Globally Convergent Optimal Dynamic Inverse Kinematics for Distributed Modular and Self-Reconfigurable Robot Trees

Thomas Collins and Wei-Min Shen

***Abstract*— Kinematic trees of self-reconfigurable, modular robots are difficult to control for at least three primary reasons: (1) they must be controlled in a distributed fashion, (2) they are often kinematically redundant or hyper-redundant, and (3) in many cases, these robots must be designed to *safely* operate autonomously in dangerous and isolated environments. Much work has been done to design hardware, distributed algorithms, and controllers to handle different aspects of this challenging problem, but the design of generalized and globally optimal inverse kinematics algorithms for such systems is largely an open problem. Jacobian-based methods have well-documented shortcomings, particularly for high-DOF systems, while alternative methods, such as those based on genetic and evolutionary algorithms, provide no guarantees of convergence to a globally optimal solution. Such a guarantee is particularly important in the types of dangerous environments in which these robots are to operate. This paper proposes a novel distributed inverse kinematics framework based on the recently proposed Branch and Bound Particle Swarm Optimization (BB-PSO) algorithm, which provably converges to a globally optimal solution (and converges in finite time given *any* positive error tolerance). This framework is demonstrated, through extensive simulations, to offer high-quality solutions in practical amounts of time, even for multi-effector and *dynamic* problems, such as those encountered in kinematic self-reconfiguration where the effector workspace goal pose is not available as input.**

I. INTRODUCTION

It has been a long-standing goal of self-reconfigurable and modular robotics to create kinematic robotic systems capable of autonomous locomotion, manipulation, and self-reconfiguration in dangerous environments without human interaction. Controlling trees of self-reconfigurable, modular robots is extremely difficult due primarily to the distributed nature of the computation involved, the high number of degrees of freedom (DOF) often present, and the need for such systems to operate *safely* – i.e., without damaging themselves or important environment features – in dangerous or isolated environments.

A number of hardware platforms have been proposed to achieve these goals, each with their own distributed control systems, novel locomotion and manipulation algorithms, docking mechanisms, and inherent advantages, disadvantages, and target applications. Many distributed controllers and self-reconfiguration algorithms applicable to broad classes of modules and modular manipulators have been proposed with very promising results.

However, very little work has been done in the area of finding generalized, convergent, and globally optimal *inverse kinematics* (IK) algorithms applicable to distributed kinematic trees of self-reconfigurable or modular robots. The *inverse kinematics problem* of a self-reconfigurable, modular robot tree is the same as the well known manipulation problem of finding a configuration of the kinematic tree that aligns the position and/or orientation of one or more of its end-effectors with given target workspace pose(s).

Traditional manipulation techniques, such as controllers based on the manipulator Jacobian, incrementally move toward a configuration that reduces the error between the poses of current end-effectors and target end-effector poses. These methods have well-documented shortcomings – such as instability at singularities – particularly for high-DOF systems. Furthermore, it may be necessary to fully compute a *globally optimal* solution configuration before any execution is performed to ensure the final configuration will not do damage to the manipulator tree or important elements in its environment.

Alternative IK solution strategies such as those based on numerical approximations, genetic algorithms, evolutionary algorithms, and other AI techniques, often work well in practice but cannot provide guarantees that they will always find globally optimal solutions or even always find solutions of sufficient quality, given a specified error tolerance (in cases where solutions of a certain quality are known to exist).

Furthermore, it is unclear how the above solution strategies would solve what this work refers to as the *dynamic inverse kinematics* (D-IK) problem, which is an important problem encountered when considering kinematic trees of hybrid-type or chain-type self-reconfigurable robot modules that must autonomously self-reconfigure. This problem can be stated as follows: given two chosen end-effectors of the $k \geq 2$ end-effectors of the tree in question, how can a configuration of the tree be found to precisely align these end-effectors with one another such that self-reconfiguration could occur by docking those effectors together? This could also be called the *kinematic self reconfiguration* problem. It is clearly a different problem than the IK problem, as the workspace pose at which the two end-effectors are to meet is not known *a priori*. Figure 1 illustrates these two problems.

