Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

Distributed Control of the Center of Mass of a Modular Robot

Mark Moll, Peter Will, Maks Krivokon, and Wei-Min Shen
Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292, USA
Email: moll@isi.edu, will@isi.edu, krivokon@usc.edu, shen@isi.edu

Abstract— **We present a distributed controller for the center of mass of a modular robot. This is useful for locomotion of a modular robot over uneven and unknown terrain. By controlling the center of mass, a robot can prevent itself from falling over. We present a distributed and decentralized algorithm that computes the mass properties of the robot. Additionally, each module also computes the mass properties of the modules that are directly or indirectly connected to each of its connectors. With this information, each module can independently steer the center of mass towards a desired position by adjusting its joint positions. We present simulation results that show the feasibility of the approach.**

I. Introduction

In recent years the area of modular and self-reconfigurable robots has seen much research activity. Much of this work focuses on specific designs, reconfiguration planning, or gait development. There has hardly been any work on locomotion of modular robots in the presence of uncertainty. In a real world environment the surface is often not level and a gait developed on a flat surface may not be as effective or, worse, make the robot fall over. We aim to develop a new approach to locomotion. The key idea is that the robot uses a gait only as a guideline for locomotion, and uses contact information together with mass information to ensure a stable pose at all times. This paper is a first step in this direction. We present a distributed and decentralized algorithm for computing the mass properties in a modular robot. Specifically, the modules compute the total mass, the center of mass (COM), and the inertia tensor. Each module also computes the mass properties of the modules that are directly or indirectly connected to each of its connectors. This information enables a module to compute joint displacements that will move the COM towards a desired position. By extending the results in this paper, we expect that a gait can be specified in terms of where the COM needs to go and which leg needs to be moved, rather than specifying joint angles for every module. This does not only greatly simplify the specification of a gait, but will also allow a modular robot to move over uneven terrain.

The work on modular robots can be divided into hyperredundant kinematic chains and self-reconfigurable robots. With hyperredundant kinematic chains [1–4] the focus is on solving the inverse kinematics or tracking a reference shape with a robot. Solutions to these problems are computed in a centralized way. Our work is more applicable to self-reconfigurable robots, where solutions often need to be computed in a distributed way. Many different types of self-reconfigurable robots have been proposed. A number of recent survey articles [5–8] and one special journal issue [9] provide a good overview of the field. The different types of self-reconfigurable robots can be divided into two categories: lattice-based and chain-based. As the name suggests, in lattice-based systems, the nodes are assumed to be positioned in a grid structure. Module movement is assumed to be discrete, going from one state to an allowable neighboring state. Reconfiguration planning is then 'simply' a discrete search. Some examples of such systems are Molecule modules [10], Crystalline modules [11], bipartite modules [12], ATRON [13] and stochastic cellular robots [14]. In chain-based systems, the modules are typically connected to form tree-like structures or loops. The modules typically have continuous degrees of freedom. Examples of this class of systems include PolyBot [15], and CONRO [16, 17]. Some new hardware modules such as MTRAN [18, 19] and our own SuperBot modules [20] have combined the features from both types. Other hardware such as Tetrobot [21] is built for flexible structures but cannot change connections autonomously.

To the best of our knowledge mass properties of a modular robot have so far not been considered in the literature. In computer graphics the center of mass information has been used to compute realistic poses for articulated figures [22]. This is, however, a centralized approach that is difficult to apply to a distributed system. Despite the apparent lack of work in this area, the advantages of a module knowing the mass properties of the surrounding modules are clear. In addition to facilitate finding stable poses, it can be used to measure the speed of a whole ensemble of modules. This assumes that at any time there exists at least one module that can measure its own speed in the global reference frame. A module in contact with the ground can potentially be used for this purpose. A module can now also anticipate the applied torque to its neighboring modules as its joints move to a given target position.

The rest of this paper is organized as follows. The next section gives an overview of the modular robot on which we have implemented our control methods. However, our approach is not specific to this type of robot, and can be applied to other chain-type modular robots as well. In section III we present a distributed algorithm for computing the mass properties of an ensemble of modules. In section IV this algorithm is then used to actively control the COM. This is done locally; each module independently tries to move the COM to a desired position. Section V describes some of our simulation results. Finally, in

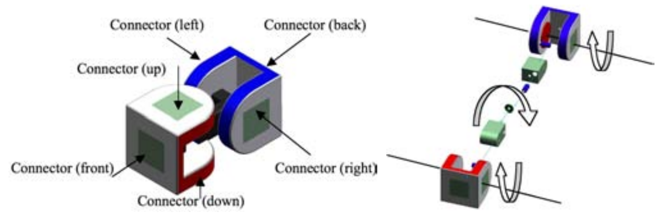


Fig. 1. Diagram of one module.

section VI we summarize the contributions of this paper and outline directions for future research.

