Hierarchical Planning for Self-Reconfiguring Robots Using Module Kinematics

Robert Fitch and Rowan McAllister

Abstract Reconfiguration allows a self-reconfiguring modular robot to adapt to its environment. The *reconfiguration planning* problem is one of the key algorithmic challenges in realizing self-reconfiguration. Many existing successful approaches rely on grouping modules together to act as meta-modules. However, we are interested in reconfiguration planning that does not impose fixed meta-module relationships but instead forms cooperative relationships between modules dynamically. This approach avoids the need to hand-code meta-module motions and potentially allows reconfiguration with fewer modules. In this paper we present a general two-level reconfiguration framework. The top level plans in module-connector space using distributed dynamic programming. The lower level accepts a transition function for the kinematic model of the chosen module type as input. As an example, we implement such a transition function for the $3R$, SuperBot-style module. Although not explored in this paper, this general approach is naturally extended to consider power use, clock time, or other quantities of interest.

1 Introduction

Self-reconfiguring modular robots use module disconnections and reconnections to change their overall shape. In so doing, these robots can adapt to the environment or task at hand. Performing such adaptation requires solving the algorithmic problem of computing a sequence of module moves that transforms an initial shape into a goal shape. This problem, known as the *reconfiguration problem*, remains one of the key algorithmic challenges in self-reconfiguring robotics.

There are several dimensions by which to categorize specific instances of the reconfiguration problem. Algorithms have been proposed for specific module types, such as unit-compressible modules [4, 25], and abstract cube modules with simple motion primitives [7]. The idea in planning for an abstract module is to compile down an abstract move into a sequence of native moves. A possible technique to accomplish this is to simplify the problem by treating a group of modules as a single meta-module with fewer kinematic constraints. Two other important issues are

Robert Fitch and Rowan McAllister
Australian Centre for Field Robotics (ACFR). ARC Centre of Excellence for Autonomous Systems
The University of Sydney, Sydney NSW Australia
e-mail: {rfitch, r.mcallister}@acfr.usyd.edu.au

parallelism – how many modules can move at one time – and decentralized versus centralized control. We are interested in the question of autonomous reconfiguration planning that acts directly in the native kinematic action space of the module, and does not use meta-modules. In this paper, we study the problem of general parallel decentralized reconfiguration planning for given module kinematics.

There are several reasons to address this specific variation of the reconfiguration problem. It is useful to consider pairs or small groups of modules working together, but planning for individual modules allows these groupings to be dynamic. Avoiding static meta-modules reduces the minimum number of modules required for reconfiguration. This is especially useful for hardware prototypes with few modules. Furthermore, planning in the native kinematic space of the module opens the possibility of optimizing reconfiguration for various quantities of interest. We are interested in a planner that can consider power use, time cost, and (heterogeneous) modules with differing capabilities. A general planning framework that easily admits changes to its underlying kinematic model also opens the possibility of using reconfiguration simulation as a tool for design optimization. The effects of simplifying a given module design by removing a degree of freedom, for example, could be readily evaluated.

The fundamental challenge in solving the reconfiguration problem is that the number of degrees of freedom in a self-reconfiguring robot increases with the number of modules. The number of possible configurations thus increases exponentially. These combinatorial issues have been understood for many years [18]. Searching this huge space directly is not possible; some structure must be imposed on the problem. The success of meta-module and cube-module planners relies on such a structuring.

Our approach is to build on our earlier planner for abstract cube-shaped modules [9] hierarchically by adding a lower level. The low-level planner computes a sequence of moves, in the joint space of the module, that results connection/ disconnection. This approach decomposes the full problem into many local subproblems. Each subproblem is a kinematic motion planning problem small enough to be solved quickly. Chained together, these solutions move a single module from one point in the robot to another along a sequence of intermediate connections. Point-to-point paths are then computed, as in our abstract cube planner, by formulating a *Markov-decision problem* (MDP) and solving it using distributed dynamic programming. The value function acts as a navigation function over all connectors that indicates the next step towards an open goal position. Many modules share this navigation function. As modules move, the navigation function is updated.

We present our reconfiguration algorithm as a general framework that accepts a module's kinematic model in the form of a transition function. We present a specific transition function for SuperBot-style modules [21] as an example, and illustrate its behavior with simple examples in simulation. Our intention is for this example to provide sufficient information such that other researchers can implement this algorithm on various module types.