Note that naive attempts to cast this as a traditional IK problem – e.g., by fixing one end-effector in space and solving the IK problem using that effector pose as the target pose – may result in sub-optimal solutions or unsolvable problems as it is unclear where the first end-effector should be placed, which end-effector should be chosen first, etc., as well as how obstacles would affect these decisions.

This work proposes a distributed optimization algorithm

Thomas Collins and Wei-Min Shen are with Information Sciences Institute, The University of Southern California, Los Angeles, U.S.A. collinst@usc.edu, and shen@isi.edu

Fig. 1: Left: An example of an *inverse kinematics* (IK) problem for a 6-Module SuperBot tree with two end effectors labeled EE1 and EE2. A 6D (3 position, 3 orientation) goal pose for EE1 is given as input. The tree is shown in one possible solution configuration. Right: An example of a *dynamic inverse kinematics* (D-IK) problem for the same tree. The workspace pose at which EE1 and EE2 align is not known *a priori*. (b.) and (c.) show one such solution configuration at different viewpoints.

for trees of modular or self-reconfigurable robots that (1) provably converges to globally optimal solutions to traditional IK problems involving one or more end-effectors in the tree (and converges in finite time given *any* positive error tolerance), and (2) provably converges to globally optimal solutions to D-IK problems (and converges in finite time given *any* positive error tolerance). This algorithm is based on the recently proposed Branch and Bound Particle Swarm Optimization algorithm (BB-PSO) [1]. The proposed algorithm requires only a forward kinematics model of the module(s) involved in the tree, making it applicable to any arbitrary tree structure of rigid body robot modules.

Section II discusses related work. Section III reviews the branch and bound framework, Particle Swarm Optimization (PSO), and BB-PSO. Section IV details the proposed algorithm. Section V presents extensive results with simulated SuperBot modules showing that the algorithm is practically efficient in a number of important scenarios.

II. RELATED WORK

A. Inverse Kinematics

The most obvious globally optimal inverse kinematics methods are those that solve the problem analytically. Unfortunately, the IK problem cannot be solved analytically in general. Thus, one must usually resort to numerical, approximate, or optimization algorithms to solve the problem, such as those based on the manipulator Jacobian. These approaches are still some of the most widely utilized [2]. They are particularly well suited for creating real-time, online controllers. However, many such approaches suffer from numerical issues around singularities and do not scale well with the number of DOF.

The limitations of Jacobian-based methods have led to a plethora of alternative solution strategies, including novel numerical techniques (e.g., [3]), those based on neural networks and fuzzy logic (e.g., [4], [5]), genetic algorithms (e.g., [6]), probability theory (e.g., [7]). Optimization approaches such as those based on Particle Swarm Optimization (PSO) (e.g., [8], [9], [10]) and the Firefly Algorithm [11] have also been proposed. Very recent work in [12] proposed solving the IK problem for a 12-DOF hyper-redundant manipulator using simulated annealing and the NSGA II genetic algorithm. Several surveys of IK techniques exist, such as [13].

Most of the methods reviewed above are local solutions, meaning that they either do not provably converge to a solution and/or are susceptible to becoming caught in local minima. It is also unclear how the above approaches would solve the D-IK problem for self-reconfiguration as the proposed work does.

B. Modular and Self-Reconfigurable Robots

Over the last 20 or so years, a plethora of modular and self-reconfigurable robot hardware systems have been developed, including [14], [15], [16], [17], [18], [19], [20], [21]. In many of these systems, distributed algorithms have been developed for various tasks, including locomotion, manipulation, forward and inverse kinematics, and self-reconfiguration. These algorithms tend to be intimately tied to the modules in question and not broadly applicable as they are primarily designed to validate the capabilities of the hardware in question.

There are, however, some powerful algorithms that have been developed to control self-reconfigurable and modular robot systems, including trees, that have broad applicability across module types, sometimes even to heterogeneous groups of modules. In [22], 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 or modular swarms of robots in [23]. General solutions to self-reconfiguration for certain classes of robots or under certain specific assumptions have been developed (e.g., [24], [25]). In these reconfiguration algorithms, the kinematics of the modules is usually assumed to be very simple or is not considered for the sake of simplicity.

