Distributed Robotic Manipulation: Experiments in Minimalism

Karl Böhringer, Russell Brown,
Bruce Donald, Jim Jennings
Cornell University
Ithaca, NY 14853, USA
brd@cs.cornell.edu

Daniela Rus
Dartmouth College
Hanover, NH 03755, USA
rus@cs.dartmouth.edu

This paper describes research done in the Robotics and Vision Laboratory at Cornell University. Support for our robotics research was provided in part by the National Science Foundation under grants No. IRI-8802390, IRI-9000532, IRI-9201699, by a Presidential Young Investigator award to Bruce Donald, by an NSF/ARPA S.G.E.R. in MEMS, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation, and AT&T Bell Laboratories.

Abstract

Minimalism pursues the following agenda: For a given robotics task, find the minimal configuration of resources required to solve the task. Thus, minimalism attempts to reduce the resource signature for a task, in the same way that (say) Stealth technology decreases the radar signature of an aircraft. Minimalism is interesting because doing task A without resource B proves that B is somehow inessential to the information structure of the task. We will present experimental demonstrations and show how they relate to our theoretical proofs of minimalist systems.

In robotics, minimalism has become increasingly influential. Marc Raibert showed that walking and running machines could be built without static stability. Erdmann and Mason showed how to do dextrous manipulation without sensing. Tad McGeer built a biped, kneed walker without sensors, computers, or actuators. Rod Brooks has developed online algorithms that rely less extensively on planning and world-models. Canny and Goldberg have demonstrated robot systems of minimal complexity. We have taken a minimalist approach to distributed manipulation. First, we describe how we built distributed systems in which a team of mobots cooperate in manipulation tasks without explicit communication.[1] Second, we are now building arrays of micromanipulators to perform sensorless micromanipulation. We describe how well our experimental designs worked, and how our manipulation experiments mirrored the theory.

1. Introduction

This paper describes our experience in building distributed systems of robots that perform manipulation tasks. We have worked at both the macroscopic and the microscopic scale. First, we describe a team of small autonomous mobile robots that cooperate to move large objects (such as couches). The robots run SPMD[2] and MPMD[2] manipulation protocols with no explicit communication. We developed these protocols by distributing offline, sequential algorithms requiring geometric models and planning. The resulting parallel protocols are more on-line, have reduced dependence on *a priori* geometric models, and are typically robust (resistant to uncertainty in control, sensing, and initial conditions).

Next, we discuss our work on sensorless manipulation using massively parallel arrays of microfabricated actuators. The single-crystal silicon fabrication process opens the door to building monolithic microelectromechanical systems (MEMS) with microactuators and control circuitry integrated on the same chip. Our actuators are servoed to uniquely orient (up to symmetries) an object lying on top, and require no sensing. We can also program the array as a sensorless geometric filter—to sort parts based on shape or size.

We developed both the macroscopic and the microscopic systems by distributing and parallelizing sequential manipulation algorithms with global control, to obtain distributed algorithms running on independent physical agents. Our MEMS control algorithms for micromanipulation are SPMD; for the macroscopic (furniture-moving) task, we describe implementations and experiments with both SPMD and MPMD control.

[1] No RF or IR messages are sent between the robots.

[2] SPMD (MPMD) = *Single (Multiple) Program, Multiple Data.*

We have implemented and extensively tested our macroscopic distributed manipulation strategies. We have built MEMS prototypes, and we are now fabricating and testing our biggest micro-array yet (the entire wafer is tiled with 7000 microactuators). Our macroscopic algorithms use no direct communication between the agents, but do employ sensing. Our microscopic algorithms are sensorless, but require a small amount of local communication to initialize and synchronize. Our theory predicts a trade-off between communication and sensing when we parallelize a manipulation strategy. We will discuss experiments we have performed to experimentally observe and validate these trade-offs.

2. Reorienting Large Objects with Autonomous Mobile Robots

We are interested in large-scale manipulation of objects by small mobile robots. In Sections 2 and 3 of this paper, the manipulated objects have comparable size and dynamic complexity to the robots. Objects used in our experiments are up to 6 times the robot's diameter in length, and up to twice the mass of one of the robots. Repositioning and reorientation of these objects may be possible only through active cooperation of a team of mobile robots; for other objects, employing multiple robots may yield performance or other benefits, such as ease of programming.