The paper is organized as follows. We discuss related work in Sect. 2. In Sect. 3, we present details of our cube-style planner as background information and then

present our general reconfiguration algorithm. We define a sample transition function for SuperBot-style modules in Sect. 4 along with implementation examples in simulation. Sect. 5 concludes the paper with discussion and future work.

2 Related Work

Reconfiguration planning is a well-studied problem. A survey of accomplishments can be found in [26]. Another survey paper is [20]. We briefly discuss a selection of relevant results in this section.

The root of this paper lies in cellular automata-based locomotion [3]. The MILLION MODULE MARCH algorithm [9] can be viewed as a generalization of this idea. The present paper can be viewed as a further generalization along two fronts: goal shape representation and module kinematics.

A number of planners leverage the concept of meta-modules. Important examples include planners for MTRAN [27], ATRON [6] and I-Cubes [19]. The key difference between this work and ours is that we are interested in the question of how to reconfigure without meta-modules.

Complete planners have been developed for unit-compressible modules [4, 25]. Other early work in reconfiguration planning includes [5, 14]. The idea of gradient-based planning is explored in [22] and [23]. A graph-signature method is presented in [1].

A planner for SuperBot modules viewed in a chain-based manner is presented in [11]. Optimal reconfiguration for chain-based robots was recently proven to be NP-complete [12].

3 Hierarchical MDP Planning with Dynamic Programming

The reconfiguration algorithm we propose in this paper builds on our earlier MILLION MODULE MARCH algorithm for scalable locomotion through reconfiguration [9]. In this section we summarize MILLION MODULE MARCH for convenience, focusing on the MDP formulation and dynamic programming solution method. We then present a new MDP formulation that, unlike MILLION MODULE MARCH, models native module kinematics. We define a general reconfiguration algorithm based on this new MDP formulation. Like MILLION MODULE MARCH, this new algorithm is fully decentralized and scalable.

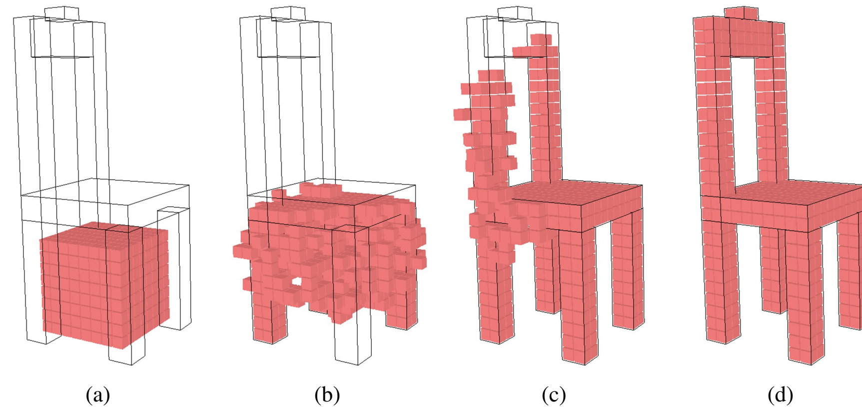

Fig. 1: Reconfiguration example using Algo. 1 and the Sliding Cube module abstraction. Simultaneously executing single module paths results in global reconfiguration. Subfigs. (a) through (d) show four stages of a reconfiguration sequence that assembles a chair shape from an initial cube shape. Simple assembly order heuristics are used that guide modules to the bottom center of the goal shape as it is formed.

3.1 Background: MDP Planning with Abstract Modules

The MILLION MODULE MARCH algorithm was originally presented as a scalable algorithm for locomotion-through-reconfiguration for the Sliding Cube [7] module abstraction. The algorithm produces locomotion by first specifying a goal bounding box at an offset to the current location. Modules move to fill the box, the box is shifted in a receding-horizon fashion, and locomotion results. Providing a different shape for the goal results in reconfiguration into that shape, for convex goal shapes. Non-convex goal shapes are also possible with the addition of local assembly rules that prevent internal holes from forming [13]. Fig. 1 shows an example of reconfiguration into a chair shape. The algorithm is fully decentralized and has been implemented in simulation with million-module systems [9]. It has also been implemented on embedded processors with wireless radio communication in hardware-in-the-loop simulation [10, 15], and extended to control a team of nine mobile robots [8].

The algorithm is composed of two main components: (1) planning via a global navigation function; and (2) control of parallel module movements (connectivity checking) via local graph search and shared locks. The essence of the second component is that each module, in parallel, searches for a local module substructure sufficient to guarantee that it is a non-articulation point in the module connectivity graph. This search is performed using message-passing. If successful, modules in the substructure are temporarily locked (prevented from moving) until the locking module has completed its move. Locks can be shared by multiple moving modules.