Modular and self-reconfigurable manipulation has been looked at primarily from a control perspective, such as in [26] and a hardware perspective, such as in [27]. In [28], the self-assembly of robotic manipulators made of heterogeneous active and passive components is successfully demonstrated, and a distributed inverse kinematics algorithm is presented. This IK algorithm is based on Jacobian damped least squares. It approximates joint angle displacements iteratively, rather than solving the problem globally (making it potentially less safe in dangerous environments) and also does not consider the D-IK problem. In [29], Shen et. al presented a distributed solution the inverse kinematics problem for docking in 3D (two position DOFs and one orientation DOF). To the best of our knowledge, there is no directly related work on developing a general and distributed globally optimal and

provably convergent full 6D D-IK and IK algorithm for trees of modular and self-reconfigurable robots.

Finally, it is noted that lattice-based systems, such as [17], need not solve the types of IK and D-IK problems considered here for hybrid and chain-type systems as the lattice facilitates connections on a discrete, regular grid.

III. ALGORITHM BACKGROUND

A. Particle Swarm Optimization

Particle Swarm Optimization (PSO) [30] is a swarm-based optimization algorithm that has been shown to be effective in solving difficult optimization problems in many diverse domains. The basic idea is that a swarm of m particles, each with n dimensions, performs a randomized search in the space of possible n-dimensional solutions while communicating with other particles and maintaining its own history. Each particle i is a point x_i in the given search space $S^n \subseteq R^n$ with velocity v_i and has an associated error value equal to $F(x_i)$, where F is the function to be minimized. Particles move around in this search space randomly but particle movement is biased toward a random weighted average of the direction to the best (lowest $F(x_i)$) position achieved by any particle in the swarm g (social component) and the direction to the best position achieved by each particle individually, p_i (history component). This focuses random searches on areas of the search space where a global optimum is expected to be. g and p_i are updated at each iteration. This randomized searching process continues until a certain fitness threshold h is reached by some particle in the swarm (i.e., a low enough value of F is found) or a maximum number of iterations N is performed. At termination, the global best position found (g) is returned.

B. Branch and Bound Framework

The branch and bound framework refers to a class of global optimization algorithms in which a finite search space is recursively and exhaustively searched until a global optimum (minimum, in this case) of some function is found. Branch and bound algorithms work by partitioning a search space recursively according to some *branching* rule. At each step, a list of currently active partition elements (those portions of the search space that may still contain a solution) is kept. Also at each step, a *search* strategy determines which partition element to further *refine* (i.e., further partition), leading to more and more active partition elements in this list as the algorithm iterates. A *bounding* rule keeps track of, for each active partition element, an upper (worst known solution error) and lower (best possible solution error) bound (α_i and β_i, respectively) on possible solutions in that partition element. After each branching and bounding operation, the overall upper and lower bounds for the entire search space across all active partition elements $\alpha = \min \alpha_i$ and $\beta = \min \beta_i$ are updated. When $\alpha - \beta = 0$, the global minimum has been found. For more details on the algorithm, please see [1], [31].

Note that to ensure convergence in a finite amount of time, an error tolerance $\epsilon = \alpha - \beta \in R^+$ must be selected and the global minimum of F, which is, of course, the best choice for β_i at each partition element, must be known (which, as is shown, is the case for the problems considered).

C. Branch and Bound PSO

Simply put, Branch and Bound PSO (BB-PSO) [1] is an embedding of PSO within the branch and bound framework. Assuming the global minimum of the function $F(S^n)$ to be minimized is known (where S^n is a hypercube in R^n) – which, as is shown later, is the case for the problems in question in this work – PSO is used as a metaheuristic to estimate the upper bound α_i of each partition. The only change required to the PSO algorithm is that each swarm must search only in the bounds of the partition element hypercube in which it spawned. The global minimum value of F is the β_i for each partition.

