace description $\mathscr{W} = (P, \mathscr{H}, N)$ (after the abstraction), and a path $\kappa = (\kappa_0, ..., \kappa_k)$ computed by the path planning algorithm (*P.Planning*), find a parameter θ_ω as well as its corresponding control policy u_0^{k-1} (subjected to the constraint that $|u_i| \leq \bar{u}, i = 0, ..., k-1$) such that the configuration robustness $\rho(\omega, \theta_\omega, \mathscr{W}^*, \bar{x}, \kappa)$ is maximized.

Notice that once a parameter has been selected, the configuration and subsequently the dynamics of the robot will be determined. Provided with different parameters, the corresponding control policies, if they exist, will be different. Thus the parameter synthesis requires solving two problems iteratively:

(i) Given a parameter θ, a workspace description $\mathscr{W} = (P, \mathscr{H}, N)$, and a path $\kappa = (\kappa_0, ..., \kappa_k)$, find whether there exists a control policy u_0^{k-1} for the robot to track the path without colliding with the obstacles.
(ii) Find the next parameter to optimize $\rho(\omega, \theta_\omega, \mathscr{W}^*, \bar{x}, \kappa)$.

For the first problem, since, in this case, the structural configuration ω is fixed and the functional configuration (described by the parameter θ_ω) is fixed as well. According to the semantics of functional reconfiguration grammar (FRG), the two configurations together will give rise to the dynamics of the robot, which can be linearized (see Eqn. (3) for a simple example). Moreover, the workspace is described by a set of linear inequalities, $C_{\mathscr{W}_i} x \leq b_{\mathscr{W}_i}$. In summary, the first problem is linear and can be efficiently solved by linear programming algorithms.

The second problem is more challenging and interesting. There is no closed form solution for $\rho(\omega, \theta_\omega, \mathscr{W}^*, \bar{x}, \kappa)$ even for simple configurations. The only way any information can be obtained regarding a particular parameter θ_ω is to first of all solve the control synthesis problem (the aforementioned problem (i)) and then find out its corresponding robustness. Essentially we are trying to solve a global optimization problem with an unknown objective function ρ. Such kind of problems can be solved by using particle swarm optimization [26], Nelder-Mead [27], simulated annealing [28], and stochastic gradient descent algorithm [20], etc. But it is worth pointing out that many of these techniques may suffer from slow convergence.

To facilitate the convergence rate of the optimization, we use an active learning algorithm called Gaussian Process Adaptive Confidence Bound (GP-ACB) developed by our group in [29]:

$$\theta_t = argmax_{\theta\in\Theta} m_{t-1}(\theta) + \eta_m(\theta)^{\frac{1}{2}}\beta_t^{\frac{1}{2}}\sigma_{t-1}(\theta), \qquad (5)$$

where t is the current step; Θ is the search space; β_t is a function of t and independent of θ; $m_{t-1}(.)$ and $\sigma_{t-1}(.)$ are the mean and covariance functions of a Gaussian process, which is unknown and characterize the underlying configuration robustness function ρ, respectively; θ_t is the instance that will be inquired at step t, meaning the label of θ_t will be obtained from the oracle (in our case, the first problem, i.e., the control synthesis problem, will be solved and the corresponding configuration robustness will be returned); $\eta_m(\theta)$ normalizes the mean $m_{t-1}(\theta)$ and can be written explicitly as

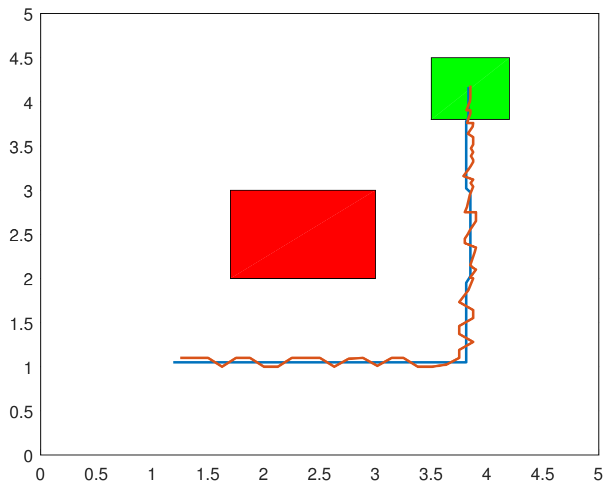
$$\eta_m(\theta) = \frac{m_{t-1}(\theta) - \min(m_{t-1}(\theta))}{\max(m_{t-1}(\theta)) - \min(m_{t-1}(\theta))}.$$

