Integrated and Adaptive Locomotion and Manipulation for Self-Reconfigurable Robots

Thomas Joseph Collins and Wei-Min Shen

Information Sciences Institute, University of Southern California,
Los Angeles, California, USA
collinst@usc.edu, shen@isi.edu

Abstract. Integrated and adaptive locomotion and manipulation (IALM) is a key capability for robots to perform real-world applications in challenging environments. It requires interleaving many tasks, sometimes simultaneously, and switching the functions and roles of body components on demand. For example, for autonomous assembly in space, a multiple-tentacle single body "octopus" may have to become a distributed group of "ant" robots, while a hand-like end-effector useful in one case may have to function as an anchor foot in a different situation. This paper presents a general control framework for coordinating high-dimensional dexterous locomotion and manipulation in self-reconfigurable robotic tree structures. The controller is implemented on the SuperBot robotic system and validated in real-time, high fidelity, physics-based simulation. The results have shown many promising capabilities in high-dimensional, dynamic kinematic control for locomotion, manipulation, and self-reconfiguration essential for future autonomous assembly applications.

Keywords: Self-reconfigurable robots, manipulation, autonomous assembly

1 Introduction

Real-world applications of robotic systems in challenging environments often demand extraordinary capabilities. For example, Figure 1 shows a potential high-payoff and high-risk application for autonomous assembly in space.

To accomplish such challenging tasks, a robotic system must self-assemble large structures from modular components. It needs to plan its course of actions, transport pieces from storage to working sites, and manipulate components for alignment, docking, and secure assembly. To meet these challenges, self-reconfigurable robots may offer some critical advantages over fixed-shape robots. For example, self-reconfigurable robots may provide on-demand shape optimization, resilience to single-point failures, and flexible, low-cost launch options. In this paper, we will focus on the capability of interleaved and even simultaneous IALM.

There are several technical challenges to achieve IALM. Specifically, a robot must deal with the high numbers of degrees of freedom (DOF) required for dextrous and precise manipulation, the fact that global configuration information is not at a central location but distributed among the network of modules, and the fact that interaction points with the environment such as the location of a hand and a foot are dynamic

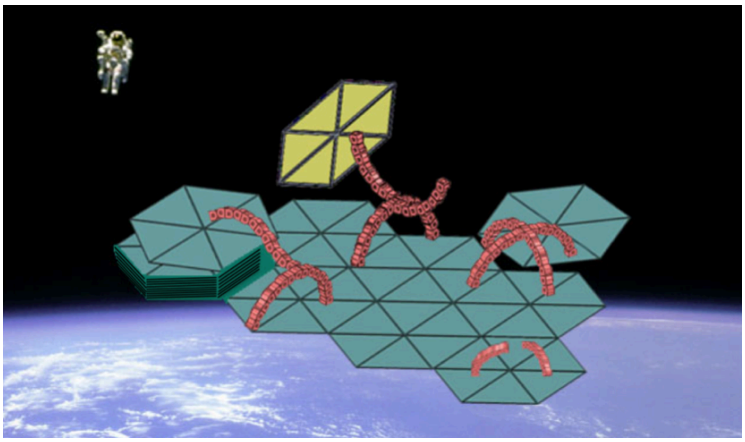

Fig. 1. Autonomous assembly of large surfaces in space using self-reconfigurable robots.

and not completely known in advance. Such challenges make it difficult to directly apply any traditional manipulation techniques because we cannot assume a fixed ”base” from which kinematics can be computed and, additionally, the roles of the modules change dynamically from situation to situation. For example, a ”foot” for walking in one step may need to become a ”hand” for grasping in the next step. Furthermore, self-reconfiguration changes the underlying kinematic structure of the tree itself, necessitating novel approaches.

The contributions of this paper include an integrated controller for adaptive and simultaneous locomotion and manipulation (named *loco-manipulation*). The configuration information of the robotic tree is distributed throughout the network of modules, and a ”brain” module is elected dynamically from step to step using local message passing. High-dimensional and precision manipulation is accomplished via a combination of the provably-convergent Particle Swarm Optimization (PSO) variant called Branch and Bound Particle Swarm Optimization [18] (for inverse kinematics) and the RRT-Connect path planner [8]. This controller enables general tree structures of self-reconfigurable robotic modules to perform sophisticated locomotion and manipulation tasks simultaneously and safely (i.e., without collision). For self-reconfiguration, this controller computes the joint angles that enable the modules in the tree to self-reconfigure into a different tree for better performance in loco-manipulation tasks. An efficient, dynamic kinematic representation keeps track of the current kinematics of the tree. The proposed controller is implemented using the SuperBot [14] robotic system concept and validated in a high-fidelity physics-based simulation for autonomous assembly in a micro-gravity environment. The results are encouraging, demonstrating that a self-reconfigurable robotic system is able to change its configuration on-demand, transport components, and assemble a simple structure at a given working site.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 details the proposed controller with system architecture and sub-system descriptions. Section 4 presents results for integrated and adaptive locomotion and ma-

nipulation with an application of autonomous assembly in mind, and finally Section 5 concludes the paper with future work.