II. Background

This section will give a brief overview of the system called SuperBot on which we plan to run our experiments. For a more detailed description see [20]. Each module has three revolute joints as shown in figure 1. Each module also has 6 genderless connectors. Any connector of a module can be connected to any connector of another module in 4 different ways (rotations of 0, 90, 180, and 270 degrees about the connector face normal). By choosing the appropriate rotation for the roll joint (the middle joint in figure 1), a SuperBot module can emulate a Conro module [17] or an MTRAN module [18, 19].

Each module contains two Atmega 128 CPUs, one in each half of the module. The software on the modules is built on top of AvrX, a small real-time kernel for embedded processors [23]. Some of the modules have additional capabilities. Some modules have wireless networking capabilities to enable easy remote control and the bootstrapping of new software. Other modules that are planned are modules with small thrusters to enable flight in micro-gravity environments, and modules with a video camera for simple target tracking.

III. Computing the Mass Properties

In this section we will present a method that will allow each module in a self-reconfigurable system to establish the mass properties of the whole system. Each module computes an estimate of the mass properties based on its own state and on information it receives from its immediate neighbors. Whenever its estimate changes, it sends a message with the new estimate to its neighbors. If the modules do not move, the modules will eventually all converge to the true mass properties and stop sending updates to each other. Below, we will assume that the modules are connected to form a tree-like structure, *i.e.*, there are no loops.

The mass properties are computed with respect to a frame local to each module. Whenever a module sends mass properties to another module, it first transforms them to be relative to a coordinate frame attached to the connector. We assume that the receiving module knows the relative rotation between its connector and the sender's connector. Based on this information and the received mass properties, the receiving module can transform the mass properties to be relative to its local frame.

The algorithm for computing and updating the mass properties is completely distributed and decentralized. The high-level pseudo-code for the main event loop is shown in algorithms 1 and 2. We write the mass properties as a 3-tuple $(m, \boldsymbol{p}, I)$, denoting the mass m, the position of the COM $\boldsymbol{p}$, and the inertia tensor I. Suppose we know the mass properties $c_1 = (m_1, \boldsymbol{p}_1, I_1)$ and $c_2 = (m_2, \boldsymbol{p}_2, I_2)$ of two sets of modules. The combined mass properties c_0 of both sets are then simply

$$c_0 = c_1 + c_2 = \left(m_1 + m_2, \frac{m_1 \boldsymbol{p}_1 + m_2 \boldsymbol{p}_2}{m_1 + m_2}, \frac{m_1 \boldsymbol{I}_1 + m_2 \boldsymbol{I}_2}{m_1 + m_2}\right).$$

Algorithm 1 UpdateMass

```
while true do
    clear update flags for all connectors
    while ¬inbox.empty() do              ▷ process all incoming
        msg = inbox.pop()                ▷ messages
        connectorMass[msg.destination] = msg.mass
        mark other connectors for update
    end while

    RecomputeMass()

    for i = 1 ... n do                   ▷ update neighbors
        if connectorMass[i] is marked for update then
            send connector i (mass − connectorMass[i])
        end if
    end for
end while
```

Algorithm 2 RecomputeMass()

```
recompute moduleMass ▷ the mass properties of just this
                     ▷ module
if moduleMass has changed then
    mark all connectors for update
end if
mass = moduleMass
for i = 1 ... n do
    mass = mass + connectorMass[i]
end for
```

Each module maintains an estimate of the mass properties of the modules attached to each of its n connectors. It also maintains the mass properties of itself and of the whole system. At each step of algorithm 1 a module processes updates from its neighbors, updates its estimate of the global mass properties, and notifies the appropriate neighbors. Note that the information sent to each neighbor i are not the global mass properties, but the mass properties of all modules attached to that neighbor. Basically, each module receives an estimate of the mass properties from a given connector of just the modules that are connected (directly or indirectly) to that connector. These mass properties can be obtained by subtracting the mass properties of all modules connected to connector i from the global estimate. Subtraction of mass properties is defined analogously to addition.