At the start of the algorithm, one active partition element exists $M_1 = S^n$, which represents the entire search space. This partition element is selected for further refinement and is split into $k \geq 2$ partition elements (according to a branching rule) $\{M_{1,i}\}$, where $\cup_i M_{1,i} = M_1$ and $i \in \{1, 2, ..., k\}$. The set $\{M_{1,i}\}$ now replaces M_1 in the active partition element list. Each active partition element $M_{1,i}$ has known lower bound $\beta_{1,i} = minF$, and PSO is used with error function F to estimate the upper bound $\alpha_{1,i}$. $\alpha = \min \alpha_{1,i}$ and $\beta = \min \beta_{1,i}$. This completes one iteration of the branch and bound PSO procedure. In the next iteration, one of the k active partition elements $\{M_{1,i}\}$ is chosen for further refinement (partitioning) and PSO is again used on all active partition elements (including the new spawned sub-partition elements and previously existing elements) to estimate the alpha values. This process continues, generating more and more active partition elements, until $\alpha - \beta \leq \epsilon$ for some solution $\mathbf{q}$. The global optimum ($\mathbf{q}$) to the given error tolerance has been found and the algorithm terminates.

The convergence of the above algorithm to the global minimum of F and the convergence in a finite amount of time given positive error ϵ is theoretically proved in [1].

IV. THE PROPOSED CONVERGENT, OPTIMAL IK AND D-IK DISTRIBUTED ALGORITHM

In the previous section, the branch and bound framework was discussed as well as a way to combine it with PSO to create a globally convergent swarm optimization procedure to minimize function F whose domain is hypercube $S^n \subset R^n$. This is an important result, as PSO by itself provides no guarantees of convergence to globally optimal solutions. However, BB-PSO is an abstract framework and many details must be filled in to make it suitable to solve IK and D-IK problems. This section first discusses the transformation of the general IK and D-IK problems for self-reconfigurable, modular robot trees into optimization problems with associated objective functions to be minimized. These objective functions are shown to have a known global minimum values (thus guaranteeing convergence to the global optimum and convergence in finite time given $\epsilon \in R^+$ error tolerance as discussed above). A simple and efficient branching operation

is also selected to keep this algorithm as computationally efficient in practice as possible. Finally, this section provides the algorithmic details necessary to implement the proposed algorithm in a distributed fashion on a set of robot modules.

A. IK and D-IK as Optimization Problems

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 joint angles (the tree configuration) 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 configuration is free and 1 otherwise. The binary nature of a collision function defined this way makes it applicable to any environment and configuration of modules. Then, the IK problem can be solved by minimizing:

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

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). a_c should be large enough such that configurations that are not free are never within the error tolerance ϵ. For example, if ϵ is 0.3 and $a_c = 1000$, no non-free configuration could ever be within the given error tolerance. 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. If singularities are of concern, quaternions could be easily used instead.

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 not considered in this work.

For the D-IK problem, 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 D-IK 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, this time to solve the D-IK problem. Note that, in the case of D-IK problems, there must usually be some fixed homogeneous transformation applied to one of W_1 or W_2 to ensure that making the two effectors converge to the same pose has the intended consequence of precisely aligning them.

It is also important to note that the function F in Eq. 1 – whether for the IK problem or D-IK problem and regardless of the number of end-effectors – has a theoretical lower bound minimum value of 0, corresponding to a perfect solution in which the end effector and target pose are perfectly aligned and the manipulator/tree is in a free configuration.

B. Globally Convergent, Optimal IK using BB-PSO

Algorithm 1: BB-PSO for D-IK and IK problems

Input:
ϵ : error threshold
F : objective/error function (Eq. 1)
S^n : initial search hypercube with bounds
$MParts$: maximum allowable partitions
Output: g: best particle found

```
1  Function BBPSOIK()
2      β := 0;
3      α := MAX_FLOAT;
4      M := {S^n} ;          // active partitions
5      g := RandSolution();
6      while α − β ≥ ε and M.size() < MParts do
7          foreach active partition M_i do
8              {bestPos, α_i} := PSOBound(ε,F,M_i);
9              if α_i < α then
10                 g := bestPos;
11                 α := α_i;
12         i := RandomPartition();
13         Remove partition M_i from list M;
14         {M_{i,1}, M_{i,2}} := BranchPartition(i);
15         Replace M_i with M_{i,1} and M_{i,2} in M;
16     return g;
```

As illustrated in Algorithm 1, by minimizing Eq. 1 using the BB-PSO framework, one arrives at a globally convergent and optimal D-IK and IK solution. Assuming the algorithm operates on partitions that are hypercubes of R^n (which is true in the case of searching the continuous space of joint angles of robot manipulators with joint limits), the proposed algorithm refines a randomly chosen partition element into two new partition elements by cutting the partition with an $n-1$ dimensional hyperplane at the midpoint of a randomly chosen dimension (Line 14, $BranchPartition()$). In general, it is possible to create more than two new partitions at each step, but this quickly becomes computationally infeasible. As [1] point out, this *branching* operation maintains the theoretical properties necessary to guarantee global convergence in a finite amount of time. Any active partition is, by construction, a hypercube in R^n and can clearly be further refined in this manner.