2 Related Work

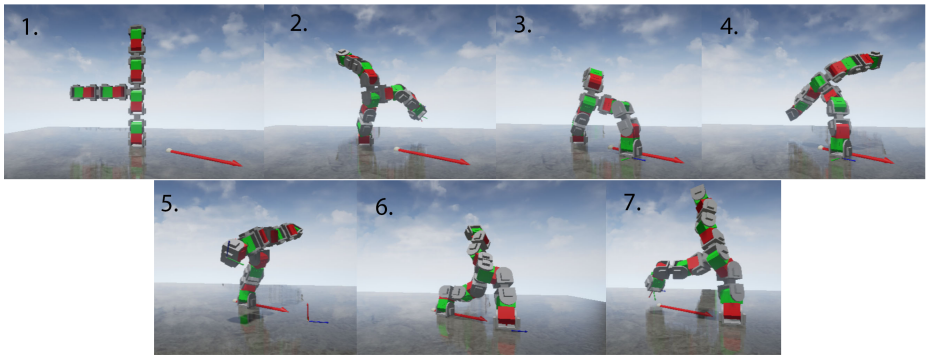

Fig. 2. Sample locomotion results. A 6-module, 18-DOF tree structure of SuperBot modules locomotes on the planar ground surface in a given direction by repeatedly anchoring itself to the ground along the given direction.

A number of modular and self-reconfigurable robot hardware systems have been proposed, including [10, 17, 3, 13, 20, 14]. In many of these systems, distributed algorithms have been developed for locomotion, manipulation, and self-reconfiguration. These algorithms tend to be intimately tied to the modules in question and not broadly applicable. On the other hand, a number of general and powerful algorithms that have been developed to control modular and self-reconfigurable robot systems. In [9], Moll et. al proposed a distributed algorithm for controlling the center of mass of arbitrary kinematic trees of self-reconfigurable modules. Shen et. al proposed a digital hormone model for controlling self-reconfigurable, modular swarms of robots in [16]. However, general-purpose IALM is still an open problem.

Manipulation using physically-connected modular robots has been looked at primarily from a low-level control perspective, such as in [6], where control laws were automatically generated to follow a known reference trajectory, and a hardware perspective, such as in [22]. In [21], the self-assembly of robotic manipulators made of heterogeneous active and passive components is successfully demonstrated, and manipulation is performed with the assembled manipulators. Modules in this work move on a discrete grid which greatly simplifies locomotion. Additionally, in [1], the cooperative locomotion and manipulation (transport) of passive components by self-reconfigurable robot manipulators was successfully demonstrated. In this work, only serial manipulators were used, and locomotion took place on a discretized grid.

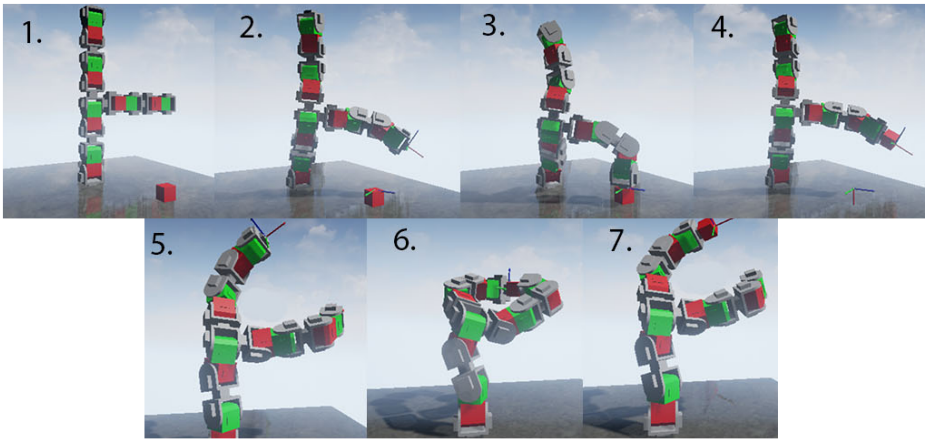

Fig. 3. Sample manipulation results. A 6-module 18-DOF tree structure of SuperBot modules picks up an object and passes it from one open end-effector to another. This is a behavior never before demonstrated, even in simulation, on a distributed system of modular robots.

Some recent work outside the realm of modular robotics has focused on controlling robots with many arms and legs, such as [19]. However, such systems are centralized, and their kinematic structures are fixed.

The use of Particle Swarm Optimization, particularly to solve the inverse kinematics problem, has been studied in [11, 12, 5], with encouraging results. The use of PSO allows for the elimination of the well known pitfalls of traditional Jacobian inverse kinematic methods [2] (singularities, poor scaling with DOF, difficulty in applying them to dynamic situations with kinematic structure changes). This work represents the first application of Branch and Bound Particle Swarm Optimization [18].

3 Overview of the IALM Controller

3.1 Overview

The proposed controller, Algorithm 1, executes as a distributed behavior independently and in parallel on each module. Modules have no fixed ID numbers and can only communicate with modules physically docked to their connectors. The overall system of modules is assumed to be a tree structure (no loops), in which a single module is attached to some fixed structure (e.g., the ground). A single module is elected at each step to (1) map the kinematic structure of the tree from its perspective (see Section 3.2), (2) compute a set of collision-free joint angles for the tree that solve the next task or subtask (e.g., picking up an object, see Section 3.3), and (3) plan a collision-free path from the current joint angles to the goal joint angles. In general, any module could be elected to be the leader. In this work, the leader is always chosen to be the module *currently* connected to the ground. This leader is called the *kinematic brain*. Note that this is purely for convenience of kinematics computations and more sophisticated leader election or task protocols, such as those presented in [15], could be used instead.