Consider the task whose goal is to change the orientation of a large object by a given amount. This is called the *reorientation* task. We have described and analyzed in detail the reorientation task in [16]. Figure 1 depicts one robot reorienting a large object. A robot can generate a rotation by applying a force that is displaced from the center of friction. This property relates the dynamics and the geometry or reorientations [12] and it can be used to effect continuous reorientations with mobile robots. The idea is to compliantly apply a sliding force on the face of the object[3]. We call this action a *pushing-tracking* step. When the end of the face is reached, the robot may turn around to reacquire contact and repeat the pushing-tracking. A robot that has gone past the end of a face effectively losing contact with the object has *broken contact* with the object. A robot whose maximum applied force (defined by a threshold) does not change the pose of the object has encountered an *impediment.*

One robot may effect any desired reorientation by repeated pushing-tracking steps if it can apply a large enough force, but it may require a large workspace area for the rotation. We are interested in strategies for reorienting objects *in place*. This can be done by a team of k robots[4]. (See Figures 1(b), 2, and 5.) The k robots can simultaneously constrain the motion of the object and execute *pushing-tracking* operations. We can measure the degree of parallelism in the reorientation algorithm by counting how many of the robots are *active*, *i.e.*, that execute push-tracking motions, and how many of the robots are *stationary*, *i.e.*, that constrain the motion of the object by staying fixed in place. We show how to select the active and stationary robots and how to decide on role-switching over the course of an algorithm.

We now present four different but "equivalent" reorientation protocols that have different degrees of parallelism, synchrony, and resource utilization. Our notion of

[3]This strategy can be implemented by a force that forms an acute angle on the contacting edge. This is similar to hybrid control [14] which would be used for a dexterous hand [17].

[4]For an excellent survey of cooperative mobile robotics see [4].

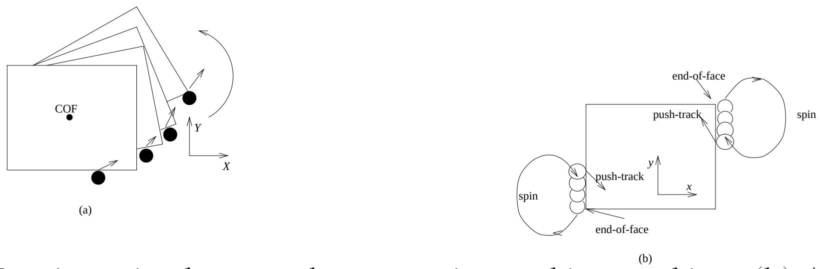

Figure 1. (a): Reorientation by one robot executing pushing-tracking. (b) A system of two robots reorienting a couch. The robot motions are shown in an object-fixed frame. Each robot executes a pushing-tracking motion. Robots recognize when they reach the end of a face by breaking contact, and execute a spinning motion to turn around and reacquire contact.

"equivalence" comes from looking at the task as a dynamical system. Consider the configuration space $\mathcal{C}$ of the manipulated couch. We call two manipulation protocols *equivalent* if the protocols have the same forward-attracting compact limit set in $\mathcal{C}$ [7] (p. 284) and [6]. All of our reorientation protocols rely on the ability of robots to execute *push-tracking* motions.

2.1. An Off-line, Global Control Protocol

The off-line global control strategy denoted by GLOBAL-OFFLINE requires three robots and is described and analyzed in detail in [17]. The algorithm consists of a sequence of pushing-tracking steps, where one robot is active and two robots are stationary at all times (see Figure 2). A global controller sends and receives control signals to the robots. Under the assumption that the robots are already in contact with the object, the algorithm can be summarized as follows:

- Active robot:
  1. push-track until position or force termination
  2. signal the global controller
  3. become stationary
- Stationary robots:
  1. sense for relative motion (*slip*)
  2. if no slip, signal the global controller
  3. when signal, become active

Figure 2. A two-step pushing-tracking sequence of snapshots for reorienting an object with three robots. The right sequence denotes three snapshots of the first step. The left sequence denotes three snapshots of the second step. The black circles denote stationary robots. The white circles denote active robots.

A planner [17] takes as input the geometric model of the object we wish to manipulate. It outputs the initial contact locations for the robots and the order in which the robots become active, which is called the push-tracking *schedule*. Moreover, the termination of each each step is characterized by "jamming" the object between the three robots and thus can be detected by an active slip sensor. The planner guarantees that *no robot will break contact* in a multi-step push-tracking reorientation protocol.