Partition elements are selected randomly for further refinement with equal probability given to all partition elements (Line 12, $RandomPartition()$). The $PSOBound()$ function in Line 8 is simply the use of traditional PSO to minimize objective function F (Eq. 1) in the bounds of the current partition element. This returns both a best solution position and best solution fitness ($bestPos$ and α_i, respectively).

For the problems considered in this work, the authors suggest that for PSO smaller swarms of particles (e.g., 30-50) should be used with small maximum iteration counts (e.g., 50-100) to allow more branching to occur, forcing PSO out of local minima. Intuitively, as a portion of the search space is iteratively refined, more and more particles are searching it. Furthermore, refinement leads to swarms searching smaller and smaller partition elements (subspaces), such that enormous maximum iteration counts will likely not lead to much solution improvement over smaller maximum iteration counts as the number of partitions grows. The ranges given are based on extensive experimentation done by the authors, but these numbers obviously depend on the computational and time resources available.

C. BB-PSO on Distributed Robotic Modules

Algorithm 2 is a high-level overview of the distributed, globally convergent, globally optimal D-IK and IK solution proposed in this work. Modules are assumed to have some fixed number of dock connectors through which they communicate with other modules via message passing only when connected directly to them. Modules have no fixed ID's or roles. A sensor that allows a module to determine whether or not it is connected to the ground (Line 2, $AmIConnectedToGround()$) is assumed to be available. Algorithm 2 executes independently on each module.

At the start of the algorithm, the single module attached to the ground (the *kinematic base module*, where $ctg == true$ in Line 5) discovers the kinematic structure of the tree using a messaging passing breadth-first search (Line 7, $BuildKinematicMapBFS()$). This kinematic base module starts by discovering each module directly attached to itself (one message hop away) with state probing messages. All other modules wait for messages (Line 4, $ProcessIncomingMessages()$). Each discovered module responds with important state information including its current joint angles, module type, and whether or not it is an end-effector. Each discovered module is put in a FIFO queue by the kinematic base module such that its connectors can be enumerated later.

The kinematic base module keeps a *kinematic map* of the modules found (*nodes*) and assigns ID's to these nodes (numbering them in the order they were discovered). Since module kinematics are assumed to be known, it suffices to store with each node the sequence of connectors (*connector list*) from the kinematic base module to that node along the shortest possible path along with the types of modules encountered on that list. After all modules one hop away are discovered (through an enumeration of all the kinematic base module's connectors), the kinematic base module discovers modules two message hops away by enumerating the connectors of each module one hop away. This process continues until all connectors of all modules have been enumerated and the queue is empty.

Note that the kinematic base module is assumed to also be acting as the central brain module for the kinematics computations in this algorithm. In theory, the kinematic base module and the brain module could be different modules, if desired. Additionally, multiple modules could perform these kinematics computations simultaneously, each discovering the structure of the tree from their own viewpoint. This could be a useful redundancy measure and will be considered in future work.

Algorithm 2: Distributed BB-PSO for D-IK and IK

```
1  Function BBPSOIKBehavior()
2      ctg := AmIConnectedToGround();
3      updateMap := IsMapUpdated();
4      ProcessIncomingMessages();
5      if ctg == true then
6          if updateMap == true then
7              BuildKinematicMapBFS();
8          else
9              g := BBPSOIK();
10     return g;
```

The complication here, when comparing this procedure to a regular breadth-first search, is that state-probing messages intended for modules k hops away must be relayed through the modules $k-1$ and fewer hops away. The generated map is valid until reconfiguration occurs, in which case it must be updated. This will change the flag $updateMap$ (Line 3).