Let d be the largest tree distance between two modules. Then after d iterations of the main loop of algorithm 1, each module will have computed the correct COM, assuming the

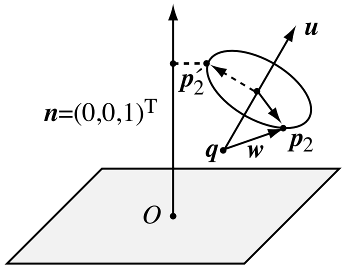


Fig. 2. Moving the center of mass towards a stable configuration reduces to finding the distance between a circle and a line in 3D.

modules do not move. The modules communicate updates only when necessary, so if the modules do not move, the modules will stop sending messages after d iterations.

We can choose to assign a world coordinate frame to a module in contact with the ground. (Electing a unique module if several modules are in contact with the ground is not completely trivial, but can be done through, *e.g.*, a randomized leader election protocol.) This 'leader' module can periodically propagate its position to the other modules. As with the mass properties, this position is transformed to modules' local frames. With this information each module can then also compute the mass properties in world coordinates.

IV. Stabilizing Behavior

As already mentioned in the introduction, one of the uses of the COM is to prevent a self-reconfigurable system from falling over. To stabilize an arrangement of modules we can change the joint angles in the modules, rearrange the modules, or a combination of both. Module rearrangements tend to be slower than changes in joint angles. The effect of rearrangements on the COM is specific to the module architecture. This section will therefore focus only on joint angle changes.

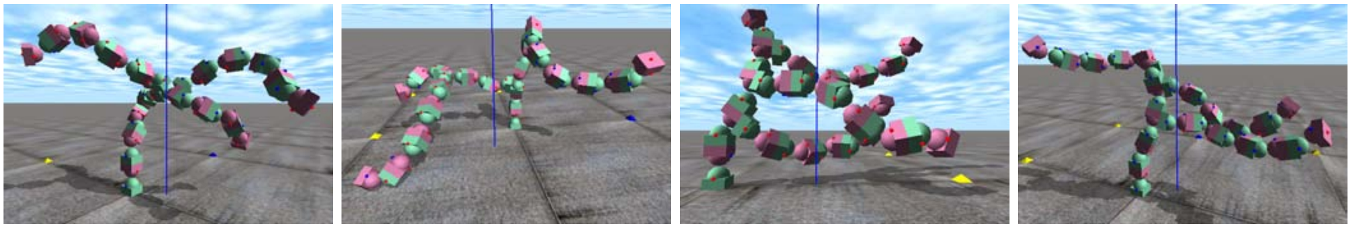
A configuration of modules is stable if the contact forces can balance the gravitational force. Let us consider the simplest case first: there is only one point of contact and there is no friction. A configuration is then stable if its center of mass lies on the *support line:* the vertical line through the point of contact. If a configuration is not stable, then each module can move the configuration closer to stability by adjusting its joint angles. For prismatic joints this is straightforward: for each such joint the COM is translated along a line to a point that is closest to the support line (or as close to that as joint limits allow). Let us now consider a revolute joint. One side of the joint is connected to the contact point and will remain in the same position as the joint angle changes. The other side of the joint and all modules attached to it move along an arc of a circle. Let $\boldsymbol{p}_1$ be the COM of the part of the system that remains fixed, and let $\boldsymbol{p}_2$ be the COM of the part of the system that is going to be rotated. The position of $\boldsymbol{p}_2$ after the joint has been rotated by θ radians about its rotation axis $\boldsymbol{u}$ is denoted by $\boldsymbol{p}_2'$. We can write the new COM $\boldsymbol{p}_0'$ of the total mass as

$$\boldsymbol{p}_0' = \frac{m_1\boldsymbol{p}_1 + m_2\boldsymbol{p}_2'}{m_1+m_2} = \frac{m_1\boldsymbol{p}_1 + m_2(\boldsymbol{q} + R_\theta\boldsymbol{w})}{m_1+m_2},$$