Algorithm 1: Overview of the tree control algorithm proposed in this work.

```
   Input:
   Tasks: a queue of tasks to perform
1  Function CONTROLTREE()
2      updateTree := ShouldUpdateTree();
3      ProcessIncomingMessages();
4      Brain = AmIConnectedToGround();
5      if Brain == true then
6          if updateTree == true then
7              ClearAndReset(KinMap);
8              KinMap := DiscoverKinematicStructure(); // Distributed BFS,
                Section 3.2
9          else if (Task := GetCurrentTask()) then
10             GoalA := FindGoalAngles(Task);        // BBPSOIK, Section 3.3
11             Path := PlanPath(CurrentAngles(), GoalA);   // Use RRT-Connect
12             ExecutePath(Path);
13             TransferBrainData(DidBrainChange(Task));
```

First, all messages are processed (Line 3). This happens at all modules. Modules relay state information (joint angles, connector statuses, etc.) to the current kinematic brain and listen to the brain's commands to set their joint angles or dock/undock with other modules. If the kinematic representation of the tree is out of date (Line 2) – which occurs, e.g., when the joint angles of any module move more some small pre-specified amount or a connector's state changes – then the kinematic representation of the tree is re-computed by the kinematic brain (Lines 7-8). Finally, with an up-to-date kinematic map of the structure, the next task or subtask (such as picking up a certain object, placing an object, locomoting in a certain direction, etc.) is determined based on some high-level goal (Line 9). These high-level goals are generally generated externally. The BBPSOIK procedure discussed below in Section 3.3 is used to find an optimal set of goal joint angles that would perform the desired subtask and meet the necessary error tolerance to enable successful docking (Line 10). The RRT-Connect path planner is then used (Line 11) to plan a collision-free motion path from the current joint angles of the tree to the computed goal angles. This path is then executed (Line 12). Finally, if the brain is to change based on the execution of the task – the brain changes at each locomotion step, as a new module connects to the ground – the state information of the current brain module is transmitted to the new brain module (Line 13).

As an example of generating subtasks from externally generated goals, consider Figure 3. The externally generated goal is to pick up the object and switch it from one end-effector to the other open end-effector. The first time Line 9 is hit, the kinematic brain realizes the object has not been picked up. The generated subtask then becomes finding a set of joint angles that would align an open end-effector of the tree with the object to grasp it. Once the object has been grasped, the next subtask would be to find a set of joint angles that aligned the object precisely with the other open end-effector in the tree in order for the switch from one effector to the other to occur.

3.2 Kinematic Structure Discovery

The controller presented here assumes that a forward kinematics model of the module types involved in the tree are known to each module. It is convenient if this forward kinematics model is given in the form of two functions $to(C, \mathbf{q_i})$ and $from(C, \mathbf{q_i})$, where C is any connector face of the module in question, and $\mathbf{q_i}$ is a set of joint angles for module i. It is assumed that any pair of connectors can dock to one another at one pre-specified orientation. Then, these functions give the local homogeneous transformations to each connector from some specified center pose of the module and from each connector to the center of the module, respectively (and with respect to the same coordinate system). $\mathbf{q} = \{\mathbf{q_i}\}$ is reserved to denote the joint angles of the entire tree as a set of joint angles of each module.

Given these two functions, it is possible to compute the pose of any module i relative to some base frame for any tree joint angles $\mathbf{q}$ by post-multiplying an alternating sequence of these $to(C, \mathbf{q_i})$ and $from(C, \mathbf{q_i})$ transformations along the shortest path from the *brain* module – the single module connected to the ground and the most convenient base module of kinematics calculations – to module i. Assume T_G is the pose of the foot module's connector that is docked to the ground relative to some base frame (e.g., the center pose of the foot module). Let $L_C = \{C_1, C_2, ..., C_{2k}\}$ be the (even-numbered, as pairs of connectors are involved in docking modules together) list of connectors along the shortest path from base module 1 to module i in question. Let C_0 be the connector docked to the ground. Then, the relative pose of module 2's center, for example, is given by $T_2 = T_G from(C_0, \mathbf{q_1}) to(C_1, \mathbf{q_1}) from(C_2, \mathbf{q_2})$.

Once the poses of every module i, $\{T_i\}$ in the tree are specified relative to the same base frame, the pose of any connector of module i can be easily represented in the same base frame by post-multiplying to the module pose the corresponding $to(C, \mathbf{q_i})$ for the connector in question. This dual-layer kinematic representation makes it easy to use the poses of module centers for tasks like collision detection while simultaneously making it easy to generate the pose of any connector in the tree, some of which are being considered as end-effectors. If each module has these functions for each module type involved in the system, this can be easily extended to heterogeneous trees. Using this representation and a distributed breadth-first search procedure, the kinematic brain module builds a *kinematic map* of module nodes that represents the kinematic structure of the tree from the point of view of the current brain module. Each kinematic node discovered reports its current state values, including its joint angles, end-effector status, connector statuses, etc. The connector path from the kinematic brain module to each module in the tree is saved and used subsequently to communicate with specific modules as needed (e.g., to command them to set their joint angles or connect to an object). The kinematic map is used for both inverse kinematics calculations (Section 3.3) and path planning.