In order to minimize Eq. 1 using PSO for $BBPSOIK()$ in Line 9 once the map is completed, the kinematic base module must be able to check the entire tree for collisions and self-collisions (i.e., implement $C(\mathbf{q})$) and compute the relative pose of the end-effectors in the tree to calculate P_{error} and O_{error} for any possible configuration of the tree $\mathbf{q}$. The easiest way to achieve this is with a hierarchical representation: first, compute the relative pose of each module to the base by traversing the connector list in the kinematic map from the kinematic base module to each other module, post-multiplying to the base frame transformation the appropriate local homogeneous transformations to and from each connector on the list as appropriate. Once each module's relative pose has been determined, the relative pose of any connector of any module is simply that module's relative pose post-multiplied by the local homogeneous transformation to the desired connector. Some of these connectors will obviously be acting as end-effectors, while the poses of each module are useful for geometric collision detection procedures.

V. EXPERIMENTS

In this section, the proposed algorithm is validated in physics-based simulation [32]. Solutions to a number of IK and D-IK problems are computed using different snakes and tree structures of simulated SuperBot modules (each 3DOF, with three revolute joints, illustrated in Fig. 4), each running Algorithm 2. Figure 2 shows the tested configurations (i) - (v) and their end-effectors (yellow). For IK problems, goal end-effector poses were determined by selecting a random free configuration of the tree or snake and giving the algorithm

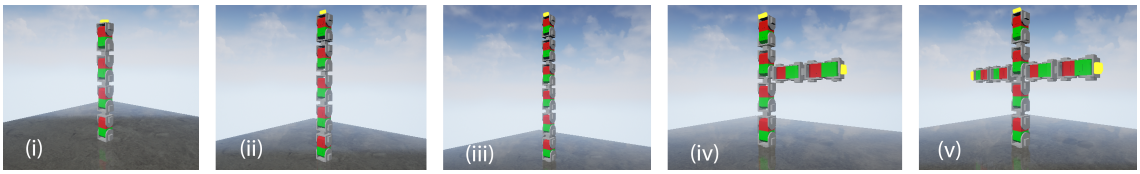

Fig. 2: The configurations of SuperBot modules used to validate the proposed algorithm. (i) A 4-SuperBot snake (12 DOF, 1 end-effector); (ii) A 5-SuperBot snake (15 DOF, 1 end-effector); (iii) A 6-SuperBot snake (18 DOF, 1 end-effector); (iv) A 6-SuperBot tree (18 DOF, 2 end-effectors); (v) A 9-SuperBot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

the corresponding end-effector pose(s). This ensured that the IK problem was solvable. For D-IK problems, the goal was to find a configuration in which the two chosen end-effectors were precisely aligned with one another. There are multiple such solutions to these D-IK problems, so, in general, different runs of the same D-IK test produced different solutions. For all tests, PSO was run for 100 iterations with 50 particles. P_{error} was measured in mm and O_{error} was measured as described in section IV. ϵ values between 4.0 and 6.0 produced solutions almost visually indistinguishable from the target poses.

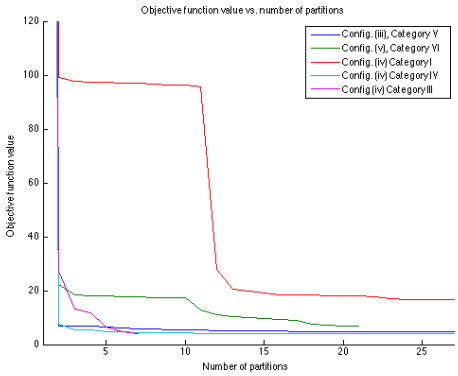

Fig. 3: Example algorithm runs showing monotonic error decreases with the number of active partitions for several different configuration, test category pairs. Note that the curve ends when the error threshold has been reached.

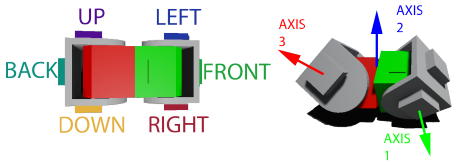

Fig. 4: SuperBot connectors and kinematic structure.