where $\boldsymbol{q}$ is the position of the joint, $\boldsymbol{w} = \boldsymbol{p}_2 - \boldsymbol{q}$, and R_θ is a 3-by-3 rotation matrix representing a rotation of θ radians about $\boldsymbol{u}$. The COM moves along a circle as θ changes. So to move the COM as close as possible to a stable configuration, we need to find the minimum distance between a circle and the support line in 3D. See figure 2. In this figure the support line passes through the origin and is parallel to the z-axis. To minimize this distance, the derivative of the distance written as a function of θ is set to 0 and converted to a fourth degree polynomial. The resulting equation has no stable analytical solution and has to be solved numerically. A fast and accurate numerical method for computing the distance between a circle and a line in 3D is given in [24]. There are a couple of special cases we need to distinguish. First, if the COM $\boldsymbol{p}_2$ lies on the axis of rotation, then no rotation will bring the COM closer to the support line. In this case the joint maintains its current position. The second special case occurs when the axis of rotation and the support line coincide. In this case the joint cannot bring the COM closer to the support line either and it will maintain its current position. Finally, if the center of rotation lies on the support line and the axis of rotation does *not* coincide with the support line, then there exist two solutions. In the current implementation we pick the solution with the smallest z-coordinate. Lowering the height of the COM tends to improve stability, but it can also restrict maneuverability of the modules.

Up to now we have only considered one joint at a time. Ideally, all joints move simultaneously towards a stable configuration. Finding optimal displacements for all joints simultaneously is very difficult, especially if we want to find a solution in a distributed fashion. We would have to solve the inverse kinematics or use a path planner to find a path between the current configuration and a stable configuration. Efficiently solving these problems in parallel and efficiently is not practical on simple embedded processors such as those we plan to use on our modules. Instead, we use an approximate solution which tends to converge to a desired configuration very quickly. Each joint computes its own optimal displacement independently of each other. This displacement is passed on to the joint's PID controller. The optimal displacement is recomputed frequently. This computation is interleaved with the computation of the COM (which continuously changes as the joints move towards their target positions). In effect, the modules make small steps towards a stable configuration. Both the absolute and relative position of the COM keeps changing, so the direction of these steps is recomputed at every step.

The approach described above computes instantaneously a desired direction to move in for all modules, but ignores momentum that builds up in the whole ensemble of modules. The PID gains need to be adjusted so that the modules do not oscillate around the support line. In the future we plan to study and develop controller that take full advantage of the kinematic



(a) (b) (c) (d)

Fig. 3. Four random trees.

information that the nodes can share, but in this initial study we just compare some simple heuristics that qualitatively achieve the desired behavior with the default behavior (*i.e.*, when the controllers on each module are using identical gains). We will describe below two heuristics. One is based on the distance between the estimated COM and the support line, the other is based on momentum.

Since the goal of the stabilizing behavior is to get the COM as close as possible to the support line, it makes sense to reduce the gains as the COM gets closer, so that the robot does not overshoot the goal position. After the first few steps of Algorithm 1, the modules will have a reasonably accurate estimate of the COM. The distance of the COM to the support line is easy to compute, and is almost the same on every module. The proportional gain is adjusted as follows:

$$K_P = (c_0 + c_1 d_{\text{support}}) K_{P0},$$

where c_0 and c_1 are constants, d_{support} is the distance to the support line, and K_{P0} is the nominal proportional gain. The integral and derivative gain are adjusted analogously. The constants c_0 and c_1 are determined through simulations (see next section).

The momentum based heuristic is based on the notion that an ensemble of modules should not gain too much momentum, as it will be difficult to stop the modules once the goal is reached. For each joint we consider the mass and distance to the joint of the COM of the modules that will be moved by this joint. The gains are scaled by the magnitude of $\boldsymbol{v} = m_2 \boldsymbol{u} \times (\boldsymbol{p}_2 - \boldsymbol{q})$. So the proportional gain is adjusted as follows:

$$K_P = (c_0 + c_1 \|v\|) K_{P0}.$$

As with the distance heuristic, the integral and derivative gain are adjusted analogously.

The approach described above can be generalized to multiple contacts with friction as follows. With friction the COM needs to be inside a *stability region* (instead of on a support line). This region is determined by the contact forces, which have to lie inside the friction cones at the contacts. In recent work [25, 26] centralized algorithms are presented to quickly compute such regions. Modules need to pass on contact information, so that each module can compute the stability region. With multiple contacts we aim to make the modules move the COM within this region. Part of a gait specification can then consist of moving the COM to a sequence of different control points, (or line segments, facets, etc.). One problem with multiple contacts is that the robot effectively forms a closed chain with the ground. The modules need to coordinate their actions more closely, and some compliance will be necessary. This is a very challenging problem that we plan to explore in future research.