Many modules can thus safely move in parallel while preserving global connectivity. This component of the algorithm is used unmodified in the present work. Full implementation details are provided in [10].

The planning component of MILLION MODULE MARCH computes a value function that acts as a global navigation function. Modules use this function as a one-step planner to choose the next move. By sequentially choosing such moves, each module is guided towards an available destination in the goal shape. As many modules move in parallel, the topology of the robot structure also changes. The value function is updated online to reflect these topology changes (continuous replanning).

The planning problem is formulated as a distributed MDP. An MDP is a sequential decision-making problem defined by a $4-tuple < S, A, T, R >$, where S is the set of states, A is the set of actions, T is the transition function that maps state-action pairs to resulting states, and R is a one-step reward function [24]. A decision-making agent repeatedly takes actions and earns rewards. Its objective (commonly) is to maximize the sum of future rewards. If the transition function is known, dynamic programming can be used to solve the MDP. A solution is a policy mapping states to actions. This policy can be encoded as a value function over states. The transition function can be either deterministic or stochastic.

The set of states in MILLION MODULE MARCH is the set of module faces. In the Sliding Cube abstraction, a module is a cube that lives in a cubic lattice. Therefore the set of allowable states can be thought of as open lattice positions adjacent to at least one other module. The Sliding Cube model provides two motion primitives - a sliding move and a convex transition. These primitives define the action set. A module can either make an axis-aligned (lateral) move, or move "diagonally" around another module. The transition function is also defined by these two motion primitives. The reward function is -1 per move.

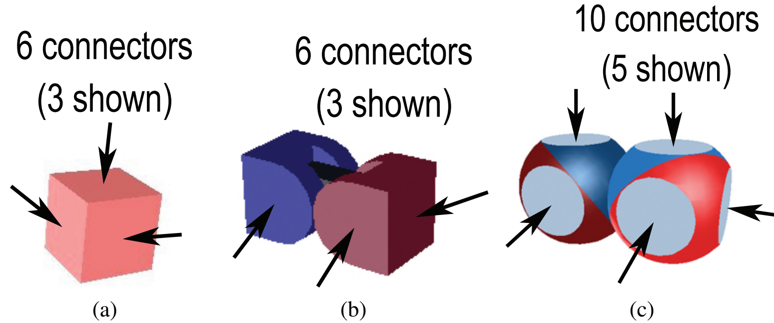

Fig. 2: $M-C$ space is a generalization of the Sliding Cube representation to any module type. The Sliding Cube abstraction, (a), has a connection on every face. Other module types, such as SuperBot-style modules, (b), and Roombot-style modules [1], (c), do not. $M-C$ space is simply defined as the set of all module-connector pairs.

The value function is stored in a distributed fashion. Each module stores the value of states corresponding to its connectors. The MDP is solved using asynchronous distributed dynamic programming implemented with message passing. An update is performed when a module receives a message with a value for a nearby state. Using the transition function, the module updates its local value function and sends these new values to its neighbors. This process is guaranteed to converge in polynomial time in the number of states [17]. A moving module queries the value function by sending a request to its connected neighbors. After a move, changed values are again sent to neighbors and the value function is updated.

3.2 MDP Planning with Native Module Kinematics

Building on the Sliding Cube MDP formulation, we now introduce a new MDP formulation that replaces the Sliding Cube and instead assumes the availability of a kinematic model for a physical module. Instead of abstract motion primitives, motion primitives now correspond to changes in module joint angle and connector state. Because actions are no longer unit-time, this is technically a semi-markov decision problem (SMDP) [2]. However, for the purposes of this paper we assume unit-time actions. The SMDP formulation allows more sophisticated optimization (time, power, etc.) but we will leave this for future work.

To define the state space, we first define the set of module-connector pairs, or *M-C space*. Fig. 2 illustrates sample $M-C$ states for three different module types. The