3.3 Inverse Kinematics (IK)

Inverse Kinematics as Optimization Assuming, for the moment, that the modular robot tree in question is simply one centralized system, is quite simple to transform any IK problem into an optimization problem. Consider workspace goal pose T. Assume

that a forward kinematics model $W = K(\mathbf{q})$ is given, where $\mathbf{q}$ is a vector of the tree's joint angles and W is the workspace pose of the end-effector corresponding to joint angles $\mathbf{q}$. Let $C(\mathbf{q})$ be a collision function which returns 0 if a set of joint angles is collision free and 1 otherwise. Then, the IK problem can be solved by minimizing:

$$F(\mathbf{q}) = a_p P_{error}(\mathbf{q}) + a_o O_{error}(\mathbf{q}) + a_c C(\mathbf{q}) \tag{1}$$

In the above equation P_{error} is the Euclidean position distance between T and W, while O_{error} is some measure of orientation error between T and W. a_p and a_o are optional constants weighing the differing importance of P_{error} and O_{error} (as they are measured on different scales). There are a number of ways to measure O_{error}, but, for this work, the magnitude of difference in Roll-Pitch-Yaw Euler Angles (in degrees) between T and W is minimized. Eq. 1 can be generalized to the case of multiple end-effectors by summing up the P_{error}'s, O_{error}'s and collision errors for each end effector and minimizing one large sum. Multi-objective optimization methods could be used instead, but they are left for future work.

For the kinematic self-reconfiguration problem, in which a configuration must be found to align to modular robot connectors for docking, a very similar function can be minimized. In Eq. 1, P_{error} and O_{error} are error terms relative to a fixed target T. In the self-reconfiguration problem, there is no fixed target T. Rather, there are two end effector poses $\{W_1, W_2\} = K(\mathbf{q})$ returned by the forward kinematics model and the error between them must be minimized. By redefining P_{error} to be the Euclidean position distance between W_1 and W_2 and redefining O_{error} to be the magnitude of difference in Roll-Pitch-Yaw Euler Angles (in degrees) between W_1 and W_2, Eq. 1 can again be minimized.

Branch and Bound Particle Swarm Optimization Simply put, Branch and Bound PSO (BBPSO, [18]) is an embedding of PSO within the branch and bound framework [7]. Assuming the global minimum of the function $F(S^n)$ to be minimized is known (where $S^n \subseteq R^n$, e.g., the space of possible joint angles subject to joint limits), PSO is used as a metaheuristic to find the current upper bound α_i of each partition, as the search space is recursively and exhaustively split. The only change required to the PSO algorithm is that each swarm must search only in the bounds of the partition element in which it is spawned. The known global minimum value of F is the β_i (lower bound) for each partition. The convergence of the above algorithm to the global minimum of F (assuming a known minimum value of F) and the convergence in a finite amount of time given *any* positive error ε is theoretically proved in [18].

By minimizing Eq. 1 using the BBPSO framework – note that traditional PSO could be used instead, but it does not provide global convergence guarantees – in the space of the joint angles of the tree of modules, one arrives at a globally convergent and optimal inverse kinematics and kinematic self-reconfiguration solution, which the authors of the present work call BBPSOIK. The is the first application of the BBPSO algorithm to the inverse kinematics (IK) problem, which results in the first globally convergent, optimal inverse kinematics solver applicable in general to modular robots that are physically connected in tree structures. Note that this claim is due to the fact that the function F in

Eq. 1 has a known theoretical lower bound minimum of 0, regardless of the number of end-effectors (provided, of course, the problem has a solution).

4 Results, Validation, and Discussion

4.1 BBPSO as an IK/self-reconfiguration solver (BBPSOIK)

In previous work [4], we evaluated traditional PSO as a high-DOF IK solver. Though the results were highly encouraging, traditional PSO does not provably converge to globally optimal solutions. Though it often works well in practice, it is not possible to *guarantee* a solution of acceptable quality is produced in a finite amount of time using basic PSO. BBPSO, on the other hand, does provide these theoretical guarantees of solution quality, which is vitally important in the sort of dangerous environments being considered. Motivated by this, the authors performed another suite of tests, this time evaluating BBPSOIK as an inverse kinematics (and kinematic self-reconfiguration) solver. The configurations used (with end-effectors highlighted yellow) are visually shown in Figure 4.

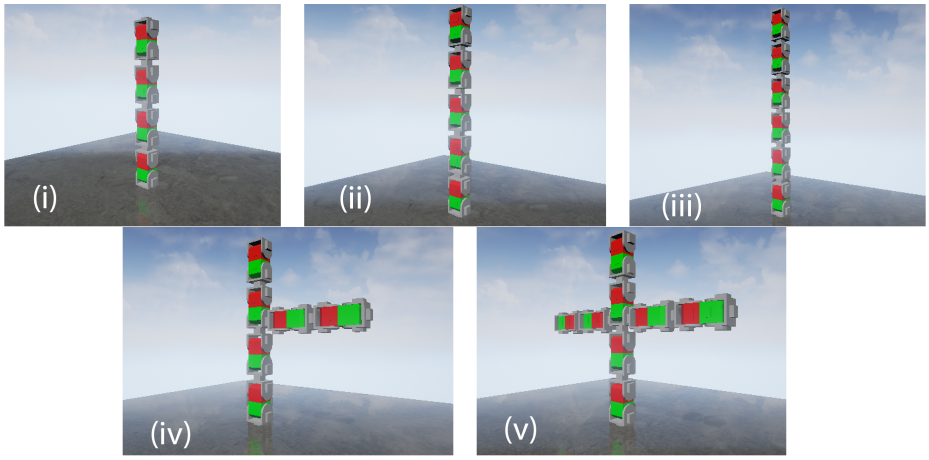

Fig. 4. The configurations of SuperBot modules used to validate the proposed BBPSOIK solver. (i) A 4-SuperBot snake (12 DOF, 1 end-effector); (ii) A 5-SuperBot snake (15 DOF, 1 end-effector); (iii) A 6-SuperBot snake (18 DOF, 1 end-effector); (iv) A 6-SuperBot tree (18 DOF, 2 end-effectors); (v) A 9-SuperBot tree (27 DOF, 3 end-effectors).