V. Simulation Results

The balancing behavior was tested in our simulation environment. This environment is built on top of the Open Dynamics Engine (ODE) [27]. The specifications of the simulated modules are based on the real hardware modules that are currently being built. The kinematics of the modules has already been described in section II. The only difference is that the simulated modules have only 10% of the actual mass. This is done to simulate the behavior with a large number of modules. In this case the motors would in practice not be able to lift long chains. Future versions of the module are planned to be lighter. By combining this with improved balancing behavior, we plan to bridge the gap between simulation and reality.

To test the balancing behavior we generated random trees of modules. The results in this section are based on random trees consisting of 20 modules divided into 4 branches of 5 modules. Since each module has three degrees of freedom, the whole tree has 60 degrees of freedom. The root is always in a vertical orientation, and has been fixed to the ground. The tests were performed on the four trees shown in figure 3. To evaluate the performance, we computed the distance between the center of mass and the support line as a function of time. We would like this distance to converge to 0 as quickly as possible. We did this for the three different control schemes discussed in the previous section: the default scheme where the gains on all modules are identical and constant, the scheme where the gains depend on the estimated distance to the support line, and the scheme that uses the momentum heuristic. For each scheme, the controller computes the desired torque 1,000 times per second. The results of our simulations are shown in figure 4. The graphs match the corresponding tree in figure 3.

The nominal proportional, integral and derivative gains are set to 10, 0.1, and 1e-4, respectively. For the distance heuristic, the constants c_0 and c_1 are set to 0 and 0.1, whereas for the momentum heuristic these constants are set to 0 and 1. The gains and constants were chosen 'by hand' to get a reasonable performance on a series of tests, but were not heavily optimized for the examples shown here. From the graphs we can conclude that no control method consistently outperforms the others. The