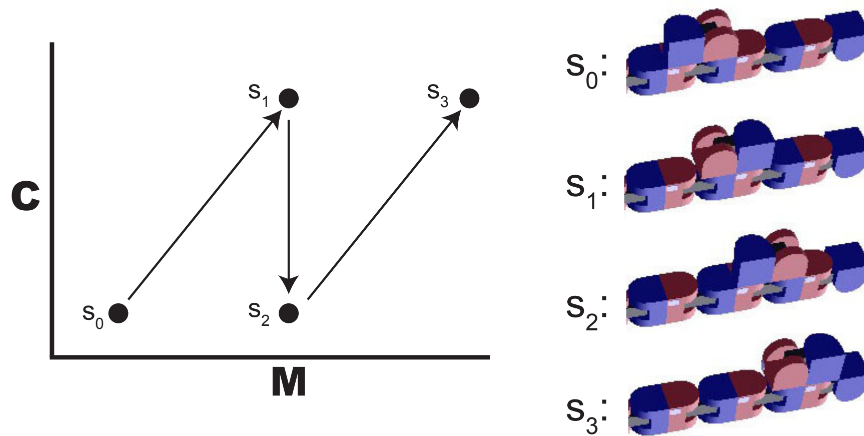

Fig. 3: Path of a single $3R$ module moving through $M-C$ space. States s_i are $M-C$ states. A state transition in $M-C$ space corresponds to a module attaching to a new connector in the robot workspace. Corresponding module movements are illustrated in the right half of the figure.

entire set is not necessarily reachable. One obvious example of a non-reachable state is a connector that is occupied (connected to another module). In general, reachability is determined by the transition function. The transition function, in turn, is partially determined by the robot configuration topology. Therefore, connectivity of $M-C$ space changes with reconfiguration. A module with k connectors can potentially occupy k M-C states simultaneously. Our state space S is therefore defined as the set of all k-tuples of $M-C$ states, including a null $M-C$ state that models a free connector. Fig. 3 shows an example of a single module making state transitions in $M-C$ space and the corresponding module movements in a sample configuration.

The set of actions is defined by the kinematic model. We assume that an action consists of a set of joint angle increments and connection/disconnection actions. An action can involve a single module, or a module plus one or more helper modules.

Actions result in changing state. This means that the set of occupied $M-C$ states will change following a successful action. An action that fails or otherwise does not result in a state change is a null action. The transition model defines this change, mapping a state-action pair to a resulting state: $T(s,a) = s'$, where $s \in S$, $s' \in S$, and $a \in A$. The transition function must take into account surrounding modules. The potential for collision means that not all actions are available at all times. The transition function can be stochastic.

The reward function is -1 for every action. This attempts to minimize the total number of actions. A more sophisticated reward function can be used to minimize other quantities, such as time, power use, heterogeneous modules, etc. Further, the reward function can be modified during reconfiguration to allow the robot to adapt to changes. However, we do not consider these possibilities in this paper.

3.3 Hierarchical Reconfiguration Algorithm

Having formulated the MDP, we solve using dynamic programming. To allow modules to move in parallel, we integrate the parallel movement control approach from MILLION MODULE MARCH. To prevent collisions, we lock all modules within the workspace of a moving module. The algorithm is listed in pseudocode as

Algorithm 1 General framework for reconfiguration.

T: a transition function
G: a goal shape
c: the current robot configuration

Generate value function V for c given T and G using dynamic programming
repeat
  Find mobile modules
  Move mobile modules one step according to V
  Recompute V using new configuration c'
until all modules in goal

Algo. 1. Transition function T, goal configuration G, and start configuration c are assumed as input. In parallel, modules follow a path to the goal by chaining together a sequence of state transitions. Within the goal, modules are guided by local assembly order heuristics as above. The algorithm terminates when all modules are in the goal.

The value function is recomputed as the robot configuration changes, as described in Sect. 3.1. The MDP will converge in polynomial time [17]. Convergence of the robot to the goal shape depends on the transition function supplied.

4 A Local Kinematic Planner for $3R$ Modules

In this section, we flesh out Algo. 1 by defining a transition function based on module kinematics. There are many ways to do this in general. We have chosen to view the problem as motion planning for an $n-$link kinematic chain among regular orthohedral obstacles. Motion planning in high-dimensional spaces is computationally intensive. By imposing this strict structure, we can use a simple grid search method for motion planning. We illustrate this technique with the $3R$ (SuperBot-style) module.