In the algorithm, $\eta_m(\theta)$ acts as an adaptive factor to uncertainty (covariance) and favors exploration directions associated with increasing rewards. We have shown in [29] theoretically that GP-ACB outperforms many state-of-the-art active learning algorithms with similar settings, e.g., GP-UCB. We have also shown empirically that GP-ACB outperforms many state-of-the-art sampling based optimization algorithms, e.g. Nelder-Mead, by an average of 30 to 40 percent faster.

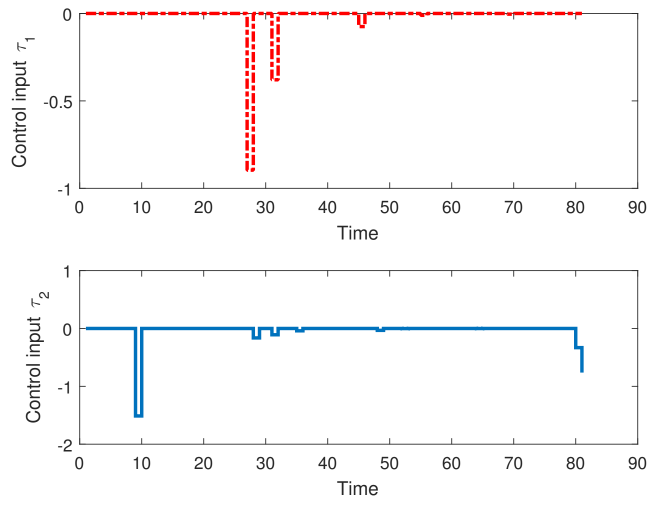
If the optimal configuration robustness for the current structural configuration is negative, meaning that there is no feasible solution regardless of the functional configuration and the control policy, we will relax the current configuration by adding another module (remember that we have shown in Theorem 1 that doing this will always improve the optimal configuration robustness). Moreover, the found counter-example ϕ_c will be added to the current path specification κ to prone the search space for the path planning algorithm.

5 Case Study

The following case study is based on the functional reconfiguration grammar FRG constructed in the first three examples and a workspace as shown in Fig. 3(a). In the workspace, there is an obstacle region shown in red and a target region shown in green.



(a) Workspace



(b) Control policy

FIGURE 3. (a) The workspace. The obstacle region is shown in red while the target region is shown in green. The blue line is the path κ obtained by using the SAT solver while the red line is the actual robot trajectory $\bar{x}$. (b) The associated control policy for two links, the first one of length L_1 2.23 and the second one of length L_2 3.35.

The two regions can be mathematically described as follows:

$$\begin{bmatrix}-1 & 0\\ 1 & 0\\ 0 & -1\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_o\\ y_o\end{bmatrix} \leq \begin{bmatrix}-1.7\\ 3\\ -2\\ 3\end{bmatrix}; \quad \begin{bmatrix}-1 & 0\\ 1 & 0\\ 0 & -1\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_t\\ y_t\end{bmatrix} \leq \begin{bmatrix}-3.5\\ 4.2\\ -3.8\\ 4.5\end{bmatrix}$$

We further set $\bar{u}$, the bound on the magnitude of the control input, to 10.

Essentially, we are given a set of modules and a workspace; we need to construct a robot by (i) selecting and connecting modules (structural configuration); and (ii) selecting appropriate parameters (in our case, the lengths of a link) for each module (functional configuration), such that the constructed robot is able to steer from the initial region to the target region while avoiding the obstacle.

Once the workspace has been abstracted (*WS.Abstraction* in Algorithm 1), the SAT-solver (*P.Planning* in Algorithm 1) is applied to find a path κ as shown by the blue line in Fig. 3(a). The path κ consists of a sequence 81 rectangular regions, i.e., $k = 81$. Associated with each region is a set of linear inequalities.