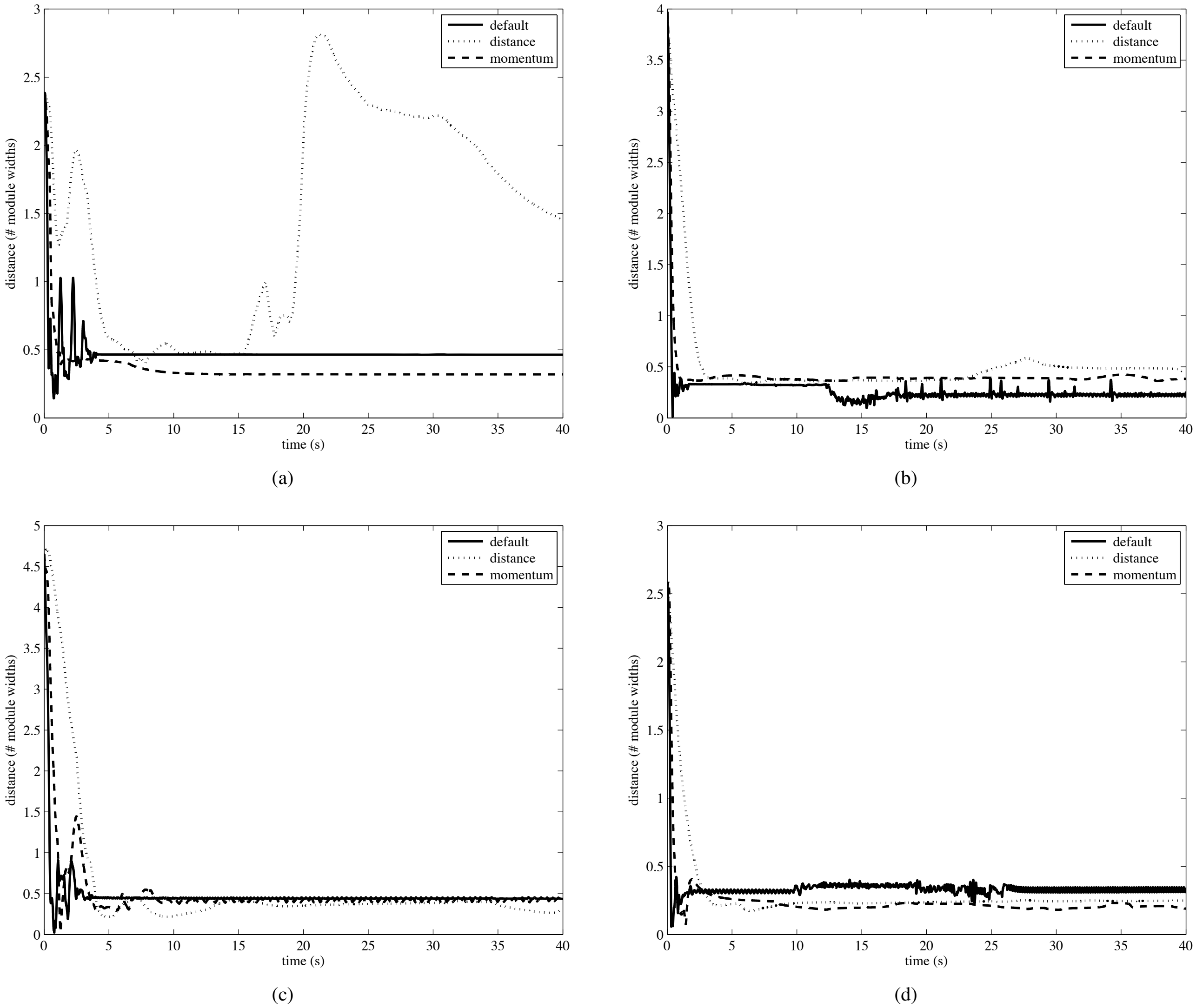


Fig. 4. Stabilizing behavior with three different control laws. The distance is measured in multiples of the width of a module (see figure 3).

momentum heuristic seems to give the best overall behavior in the sense that it always converges quickly and tends to have a small steady-state error. The other two methods will sometimes converge to a smaller distance faster, as is shown in figure 4(b). In general, though, all methods exhibit the desired behavior most of the time. Clearly, more work is still needed to improve the performance in terms of reliability and convergence rate.

VI. Discussion

In this paper we have shown the feasibility of using distributed control to move the COM of a modular robot to a desired position. The control programs on the modules are distributed and decentralized. By exchanging information with the modules it is connected to, each module can accurately track the mass properties of the whole ensemble as the modules are moving around. Moreover, each module can move the COM to its desired position independently of what the other modules are doing. The control method presented in this paper is a simple, greedy, local method. In our simulations the basic method of controlling the COM, in combination with a heuristic to improve stability, has been shown to successfully move the COM to a desired position.

There are many different directions in which we plan to expand this work. First, the performance can be improved if each module computes the optimal joint angles for all three joints simultaneously (rather than one-by-one). With three degrees of freedom we can move the COM along a straight line to the desired position (but without controlling the orientation of the associated mass). If we can automatically form *pairs* of modules in an ensemble of modules, then each pair has six degrees of freedom and can control both position and orientation of the COM. Of course, in practice joint limits and (self-)collisions may make impossible movement in some directions.

The second direction for future research is to use the inertia tensor in the balancing behavior (in addition to the mass and the COM). Using the (approximate) inertia tensor, each module can predict more accurately the system response to a torque applied at one its joints. It may also be useful to compute simple shape descriptors of the whole ensemble of modules. The shape information does not necessarily lead to better balancing behavior, but can be used to modify the posture of a modular robot.

When a modular robot is arranged in a multi-legged formation, the robot effectively forms closed kinematic chains with the ground. This means that modules will have to collaborate more closely to avoid jamming the joints. The joints can be partitioned in active and compliant joints, but care should be taken that the robot does not fall over.

Finally, we plan to take external forces such as gravity and friction at the contact points into consideration. By trying to compensate for those forces, we aim to maintain more or less the same posture while at the same time move the COM to its desired position.

ACKNOWLEDGEMENT

This research is supported in part by US Army Research Office under the grants W911NF-04-1-0317 and W911NF-05-1-0134, and in part by NASA's Cooperative Agreement NNA05CS38A. We thank Alliance Spacesystems Inc. and USC Engineering Machine Shop for the fabrication of prototype SuperBot modules, and we are also grateful for Professors Berok Khoshnevis, Yigal Arens, and other members in our Polymorphic Robotics Laboratory for their intellectual and moral support.

REFERENCES

[1] K. E. Zanganeh and J. Angeles, "The inverse kinematics of hyper-redundant manipulators using splines," in *Proc. 1995 IEEE Intl. Conf. on Robotics and Automation*, 1995, pp. 2797–2802.

[2] G. S. Chirikjian and J. W. Burdick, "The kinematics of hyper-redundant robot locomotion," *IEEE Trans. on Robotics and Automation*, vol. 11, no. 6, pp. 781–793, Dec. 1995.