Algorithm 2 A local kinematic planner. This planner dynamically computes the transition function for the reconfiguration MDP.

s_{start}: starting configuration
N: local neighborhood of modules around s_{start}
M: list of moves for output, initially empty
A: set of actions (joint angle increments)
T: search tree, initially empty
S: search queue, initialized with s_{start}

while S not empty **do**
  pop search node s from S
  if s not in T **then**
    add s to T
    if s is a goal configuration in N **then**
      add new move to M
    end if
    for all actions $a \in A$ **do**
      generate new state s' by integrating forward from s
      if path from s to s' is collision-free **then**
        add s' to S
      end if
    end for
  end if
end while

output M

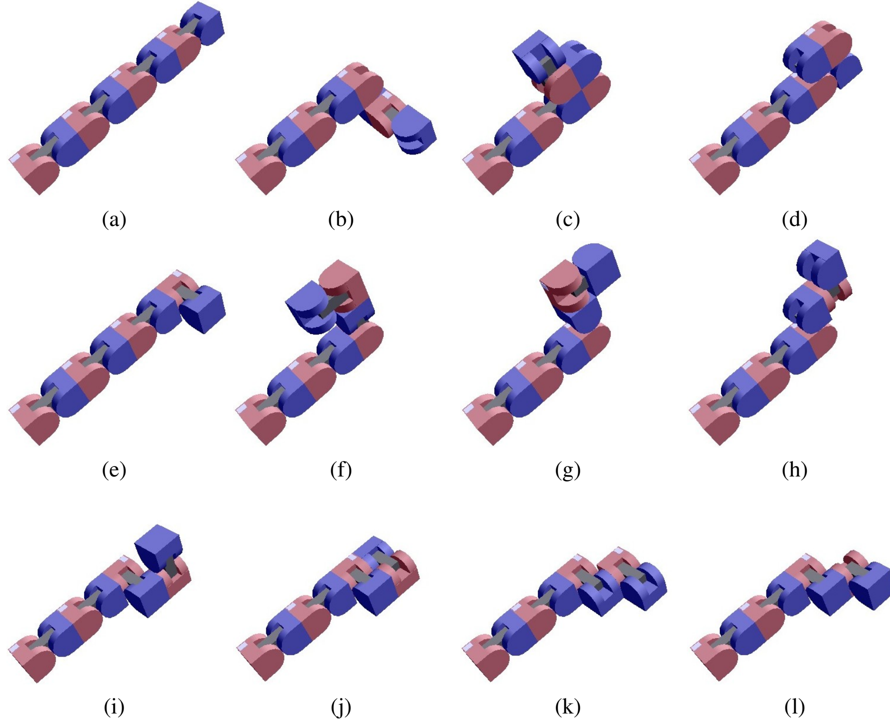

Fig. 4: Workspace reachability. Subfigs. (a) through (d) show sample configurations reachable with a helper module attached in an end-to-end configuration. Likewise, Subfigs. (e) through (h) show samples reachable from an end-to-side configuration. Subfigs. (i) through (l) correspond to a side-to-side configuration.

$M-C-O-\Theta$ space is $M-C$ space augmented by adding two extra dimensions, $O \in \{e,s\}$ and $\Theta \in \{0,90,180,270\}$, that represent how a module is connected to the $M-C$ pair. Due to connector symmetry, we can encode which connector is connected by specifying and end (e) or a side (s). The Θ dimension encodes rotation represented discretely in 90-degree increments.

We consider two cases for planning. The first is single module motion. Given a starting (m,c,o,θ) state, lattice (workspace) position, and vector of joint angles x, we use the forward kinematics of the $3R$ module to determine the position of its connectors in the workspace. We then consider the set of actions formed by all permutations of discrete 90-degree increments/decrements of joint angles. We iterate through this set of actions. At each iteration, we add the joint angle increments to the initial position, resulting in a new joint angle vector x'. We again use the forward kinematics to compute the new position of connectors in workspace. If no connectors are in a position to connect to some other connector in the neighborhood, this configuration is discarded. Otherwise we perform collision checking in