For clarity of presentation, the test cases are divided into the following categories (applicable tree configurations from Figure 4 are given in parentheses):

1. Category I: Solve position and orientation IK for all end-effectors ((i) - (v)).
2. Category II: Solve position and orientation IK for one end-effector((iv), (v)).
3. Category III: Solve position IK for all end-effectors ((i) - (v)).

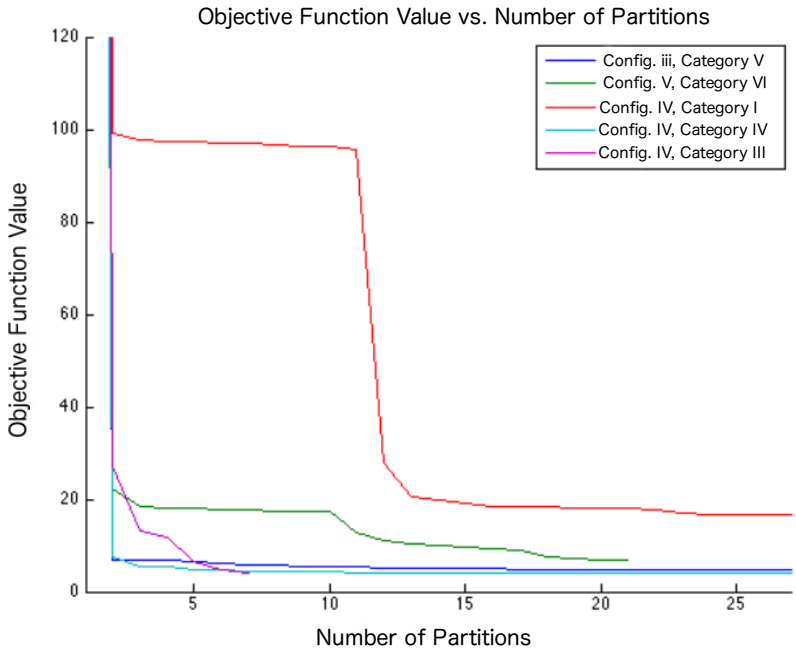

Fig. 5. Example algorithm runs showing monotonic error decreases with the number of active partitions.

Config.,Category	$\varepsilon-$Success	ε	Avg. Error	Avg. Partitions	Avg. Runtime
(i), I	96/100	0.004	0.00374	6.7	20.1s
(ii), I	99/100	0.004	0.00374	6.13	19.6s
(iii), I	95/100	0.004	0.00367	6.79	29.2s
(iv), I	46/100	0.05	0.069	18.12	126.1s
(i), III	100/100	0.004	0.00303	1	0.08s
(ii), III	100/100	0.004	0.00291	1	0.13s
(iii), III	100/100	0.004	0.00295	1	0.19s
(iv), III	99/100	0.004	0.00355	3.95	15.4s
(v), III	46/100	0.004	0.0221	15.49	139.1s
(iii), V	94/100	0.006	0.00624	4.91	24.2s
(iv), VII	97/100	0.006	0.00583	4.78	16.4s
(v), VI	87/100	0.006	0.00647	7.26	49.39s
(iv), IV	59/100	0.025	0.037	14.9	100.2s
(iv), II	93/100	0.004	0.00436	7.02	31.4s
(v), II	94/100	0.004	0.00363	5.73	33.5s

Fig. 6. BBPSOIK Results.

4. Category IV: Solve position IK on one end-effector and position and orientation IK on the other ((iv)).
5. Category V: Solve kinematic self-reconfiguration problem to reconfigure from configuration (iii) to configuration (iv).
6. Category VI: Solve kinematic self-reconfiguration problem to reconfigure configuration (v) by randomly selecting two end-effectors to connect.
7. Category VII: Solve kinematic self-reconfiguration problem to reconfigure from configuration (iv) to configuration (iii).