It is quite obvious that there is no way for a robot with only one link to move from the starting region to the target region without hitting the obstacle. We are able to confirm this observation by using our algorithm. Basically, the linear program algorithm and the active learning algorithm, GP-ACB, are combined to solve the parameter synthesis problem (*C.Synthesis* in Algorithm 1) and we are unable to get a solution, i.e., a parameter resulting positive configuration robustness.

Thus, the structural configuration of the robot is relaxed. According to the production rules, the structural reconfiguration automaton (SRA) will transit to the next state, which corresponds a robot with two links. According the semantics of the corresponding functional reconfiguration grammar (FRG), the dynamics of the two-linked manipulator is as follows:

$$\frac{d}{dt}\begin{bmatrix}\triangle\,\theta_1\\ \triangle\,\theta_2\\ \triangle\,\dot{\theta}_1\\ \triangle\,\dot{\theta}_2\end{bmatrix} = \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ \multicolumn{2}{c}{\text{-E/D}} & 0 & 0\\ & & 0 & 0\end{bmatrix}\begin{bmatrix}\triangle\,\theta_1\\ \triangle\,\theta_2\\ \triangle\,\dot{\theta}_1\\ \triangle\,\dot{\theta}_2\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ \multicolumn{2}{c}{\text{1/D}}\\ & \end{bmatrix}\begin{bmatrix}\triangle\,\tau_1\\ \triangle\,\tau_2\end{bmatrix} \tag{6}$$

with

$$D = \begin{bmatrix}\frac{l_1^3+l_2^3+3l_1^2l_2}{3} + l_2^2 l_1 cos(\theta_2) & \frac{l_1^3}{3} + \frac{l_2^2 l_1}{2} cos(\theta_2)\\ \frac{l_1^3}{3} + \frac{l_2^2 l_1}{2} cos(\theta_2) & \frac{l_2^3}{3}\end{bmatrix}$$

$$E = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}.$$

Then a parameter synthesis problem (*C.Synthesis* in Algorithm 1) is solved and we are able to find a solution as shown by the red trajectory as shown in Fig. 3(b). The corresponding control policy is shown in Fig. 3(b). The parameters are found by using the GP-ACB algorithm to optimize the configuration robustness over the two parameters, l_1 and l_2, the lengths of the two links (their relationship is shown in Fig. 4.). They are $\theta_1^* = L_1^* = 2.23$ and $\theta_2^* = L_2^* = 3.35$, respectively.

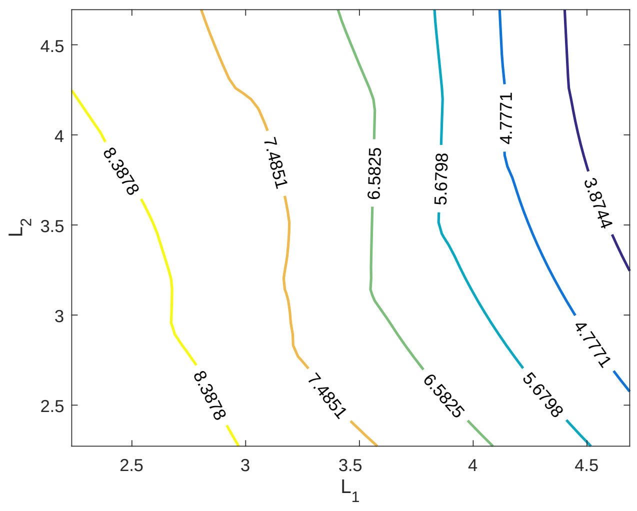


FIGURE 4. The relationship between the configuration robustness and the two parameters L_1 and L_2 for a robot with two links.

6 Conclusion

The paper presents a new way of modeling and designing reconfigurable robots. We propose a series of concepts, including structural reconfigurable grammar, structural reconfigurable automaton, and functional reconfigurable grammar, to formally characterize how a reconfigurable robot can be configured and re-configured not only structurally but also functionally. Furthermore, we propose a correct-by-construction design strategy of utilizing such models. We demonstrate with a planar multi-link manipulator and a pick-and-place task as an example to show how such a strategy works.

REFERENCES

[1] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[2] H. Ahmadzadeh, E. Masehian, and M. Asadpour, "Modular robotic systems: Characteristics and applications," *Journal of Intelligent & Robotic Systems*, vol. 81, no. 3-4, pp. 317–357, 2016.