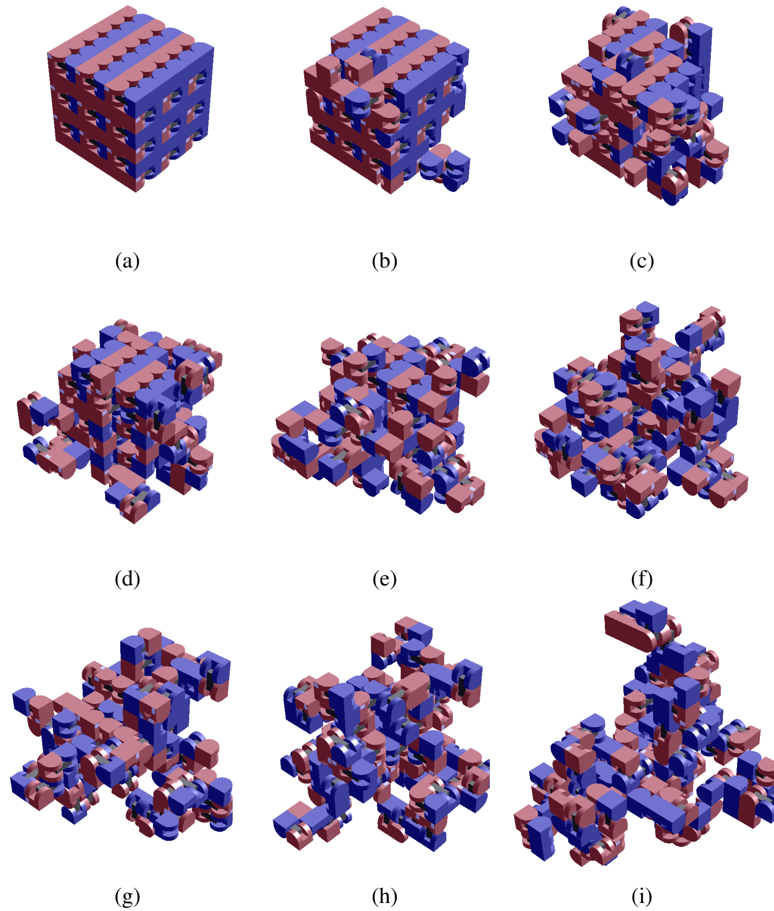

Fig. 5: Sample configurations generated by successive random module movements in a robot with 108 modules.

the workspace. We check intermediate configurations between x and x' in small increments, as described in [16]. If there is a collision, this configuration is discarded. Else we place x' on a queue and continue. We then pop the queue and repeat. When the queue is empty, the algorithm terminates.

The second case involves a helper module. A helper module is a (connected) neighbor. In this case, the joint angle vector includes joints of both modules. We search as described above.

The algorithm is listed in pseudocode as Algo. 2. Fig. 4 shows examples of different $O-\Theta$ configurations. In the helper case, a module can reach positions up to a radius of manhattan distance four from its end connector. Fig. 5 shows examples

drawn from a sequence of configurations generated by successive random module movements.

Because we search all joint angle positions, the running time of this algorithm is exponential in the number of joint angles. For constant-length chains of helper modules, time is constant (albeit with a potentially large constant factor). For two $3R$ modules, the size of the search space is $3*4*3*3*4*3 = 1296$. This is reasonable to implement with modest embedded computational resources, even considering that in computing the value function, each module must perform this computation for each possible (o,θ) pair $(2*4 = 8)$ and each of its open connectors.

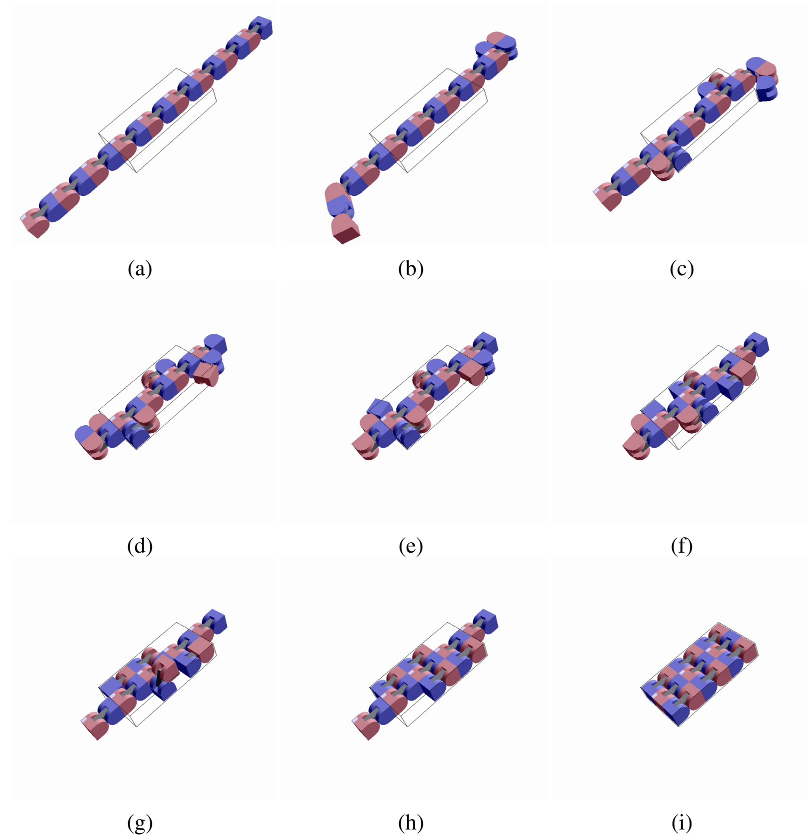

Fig. 6: Nine modules reconfiguring from a line shape into a box shape.