For tests in category II, one tree end-effector was randomly selected during each of the 100 test runs. Figure 6 tabulates the results. Each row is a configuration/test category pair. For each such pair, the algorithm was run 100 times. ε represents the error tolerance given to the program. The column ε-Success is the percentage of the 100 test cases in which a solution of acceptable quality was found within a fixed time limit (200 seconds, leading to at most 27 partitions). The Avg. Partitions and Avg. Runtime columns gives the average number of branch and bound partitions and average runtime (over the 100 runs, including ε-failures). Since ε is a combined measure of position and orientation error (see Equation 1), selecting it is not completely straightforward. If a_p an a_o are both 1 (as they are in this work), orientation error is much more heavily penalized than position error, and ε will primarily constitute position error (in meters). Based on experiments the authors have performed with new versions of the SINGO connector of SuperBot modules, the position tolerance of the connector is approximately 5-6 *mm* with very little tolerance for orientation error. Therefore, ε values in most of the tests above were chosen to be between 0.004 and 0.006. The cases in which ε is much greater correspond to cases where BBPSOIK has difficulty converging to such small ε values. Figure 5 shows sample numerical runs in which it is observed that the error monotonically decreases as a function of the number of partitions active. This provides validation that the spawning of partition elements in BBPSO forces PSO out of local minima, decreasing solution error as expected. In each partition, 20 particles were used with a maximum iteration count of 50. The authors observed that smaller particle swarms with smaller maximum iteration counts allowed for more branching, which more quickly forced PSO out of local minima, leading to better performance than with larger swarms. Figure 6 shows that the proposed algorithm converges well to optimal solutions for single-end-effector position and orientation IK problems, two-end-effector position IK problems, and problems in categories V-VII. It has difficulty consistently converging quickly when the position and orientation IK of multiple end effectors must be solved simultaneously and for tests in category IV. This makes intuitive sense, given that they are difficult multi-objective optimization problems. Future work will apply more advanced multi-objective PSO methods to such problems within a branch and bound framework.

4.2 Loco-Manipulation Results

The controller proposed in this work is a novel combination of an efficient kinematic representation and discovery procedure, a novel application of BBPSOIK for solving inverse kinematics and kinematic self-reconfiguration problems, and a probabilistically

complete path planner (RRT-connect). This combination, including the ability of the kinematic brain module to dynamically move from module to module, permits the controller to select optimal collision-free joint angles that solve tasks and then move in collision-free paths to those joint configurations, regardless of dynamic changes to the kinematic base frame or the structural configuration of the modules (i.e., their connections to one another). This facilitates adaptive loco-manipulation, including novel behaviors never before demonstrated on modular robotic systems. Note that, in the subsequent results, the end-effectors of the robots are assumed to be able to connect to the ground or objects at any orientation around the vector normal to the ground or the object. The environment, including the poses of the ground and objects, are assumed known to each module.

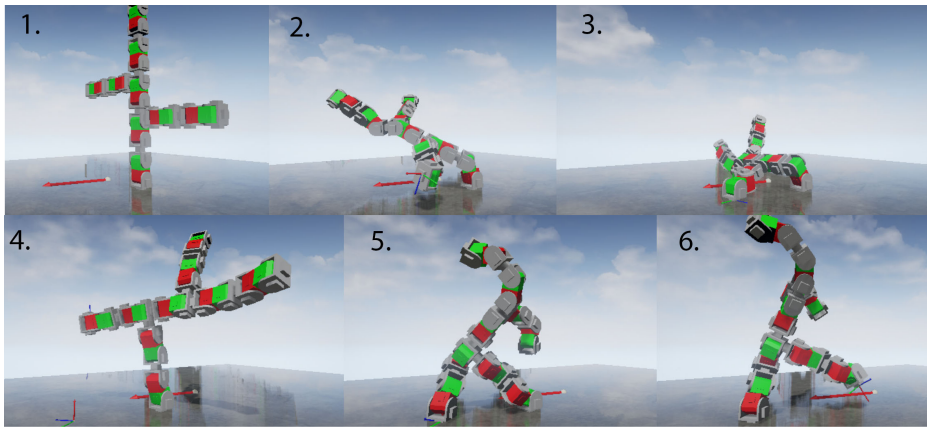

Fig. 7. Example locomotion results with an 8-module 24-DOF SuperBot tree with 4 extremities/end-effectors.

Locomotion Figure 7 demonstrates locomotion using an 8-module (24-DOF) SuperBot tree with 4 extremities/end-effectors. At each step in the locomotion, a random free end-effector is selected, BBPSOIK is used to find a collision-free set of joint angles that aligns the end-effector with the ground in the direction of locomotion (the red arrow in the figure), and RRT-connect plans a collision-free path to the goal angles. Once the path is executed, the end-effector module attaches to the ground, becoming the new kinematic brain, and the process is repeated. Locomotion toward a target location is also possible. In such cases, the direction of motion is dynamically determined based on the position difference between the current kinematic brain module and the target location. Though locomotion with modular robots has been demonstrated many times before, gaits to facilitate locomotion are most often tied intimately to the hardware and module configuration used. This represents one of the first general-purpose locomotion strategies theoretically applicable to any tree of modular or self-reconfigurable robots.

This test was repeated for each configuration in Figure 4 for 20 randomly selected motion directions.

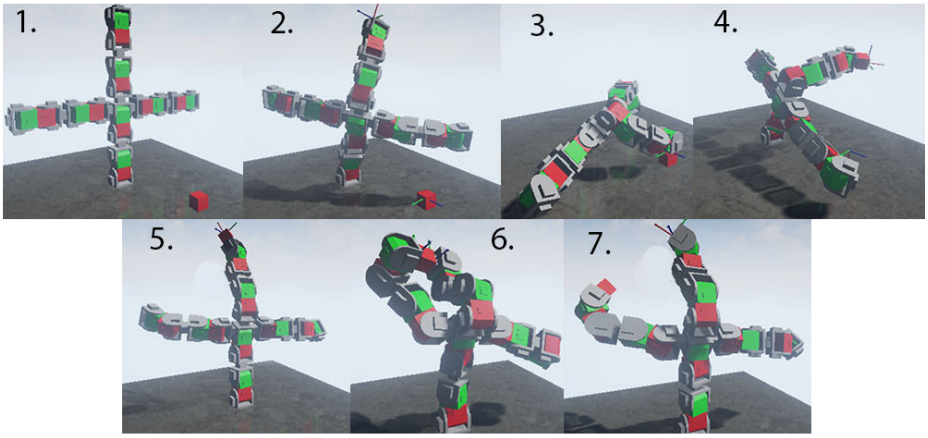

Fig. 8. Example manipulation results with an 8-module 24-DOF SuperBot tree with 4 extremities/end-effectors. The object is picked up by one end-effector before being passed from to another open end-effector.