For clarity of presentation, the test cases are divided into the following categories:

1) Category I: Solve position and orientation IK for all end-effectors. Applies to configurations (i) - (v).
2) Category II: Solve position and orientation IK for one end-effector. Applies to trees (iv), (v).
3) Category III: Solve position IK for all end-effectors. Applies to all configurations (i) - (v).
4) Category IV: Solve position IK on one end-effector and solve position and orientation IK on the other. Applies to configuration (iv).
5) Category V: Solve D-IK problem to reconfigure from configuration (iii) to configuration (iv).
6) Category VI: Solve D-IK problem to reconfigure configuration (v) by randomly selecting two end-effectors to connect.
7) Category VII: Solve D-IK problem to reconfigure from configuration (iv) to configuration (iii).

TABLE I: IK and D-IK Results.

Config.,Category	ϵ — Success	ϵ	Avg. Error	Avg. Partitions	Avg. Runtime
(i), I	96/100	4.0	3.74	6.7	20.1s
(ii), I	99/100	4.0	3.74	6.13	19.6s
(iii), I	95/100	4.0	3.67	6.79	29.2s
(iv), I	46/100	50.0	69.01	18.12	126.1s
(i), III	100/100	4.0	3.03	1	0.08s
(ii), III	100/100	4.0	2.91	1	0.13s
(iii), III	100/100	4.0	2.95	1	0.19s
(iv), III	99/100	4.0	3.55	3.95	15.4s
(v), III	46/100	4.0	22.14	15.49	139.1s
(iii), V	94/100	6.0	6.24	4.91	24.2s
(iv), VII	97/100	6.0	5.83	4.78	16.4s
(v), VI	87/100	6.0	6.47	7.26	49.39s
(iv), IV	59/100	25.0	36.51	14.9	100.2s
(iv), II	93/100	4.0	4.36	7.02	31.4s
(v), II	94/100	4.0	3.63	5.73	33.5s

For tests in category II, one tree end-effector was randomly selected during each of the 100 test runs. Table I 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. By construction, the algorithm does not fail to find a solution. Rather, 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). Figure 3 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 BB-PSO forces PSO out of local minima, decreasing solution error as expected. Though this algorithm should in theory be applicable to heterogeneous systems if the different kinematics of the different module types are known to each module, only homogeneous systems are considered in this work.

Table I 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.

VI. CONCLUSIONS AND FUTURE WORK

This work proposed a novel distributed globally convergent, globally optimal inverse kinematics and dynamic inverse kinematics algorithm applicable to general tree structures of modular and self-reconfigurable robots. This work represents the first use of Branch and Bound PSO to inverse kinematics problems and dynamic inverse kinematics problems (such as kinematic reconfiguration). A series of simulations were performed to illustrate that it is capable of finding high-quality solutions to a number of such problems on simulated robots in practical amounts of time. Future work will explore multi-objective optimization methods for better handling multiple end-effector systems, GPU-acceleration to exploit possible parallelism in both the branching part and the PSO part of BB-PSO, and alternative branching schemes to more effectively find solutions as well as provide validation for this algorithm's theoretical applicability to heterogeneous systems.

REFERENCES

[1] Z. Tang and K. K. Bagchi, "Globally convergent particle swarm optimization via branch-and-bound," *Computer and Information Science*, vol. 3, no. 4, p. 60, 2010.

[2] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[3] A. Aristidou and J. Lasenby, "Fabrik: a fast, iterative solver for the inverse kinematics problem," *Graphical Models*, vol. 73, no. 5, pp. 243–260, 2011.

[4] A. Benallegue, B. Daachi, and A. R. Cherif, "Stable neural network adaptive control of constrained redundant robot manipulators," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3. IEEE, 2002, pp. 2193–2199.

[5] R. V. Mayorga and S. Chandana, "A neuro-fuzzy approach for the motion planning of redundant manipulators," in *Neural Networks, 2006. IJCNN'06. International Joint Conference on*. IEEE, 2006, pp. 2873–2878.

[6] Y. Peng and W. Wei, "A new trajectory planning method of redundant manipulator based on adaptive simulated annealing genetic algorithm (asaga)," in *Computational Intelligence and Security, 2006 International Conference on*, vol. 1. IEEE, 2006, pp. 262–265.

[7] N. Courty and E. Arnaud, "Inverse kinematics using sequential monte carlo methods," in *Articulated Motion and Deformable Objects*. Springer, 2008, pp. 1–10.