The output of the planner is used for the *setup* phase, in which the robots make contact with the object in the areas generated by the planner, by using motion plans generated by a trajectory planner. The output of the planner is also used by the global controller to signal the robots when to become active.

The planning system diagram, the execution system diagram and the setup procedure are depicted in Figures 3(a) and 4(a).

A system of robots executing this algorithm requires the following *skills* of each robot:

- `(goto` ⟨*position*⟩`)` to contact the object during the setup phase in the areas computed by the planner. `(goto` ⟨*position*⟩`)` can be implemented by using a trajectory generator and a localization system, or even dead-reckoning [3].

- `(push-track)` to control the pushing-tracking motion of each active robot. This entails exerting a force in the normal direction to the face of the object while commanding a velocity in the tangential direction (see Figures 2 and 1.) This can be implemented with a hybrid controller [14] which servoes on the orientation of a face of the manipulated object.

- `(active-slip?)` used by the stationary robots to detect slip at their points of contact. The value of this predicate is passed to the global controller which evaluates the termination predicate.

2.2. An Off-line, Local Control Protocol

A variant of this protocol denoted by LOCAL-OFFLINE can be derived for three (or more) independent robots. This system of autonomous robots cooperates to complete a reorientation and does not have a global controller. Instead, the robots use communication (IR or RF) to synchronize their actions.

Three robots suffice in this protocol. At every point in time, there is one active robot and two stationary robots. Under the assumption that the robots begin in contact with the object and each robot knows the push-tracking schedule, the algorithm can be summarized as follows:

- Active robot:

    1. push-track until position or force termination
    2. signal the next robot in the schedule
    3. become stationary

- Stationary robots:

    1. sense for relative motion (*slip*)

2. if no slip, signal the active robot
3. when signalled, become active

The same planner as in the algorithm GLOBAL-OFFLINE (Section 2.1) is used to plan the setup phase and to determine the push-tracking schedule (see Figure 3(b)). The schedule is used for communication and it can be represented as a directed graph. The nodes of this graph are robots and the edges are communication paths as in Figure 4(b). This graph is a directed Hamiltonian cycle. The planner can be thought of as giving the topology of the communication graph. A token-passing mechanism can be used to achieve the necessary communication. Note that in this scheme, the active robot receives messages from one robot only. Since all the stationary robots can sense the "no slip" condition approximately simultaneously, the active robot will receive the signal.

Consider the skills necessary for a team of robots to execute this algorithm. In addition to the skills of GLOBAL-OFFLINE, the robots need the ability to communicate turn-taking conditions to each other. Another difference is the implementation of push-tracking and of the sensor for the jamming condition. The key issue is that an autonomous mobile robot rarely has the sophistication required for hybrid control or for active slip sensing. Additional complications are introduced by the non-holonomic constraints imposed by the wheeled kinematics of our RWI bases. From a control point of view, we wish to implement the following abstraction on top of the non-holonomic, torque-controlled wheeled bases: we wish to program the pushing robots as if they were *holonomic* moving kinematic constraints, subject to sliding friction during contact. Here is how we do this.

We implement push-tracking using the following steps: (i) Choosing a suitable velocity direction v specified as a relative angle to the surface of the manipulated object. (ii) Aligning to the object face using the relative angle between the robot and the object (which our robots can measure directly using a ring of contact-sensors[10].) (iii) The robot moves in the direction v thereby applying a force to the object. That is, after step (ii), the robot's current heading is v. In step (iii) the robot applies a force in the direction of the current heading, and sliding is permitted at the contact point(s). Thus, two skills are required of each robot: (I) the ability to measure the relative orientation at the point(s) of contact and select a heading direction; and (II), the ability to apply a compliant pushing force given a heading direction (as described above). Skill (I) is implemented by our virtual orientation sensor `(align)` described below. Skill (II) is implemented by a pushing primitive called `(prim-push)` described below. The skill `(push-track)` can then be described as the sequential composition of `(prim-push)` and `(align)`. In the language of information invariants [7], this implies the equation[5]

$$\texttt{(push-track)} =_0 \texttt{(prim-push)} + \texttt{(align)}. \tag{1}$$

We implemented the sensor for the termination of a pushing-tracking step (the "jamming" condition) by allowing the robots to detect impediment conditions thresholded by force [15, 9]. In summary, the following skills are required of each robot involved in this protocol:

- `(goto` $\langle position \rangle$`)`

[5] $A =_0 B$ when $A \leq_0 B$ and $B \leq_0 A$.