Manipulation Figure 8 demonstrates manipulation using an 8-module (24-DOF) SuperBot tree. The red object is picked up by one randomly-selected end-effector and passed to another open end-effector in the tree. This is a novel manipulation behavior never before demonstrated (even in simulation) using modular or self-reconfigurable robots. The kinematic brain keeps track of which end-effectors are connected to objects (of known geometry), so the object dynamically becomes part of the kinematic representation of the tree as long as it remains connected to one of the modules. This enables the controller to facilitate the transportation (simultaneous locomotion and manipulation) of passive objects from place to place while ensuring that the object being carried does not collide with the robot or its environment along the way. This test was repeated with configurations (iv) and (v) in Figure 4 for 20 randomly selected (but reachable) object locations.

Self-Reconfiguration Figure 9 demonstrates self-reconfiguration of a 6-module 18-DOF SuperBot tree into a long snake and vice versa. This is achieved by reducing the problem of lining up two end-effectors for self-reconfiguration to an optimization problem and efficiently solving using BBPSOIK. This allows the tree structures in question to change their structural configurations (connections) to best match the task at hand. The kinematic discovery procedure dynamically adapts to these changes, and, assuming

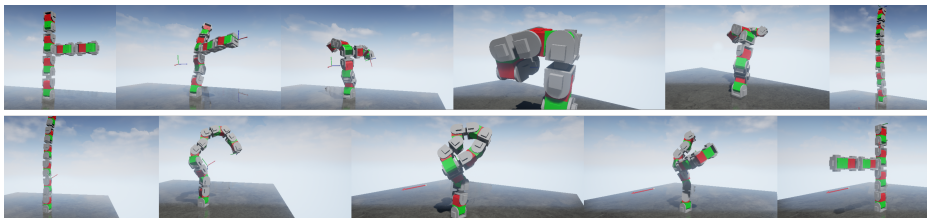

Fig. 9. Top: A 6-module, 18-DOF SuperBot tree reconfiguring into a snake. Bottom: A 6-module, 18-DOF SuperBot snake manipulator reconfiguring into a 3-extremity tree.

the changes result in a new tree structure of modules, the kinematic brain can immediately use the new tree structure and its extremities for loco-manipulation.

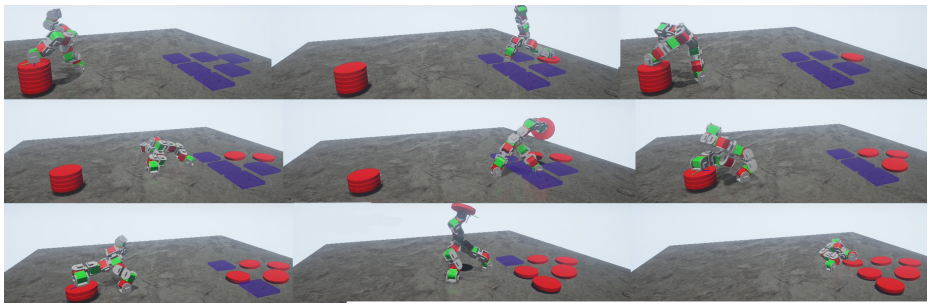

Fig. 10. Top left to bottom right: a demonstration of a 6-module, 18-DOF SuperBot tree performing a pick, carry, and place task. The six red cylinders are grabbed (in descending order of height) from the pile and placed in the goal areas.

Autonomous Building by Pick, Carry, Place One of the goal usages of this framework is to enable trees of robots with multiple extremities to autonomously build structures. As a first step toward this type of behavior, Figure 10 demonstrates a 6-module, 18-DOF SuperBot tree locomoting to, picking up, transporting, and placing the six red cylindrical objects in their respective goal areas (blue). This is a novel demonstration of locomotion with manipulation never before performed by a system of modular robots (even in simulation). It differs from the collaborative manipulation in [1], as the proposed controller makes use of only a single tree of modules (rather than multiple serial manipulators) to perform the transportation and manipulation. Also, the robot is free to move anywhere on the ground plane, as its foot placements are not discretize. Future work will demonstrate the autonomous building of more complex structures with different sizes and shapes of building materials.

Limitations The proposed controller has been shown to facilitate novel loco-manipulation using self-reconfigurable, modular robot trees. However, some points are worth mentioning. First, the framework currently assumes a single point of contact, with motors strong enough to support the entire structure during loco-manipulation activities. This assumption is more reasonable in environments such as outer space and under water, both of which are prime examples of dangerous and isolated environments, but it nevertheless provides a limitation.

Currently, the controller operates on a simulated distributed set of robotic modules in a hybrid distributed and centralized fashion, with the most important computations happening in a centralized place (the kinematic brain). The benefit of performing computations this way, particularly in dangerous environments, is that claims about the optimality and collision-freeness of solutions for inverse kinematics (IK) and kinematic self-reconfiguration can be made from a global perspective to ensure the safe operation of the system in the environment. However, a fully distributed methodology would provide increased robustness and fault-tolerance.