[3] H. Mochiyama, E. Shimemura, and H. Kobayashi, "Shape control of manipulators with hyper degrees of freedom," *Intl. J. of Robotics Research*, vol. 18, no. 6, pp. 584–600, June 1999.

[4] I. A. Gravagne, "Asymptotic regulation of a one-section continuum manipulator," in *Proc. 2003 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Las Vegas, NV, 2003, pp. 2779–2784.

[5] D. Rus, Z. Butler, K. Kotay, and M. Vona, "Self-reconfiguring robots," *Communications of the ACM*, vol. 45, no. 3, pp. 39–45, Mar. 2002.

[6] M. Yim, Y. Zhang, and D. Duff, "Modular robots," *IEEE Spectrum*, vol. 39, no. 2, pp. 30–34, 2002.

[7] P. Jantapremjit and D. Austin, "Design of a modular self-reconfigurable robot," in *Proc. 2001 Australian Conf. on Robotics and Automation*, Sydney, Australia, Nov. 2001, pp. 38–43.

[8] D. Mackenzie, "Shape shifters tread a daunting path toward reality," *Science*, vol. 301, no. 5634, pp. 754–756, Aug. 2003.

[9] W.-M. Shen and M. Yim, "Self-reconfigurable modular robots, guest editorial," *IEEE/ASME Trans. on Mechatronics*, vol. 7, no. 4, pp. 401–402, 2002. [Online]. Available: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=25977

[10] K. Kotay and D. Rus, "Locomotion versatility through self-reconfiguration," *Robotics and Autonomous Systems*, vol. 26, pp. 217–232, 1999.

[11] D. Rus and M. A. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Autonomous Robots*, vol. 10, pp. 107–124, 2001.

[12] C. Ünsal, H. Kiliççote, and P. K. Khosla, "A modular self-reconfigurable bipartite robotic system: Implementation and motion planning," *Autonomous Robots*, vol. 10, pp. 67–82, 2001.

[13] M. W. Jørgensen, E. H. Østergard, and H. H. Lund, "Modular ATRON: Modules for a self-reconfigurable robot," in *Proc. 2004 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2004, pp. 2068–2073.

[14] P. J. White, K. Kopanski, and H. Lipson, "Stochastic self-reconfigurable cellular robotics," in *Proc. 2004 IEEE Intl. Conf. on Robotics and Automation*, 2004, pp. 2888–2893.

[15] M. Yim, D. G. Duff, and K. D. Roufas, "PolyBot: a modular reconfigurable robot," in *Proc. 2000 IEEE Intl. Conf. on Robotics and Automation*, 2000, pp. 514–520.

[16] W.-M. Shen, B. Salemi, and P. Will, "Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots," *IEEE Trans. on Robotics and Automation*, vol. 18, no. 5, pp. 700–712, Oct. 2002.

[17] 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, June 2000.

[18] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata, "M-TRAN II: metamorphosis from a four-legged walker to a caterpillar," in *Proc. 2003 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2003, pp. 2454–2459.

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

[20] B. Salemi, M. Moll, and W.-M. Shen, "SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system," in *Proc. 2006 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Beijing, China, Oct. 2006.

[21] G. J. Hamlin and A. C. Sanderson, *Tetrobot: A Modular Approach to Reconfigurable Parallel Robotics*, ser. The International Series in Engineering and Computer Science. Springer Verlag, 1998, vol. 423.

[22] R. Boulic, R. Mas, and D. Thalmann, "A robust approach for the center of mass position control with inverse kinetics," *Journal of Computers and Graphics*, vol. 20, no. 5, 1996.

[23] L. Barello, "AvrX real time kernel." [Online]. Available: http://www.barello.net/avrx/

[24] D. Vranek, "Fast and accurate circle-circle and circle-line 3D distance computation," *Journal of Graphics Tools*, vol. 7, no. 1, pp. 23–32, 2002.

[25] Y. Or and E. Rimon, "Computing 3-legged equilibrium stances in three-dimensional gravitational environments," in *Proc. 2006 IEEE Intl. Conf. on Robotics and Automation*, 2006, pp. 1984–1989.

[26] T. Bretl and S. Lall, "A fast and adaptive test of static equilibrium for legged robots," in *Proc. 2006 IEEE Intl. Conf. on Robotics and Automation*, 2006, pp. 1109–1116.

[27] R. Smith, "Open dynamics engine." [Online]. Available: http://ode.org/