- `(push-track)` Recall that by Equation(1) `(push-track)` is composed of two skills:

  I . `(align)` Each robot has a relative orientation sensor that allows it to align [16] itself at a specified angle to the face of the object.

  II . `(prim-push)` Given a heading direction, each robot has the ability to compliantly exert a pushing force.

- `(blocked?)` The active robots must be able to detect the "jamming" condition. This is implemented in our system by sensing *impediment* conditions.

- `(pass-token)` Active robots have the ability to pass the pushing-tracking control to stationary robots.

- `(receive-token)` Stationary robots have the ability to receive a signal that makes them active.

2.3. An On-line, Synchronous Protocol

The two previous protocols require a planner. We now ask: do reorientation protocols depend crucially on planning? We present another version of the algorithm (which we denote by SYNCH-ONLINE) that does not use a geometric model for the object being manipulated and does not necessitate a planner.

Three robots suffice for this algorithm. At any point in time there is one active robot and two stationary robots. Under the assumption that the robots begin in contact with the object, we have:

- Active robot:

  1. push-track until sensing contact break or impediment
  2. if contact break than *spin*
  3. else if impediment then signal a stationary robot and become stationary

- Stationary robots:

  1. wait for a signal
  2. if contact break than guarded-move to contact
  3. when token received, become active

In this algorithm, *spin* is a motion that is executed when an active robot has gone past the end of the face. In this case, the robot turns around and reacquires contact by traveling around a circular trajectory (see Figure 1(b)).

Let us determine precisely what is the information provided by the planner and consider removing it. Without a reorientation planner, we note the following:

1. There are no initial contact regions for the robots. However, in the setup phase, the robots can acquire contact by executing guarded moves in directions that are random but towards the center of the object. The robots can distribute themselves randomly around a circle about the object in the same manner as any formation algorithm. (For example, we can use a potential field with a large repeller at any point inside the object and small repellers on each robot.) Hence we must add randomization, but only in the setup phase.

2. There is no particular sequencing of push-tracking steps (as opposed to having a fixed sequence computed by the planner), thus there is no particular communication graph for the reorientation phase. As a result, there can be no guarantee that the robots maintain contact at all times. While maintaining contact is important for fine manipulation within a dexterous hand, it is not necessary for the task of large-scale manipulation with mobile robots. Therefore, we are willing to give up maintaining contact in favor of an online algorithm without a planner and without a world model. We cope with robots breaking contact in the following way. When a stationary robot breaks contact it executes a `(guarded-move)` (forward) to reacquire contact. When an active robot breaks contact, it has effectively gone past the end of a face. This condition, which we label as `(end-of-face?)` (see Figure 2) can be monitored by using a robot's contact bumpers. When a robot goes past the end of a face, it reacquires contact by `spin` (see Figure 1(b)). In `(spin)` the rate and radius of rotation can be chosen so that in our scale and speed regime the robot is guaranteed to reacquire contact.

An additional problem may occur. Without a planner's Hamiltonian cycle the agents may immobilize the object in a configuration for which no pushing action by any agent will result in motion. This condition must be detected. When it is detected, the robots randomize to pick a new configuration around the object. To randomize, every robot executes a `(spin)`.

Figure 3(c) describes the planning system, Figure 4(c) describes the communication graph for this protocol. The following skills are required of each robot involved in this protocol:

- `(pass-token`$\langle token \rangle$`)`, `(receive-token)`, `(blocked?)`, and

  `(push-track)` $=_0$ `(prim-push)` $+$ `(align)`
- `(guarded-move` $\langle direction \rangle$`)` to reacquire contact
- `(end-of-face?)` The active robot must be able to detect when sliding past the end of the face.
- `(spin)` Each robot must be able to reacquire contact when the end of the face is reached.

2.4. An On-line, Uniform, Asynchronous Protocol

The three previous protocols require explicit communication. We now ask: how essential is the explicit communication for reorientation protocols? We present a protocol (which we denote by ASYNCH-ONLINE) that is *uniform* (SPMD), in that the robots execute the same program asynchronously and in parallel and there is no explicit communication between the robots.