5 Conclusions and Future Work

This work proposed a novel, hybrid centralized and distributed controller based on Branch and Bound Particle Swarm Optimization (BBPSO) and Rapidly-Exploring Random Trees (RRT-connect) that takes an important step in facilitating adaptive loco-manipulation in trees of modular and self-reconfigurable robots. This work represents the first application of BBPSO to the problems of inverse kinematics and kinematic self-reconfiguration.

Results in physics-based simulation demonstrate the generality and power of the proposed approach and include demonstrations of loco-manipulation behaviors never before shown on systems of modular or self-reconfigurable robots. Future work will aim to enable multi-contact support during loco-manipulation and parallel kinematic structures of modules (such as those with loops). A fully distributed implementation of the controller is also currently being developed. Further validation of the controller, including validation on different module types, more complex tree configurations, and more complex autonomous building tasks, will all be performed.

References

1. Bonardi, S., Vespignani, M., Moeckel, R., Ijspeert, A.J.: Collaborative manipulation and transport of passive pieces using the self-reconfigurable modular robots roombots. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2406–2412 (2013). DOI 10.1109/IROS.2013.6696694
2. Buss, S.R.: Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. IEEE Journal of Robotics and Automation **17**(1-19), 16 (2004)
3. Castano, A., Shen, W.M., Will, P.: Conro: Towards deployable robots with inter-robots metamorphic capabilities. Autonomous Robots **8**(3), 309–324 (2000)

4. Collins, T., Shen, W.M.: Particle swarm optimization for high-dof inverse kinematics. In: Control, Automation, and Robotics, 2017. Proceedings. 2017 IEEE International Conference on, vol. 2, pp. 1049–1054. IEEE (2017)
5. Durmus, B., Temurtas, H., Gun, A.: An inverse kinematics solution using particle swarm optimization. In: Proc. of sixth International Advanced Technologies Symposium (IATS?11), Turkey, pp. 193–197 (2011)
6. Giusti, A., Althoff, M.: Automatic centralized controller design for modular and reconfigurable robot manipulators. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, pp. 3268–3275. IEEE (2015)
7. Horst, R., Tuy, H.: Global optimization: Deterministic approaches. Springer Science & Business Media (2013)
8. Kuffner, J.J., LaValle, S.M.: Rrt-connect: An efficient approach to single-query path planning. In: Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, vol. 2, pp. 995–1001. IEEE (2000)
9. Moll, M., Will, P., Krivokon, M., Shen, W.M.: Distributed control of the center of mass of a modular robot. In: Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pp. 4710–4715. IEEE (2006)
10. Murata, S., Yoshida, E., Kamimura, A., Kurokawa, H., Tomita, K., Kokaji, S.: M-tran: Self-reconfigurable modular robotic system. Mechatronics, IEEE/ASME Transactions on **7**(4), 431–441 (2002)
11. Rokbani, N., Alimi, A.: Inverse kinematics using particle swarm optimization, a statistical analysis. Procedia Engineering **64**(0), 1602 – 1611 (2013). DOI http://dx.doi.org/10.1016/j.proeng.2013.09.242
12. Rokbani, N., Alimi, A.M.: Ik-pso, pso inverse kinematics solver with application to biped gait generation. arXiv preprint arXiv:1212.1798 (2012)
13. Romanishin, J.W., Gilpin, K., Rus, D.: M-blocks: Momentum-driven, magnetic modular robots. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, pp. 4288–4295. IEEE (2013)
14. Salemi, B., Moll, M., Shen, W.M.: Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. In: Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pp. 3636–3641 (2006). DOI 10.1109/IROS.2006.281719
15. Salemi, B., Will, P., Shen, W.M.: Autonomous Discovery and Functional Response to Topology Change in Self-Reconfigurable Robots, pp. 364–384. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). DOI 10.1007/3-540-32834-3_16. URL http://dx.doi.org/10.1007/3-540-32834-3_16
16. Shen, W.M., Will, P., Galstyan, A., Chuong, C.M.: Hormone-inspired self-organization and distributed control of robotic swarms. Autonomous Robots **17**(1), 93–105 (2004)
17. Sprowitz, A., Pouya, S., Bonardi, S., Van den Kieboom, J., Möckel, R., Billard, A., Dillenbourg, P., Ijspeert, A.J.: Roombots: reconfigurable robots for adaptive furniture. Computational Intelligence Magazine, IEEE **5**(3), 20–32 (2010)
18. Tang, Z., Bagchi, K.K.: Globally convergent particle swarm optimization via branch-and-bound. Computer and Information Science **3**(4), 60 (2010)
19. Tonneau, S., Del Prete, A., Pettré, J., Park, C., Manocha, D., Mansard, N.: An efficient acyclic contact planner for multiped robots (2016)
20. Yoon, Y., Rus, D.: Shady3d: A robot that climbs 3d trusses. In: Robotics and Automation, 2007 IEEE International Conference on, pp. 4071–4076. IEEE (2007)
21. Yun, S.k., Rus, D.: Self assembly of modular manipulators with active and passive modules. In: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pp. 1477–1482. IEEE (2008)
22. Zhu, W.H., Lamarche, T., Dupuis, E., Martin, E.: Modular robot manipulators with precision control