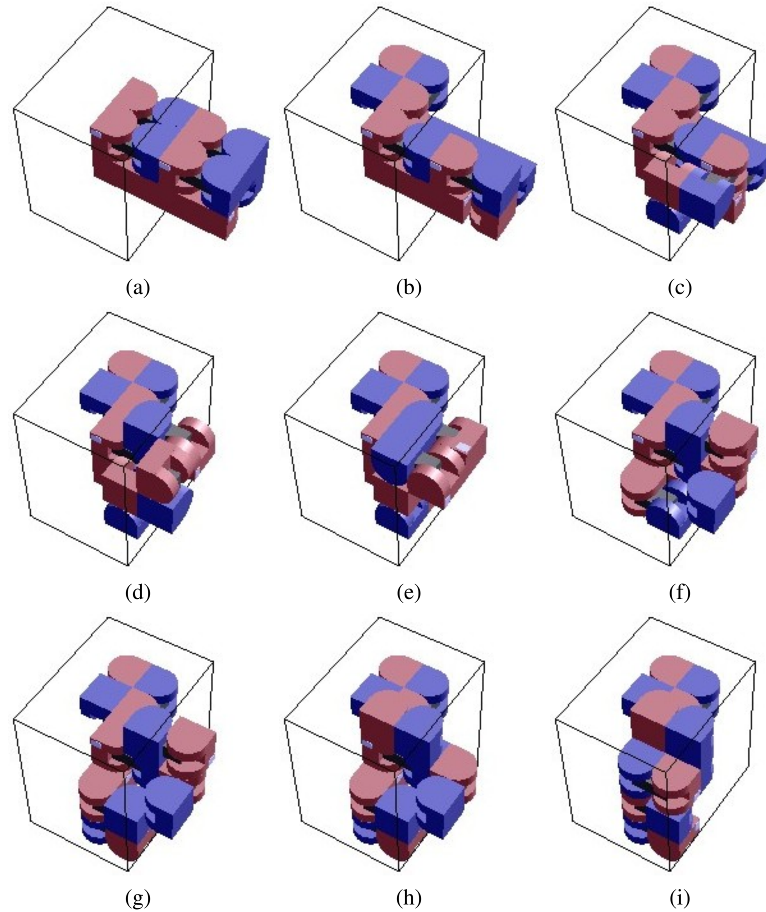

Fig. 7: Eight modules reconfiguring from an initial cuboid configuration into a goal configuration specified by the wire-frame bounding box shown.

4.1 Implementation

We implemented the reconfiguration algorithm in the SRSim simulation environment [7], with SuperBot graphics rendered by a simulation developed at ISI. Collision checking is implemented by testing for intersections between the bounding box surrounding each module part and those surrounding modules in its neighborhood. Configuration space is represented as a 6D grid corresponding to module joint angles. The grid can represent one helper module in addition to the main module. Fig. 6 shows an example of nine modules reconfiguring from a line shape into a box. Fig. 7 shows an example reconfiguration between two cuboid configurations.

5 Discussion and Future Work

We have presented a general framework for reconfiguration and an example implementation for SuperBot-style modules. Because this simple implementation is exponential in the degrees of freedom of the kinematic chain, this planner is suited mainly to lattice-based and hybrid robot types. A planner for chain-based robots (with short chains) could possibly be developed. We have not yet explored the potential in optimizing for quantities other than number of connection/disconnection cycles, but this should be a promising avenue. So far we have been concerned only with finding a feasible reconfiguration plan, but another interesting problem would be to attempt to prove an approximation to optimal reconfiguration. One idea is to build on the lower-bound construction for reconfiguration [18] and attempt to prove an upper-bound on the maximum deviation from shortest path taken by any module in travelling to the goal.

We are currently implementing our algorithm in a decentralized fashion in hardware-in-the-loop simulation [15]. Computation and communication run on embedded processors but actuation is simulated on a desktop computer. It is also our intention to test the algorithm on real robots. One possible platform is a new module we are currently constructing. This module has SuperBot-style kinematics combined with a novel connection mechanism based on grippers or pincers. We would also like to implement and test our algorithm on other module types.

Acknowledgements This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales (NSW) State Government. Many thanks to Surya Singh for lending expertise in robot kinematics. The SuperBot simulator was written by David Brandt under the auspices of ISI.