[3] K. Harada, E. Susilo, A. Menciassi, and P. Dario, "Wireless reconfigurable modules for robotic endoluminal surgery," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 2699–2704.

[4] A. C. Satici, A. Erdogan, and V. Patoglu, "Design of a reconfigurable ankle rehabilitation robot and its use for the estimation of the ankle impedance," in *2009 IEEE Inter-*

national Conference on Rehabilitation Robotics. IEEE, 2009, pp. 257–264.

[5] A. M. Farid and L. Ribeiro, "An axiomatic design of a multi-agent reconfigurable manufacturing system architecture," in *International Conference on Axiomatic Design*, 2014, pp. 1–8.

[6] N. Eckenstein and M. Yim, "Modular reconfigurable robotic systems: Lattice automata," in *Robots and Lattice Automata*. Springer, 2015, pp. 47–75.

[7] J. Neubert and H. Lipson, "Soldercubes: a self-soldering self-reconfiguring modular robot system," *Autonomous Robots*, vol. 40, no. 1, pp. 139–158, 2016.

[8] C. Sung, J. Bern, J. Romanishin, and D. Rus, "Reconfiguration planning for pivoting cube modular robots," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1933–1940.

[9] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 403–421, 2008.

[10] G. Freitas, G. Gleizer, F. Lizarralde, L. Hsu, and N. R. S. dos Reis, "Kinematic reconfigurability control for an environmental mobile robot operating in the amazon rain forest," *Journal of Field Robotics*, vol. 27, no. 2, pp. 197–216, 2010.

[11] S. Ferguson, A. Siddiqi, K. Lewis, and O. L. de Weck, "Flexible and reconfigurable systems: Nomenclature and review," in *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2007, pp. 249–263.

[12] A. L. Balmaceda-Santamaría, E. Castillo-Castaneda, and J. Gallardo-Alvarado, "A novel reconfiguration strategy of a delta-type parallel manipulator," *International Journal of Advanced Robotic Systems*, vol. 13, 2016.

[13] L. Cucu, M. Rubenstein, and R. Nagpal, "Towards self-assembled structures with mobile climbing robots," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1955–1961.

[14] A. M. Djuric, R. Al Saidi, and W. ElMaraghy, "Global kinematic model generation for n-dof reconfigurable machinery structure," in *2010 IEEE International Conference on Automation Science and Engineering*. IEEE, 2010, pp. 804–809.

[15] F. Reiter, "Distributed graph automata," in *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2015, pp. 192–201.

[16] A. Aho and J. D. Ullman, "Introduction to automata theory, languages and computation," 1979.

[17] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of model checking*. MIT press, 2008.

[18] N. Dantam and M. Stilman, "The motion grammar: Analysis of a linguistic method for robot control," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 704–718, 2013.

[19] W. Chen, "Dynamic modeling of multi-link flexible robotic manipulators," *Computers & Structures*, vol. 79, no. 2, pp. 183–195, 2001.

[20] Z. Kong, A. Jones, and C. Belta, "Temporal logics for learning and detection of anomalous behavior," *IEEE Transactions on Automatic Control*, in press.

[21] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016, pp. 6565–6570.

[22] G. Chen and Z. Kong, "Correct-by-construction for self-evolvable robots," *Arxiv*.

[23] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta, "Automatic deployment of robotic teams," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 75–86, 2011.

[24] Y. Shoukry, P. Nuzzo, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Scalable motion planning using lazy smt-based solving," *pdfs.semanticscholar.org*, 2016.

[25] A. Komuravelli and et.al, "Smt-based model checking for recursive programs," in *International Conference on Computer Aided Verification*. Springer, 2014, pp. 17–34.

[26] I. Haghighi, A. Jones, Z. Kong, E. Bartocci, R. Gros, and C. Belta, "Spatel: a novel spatial-temporal logic and its applications to networked systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 189–198.

[27] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*. ACM, 2013, pp. 43–52.

[28] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," in *Proceedings of the 17th international conference on Hybrid systems: computation and control*. ACM, 2014, pp. 273–282.

[29] G. Chen, Z. Sabato, and Z. Kong, "Active learning based requirement mining for cyber-physical systems," in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016, pp. 4586–4593.