[8] N. Rokbani and A. Alimi, "Inverse kinematics using particle swarm optimization, a statistical analysis," *Procedia Engineering*, vol. 64, no. 0, pp. 1602 – 1611, 2013.

[9] N. Rokbani and A. M. Alimi, "Ik-pso, pso inverse kinematics solver with application to biped gait generation," *arXiv preprint arXiv:1212.1798*, 2012.

[10] B. Durmus, H. Temurtas, and A. Gun, "An inverse kinematics solution using particle swarm optimization," in *Proc. of sixth International Advanced Technologies Symposium (IATS?11), Turkey*, 2011, pp. 193–197.

[11] N. Rokbani, A. Casals, and A. M. Alimi, "Ik-fa, a new heuristic inverse kinematics solver using firefly algorithm," in *Computational Intelligence Applications in Modeling and Control*. Springer, 2015, pp. 369–395.

[12] P. Costa, J. Lima, A. I. Pereira, P. Costa, and A. Pinto, "An optimization approach for the inverse kinematics of a highly redundant robot," in *Proceedings of the Second International Afro-European Conference for Industrial Advancement AECIA 2015*. Springer, 2016, pp. 433–442.

[13] A. Aristidou and J. Lasenby, *Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver*. University of Cambridge, Department of Engineering, 2009.

[14] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *Mechatronics, IEEE/ASME Transactions on*, vol. 7, no. 4, pp. 431–441, 2002.

[15] A. Sprowitz, S. Pouya, S. Bonardi, J. Van den Kieboom, R. Möckel, A. Billard, P. Dillenbourg, and A. J. Ijspeert, "Roombots: reconfigurable robots for adaptive furniture," *Computational Intelligence Magazine, IEEE*, vol. 5, no. 3, pp. 20–32, 2010.

[16] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: a modular reconfigurable robot," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 514–520.

[17] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund, "Modular atron: Modules for a self-reconfigurable robot," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 2. IEEE, 2004, pp. 2068–2073.

[18] A. Castano, W.-M. Shen, and P. Will, "Conro: Towards deployable robots with inter-robots metamorphic capabilities," *Autonomous Robots*, vol. 8, no. 3, pp. 309–324, 2000.

[19] J. W. Romanishin, K. Gilpin, and D. Rus, "M-blocks: Momentum-driven, magnetic modular robots," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 4288–4295.

[20] J. Davey, N. Kwok, and M. Yim, "Emulating self-reconfigurable robots-design of the smores system," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4464–4469.

[21] B. Salemi, M. Moll, and W.-M. Shen, "Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct 2006, pp. 3636–3641.

[22] M. Moll, P. Will, M. Krivokon, and W.-M. Shen, "Distributed control of the center of mass of a modular robot," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 4710–4715.

[23] W.-M. Shen, P. Will, A. Galstyan, and C.-M. Chuong, "Hormone-inspired self-organization and distributed control of robotic swarms," *Autonomous Robots*, vol. 17, no. 1, pp. 93–105, 2004.

[24] F. Hou and W.-M. Shen, "Graph-based optimal reconfiguration planning for self-reconfigurable robots," vol. 1, no. 1-2, pp. 1–24, Aug. 2013.

[25] R. Fitch and R. McAllister, "Hierarchical planning for self-reconfiguring robots using module kinematics," in *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 477–490.

[26] A. Giusti and M. Althoff, "Automatic centralized controller design for modular and reconfigurable robot manipulators," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 3268–3275.

[27] W.-H. Zhu, T. Lamarche, E. Dupuis, and E. Martin, "Modular robot manipulators with precision control."

[28] S.-k. Yun and D. Rus, "Self assembly of modular manipulators with active and passive modules," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1477–1482.

[29] W.-M. Shen and P. Will, "Docking in self-reconfigurable robots," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 2. IEEE, 2001, pp. 1049–1054.

[30] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, 1995, pp. 39–43.

[31] R. Horst and H. Tuy, *Global optimization: Deterministic approaches*. Springer Science & Business Media, 2013.

[32] T. Collins, N. O. Ranasinghe, and W.-M. Shen, "Remod3d: A high-performance simulator for autonomous, self-reconfigurable robots," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 4281–4287.