References

1. Asadpour, M., Sproewitz, A., Billard, A., Dillenbourg, P., Ijspeert, A.J.: Graph signature for self-reconfiguration planning. In: Proc. of the IEEE/RSJ IROS, pp. 863–869 (2008)
2. Barto, A., Mahadevan, S.: Recent advances in hiearchical reinforcement learning. Special Issue on Reinforcement Learning, Discrete Event Systems Journal **13**, 41–77 (2003)
3. Butler, Z., Kotay, K., Rus, D., Tomita, K.: Generic decentralized locomotion control for lattice-based self-reconfigurable robots. Int. J. Rob. Res. **23**(9) (2004)
4. Butler, Z., Rus, D.: Distributed motion planning for modular robots with unit-compressible modules. Int. J. Rob. Res. **22**(9), 699–716 (2003)
5. Chiang, C.H., Chirikjian, G.: Modular robot motion planning using similarity metrics. Autonomous Robots **10**(1), 91–106 (2001)
6. Christensen, D., Stoy, K.: Selecting a meta-module to shape-change the atron self-reconfigurable robot. In: Proc. of IEEE ICRA, pp. 2532 –2538 (2006)
7. Fitch, R.: Heterogeneous self-reconfiguring robotics. Ph.D. thesis, Dartmouth College (2004)
8. Fitch, R., Alempijevic, A., Lal, R.: A self-reconfiguring team of mobile robots. In: Proc. of IEEE ICRA, Workshop on Network Science and Systems in Multi-Robot Autonomy (2010)
9. Fitch, R., Butler, Z.: Million module march: Scalable locomotion for large self-reconfiguring robots. Int. J. Rob. Res. **27**(3-4), 331–343 (2008)
10. Fitch, R., Lal, R.: Experiments with a ZigBee wireless communication system for self-reconfiguring modular robots. In: Proc. of IEEE ICRA, pp. 1947–1952 (2009)

11. Hou, F., Shen, W.M.: Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots. In: Proc. of IEEE ICRA (2008)
12. Hou, F., Shen, W.M.: On the complexity of optimal reconfiguration planning for modular reconfigurable robots. In: Proc. of IEEE ICRA (2010)
13. Itzstein, B.: Assembly order planning for stable self-reconfiguration of modular robots. Undergraduate thesis, The University of Sydney (2009)
14. Kotay, K., Rus, D.: Algorithms for self-reconfiguring molecule motion planning. In: Proc. of the IEEE/RSJ IROS (2000)
15. Lal, R., Fitch, R.: A hardware-in-the-loop simulator for distributed robotics. In: Proc. of ARAA Australasian Conference on Robotics and Automation (ACRA) (2009)
16. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge, U.K. (2006)
17. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving markov decision problems. In: Proc. of UAI, pp. 394–402 (1995)
18. Pamecha, A., Ebert-Uphoff, I., Chirikjian, G.: Useful metrics for modular robot motion planning. IEEE Trans. on Robotics and Automation **13**(4), 531–45 (1997)
19. Prevas, K.C., Unsal, C., Efe, M.O., Khosla, P.K.: A hierarchical motion planning strategy for a uniform self-reconfigurable modular robotic system. In: Proc. of IEEE ICRA (2002)
20. R.Fitch, Rus, D.: Self-reconfiguring robots in the USA. Journal of the Robotics Society of Japan **21**(8), 4–10 (2003)
21. Salemi, B., Moll, M., Shen, W.M.: SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. In: Proc. of IEEE/RSJ IROS (2006)
22. Stoy, K.: Controlling self-reconfiguration using cellular automata and gradients. In: Proceedings of IAS-8 (2004)
23. Stoy, K., Nagpal, R.: Self-reconfiguration using directed growth. In: 7th International Symposium on Distributed Autonomous Robotic Systems (DARS'04) (2004)
24. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
25. Vassilvitskii, S., Yim, M., Suh, J.: A complete, local and parallel reconfiguration algorithm for cube style modular robots. In: Proc. of IEEE ICRA, pp. 117–22 (2002)
26. Yim, M., Shen, W.M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., Chirikjian, G.: Modular self-reconfigurable robot systems. IEEE Robot. Automat. Mag. **14**(1), 43–52 (2007)
27. Yoshida, E., Matura, S., Kamimura, A., Tomita, K., Kurokawa, H., Kokaji, S.: A Self-Reconfigurable Modular Robot: Reconfiguration Planning and Experiments. Int. J. Rob. Res. pp. 903–915 (2002)