For this protocol, two robots suffice. All the robots are active all the time. Assuming that the robots are in contact with the object, the following algorithm achieves the reorientation.

- Active Robots:

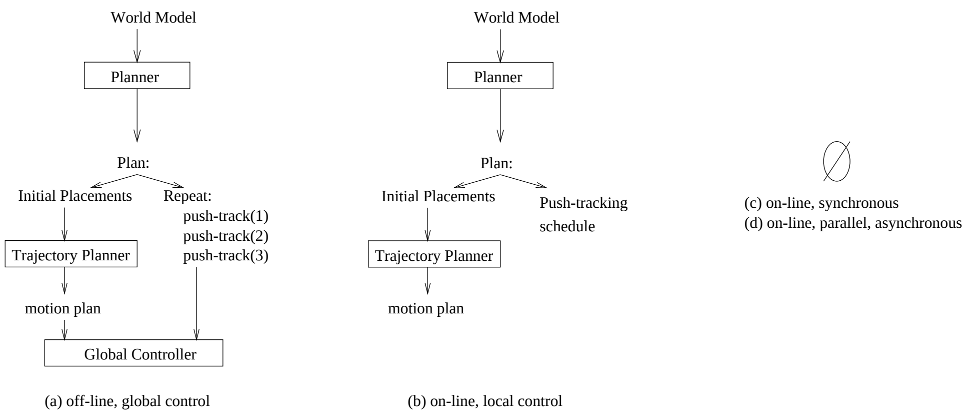

Figure 3. Planning System Diagram. Data flow, information processing, use of world models, and output of the planners. Arrows show I/O structure (data input and output) to and from computational processes. Note that Planner (b) can be obtained from Planner (a) by filtering the push-tracking *schedule*.

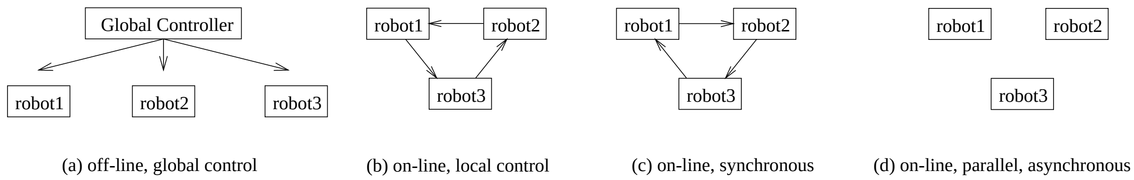

Figure 4. Execution System Diagram and Communication Graph. Arrows show messages passed using *direct* communication. Note that the communication graph in (b) is determined by the pushing schedule, while the communication graph in (c) is any randomly selected Hamiltonian cycle.

1. push-track until contact break or impediment
2. if contact break then spin
3. if impediment then graze

The intuition is that the robots try to maintain the push-tracking state. When the end of the face is reached, the *spinning* motion is employed to reacquire contact. When an impediment is encountered, the robot executes a guarded move near-parallel to, but towards the face of the object, effectively *grazing* the object. Graze terminates when the robot recontacts the object or when it detects that it has

Figure 5. Two mobile robots cooperating to reorient a couch: a snapshot taken from a couch reorientation experiment.

traveled past the end of the face. Hence, graze terminates in (I) reacquiring the object at a contact with a longer moment arm, or (II) missing altogether. (I) is detected with a guarded move. (II) is detected using a sonar-based wall/corner detection algorithm of [10]. When (II) occurs, the robot executes `(spin)`.

In the setup phase the robots contact the object using the same procedure as in protocol SYNCH-ONLINE. The robots may reach the object at different times. As soon as a robot makes contact with the object, it proceeds with executing a push-tracking motion.

Figures 3(d) and 4(d) describe the robot activities in this protocol. Note that there is no explicit communication. The following skills are required of the robots:

- `(guarded-move` $\langle$*direction*$\rangle$ `)`, `(blocked?)`, `(end-of-face?)`, `(spin)`, and `(push-track)` $=_0$ `(prim-push)` + `(align)`,
- `(graze)` Each robot must be able to translate near-parallel to the face of an object to find a new contacting point. Graze uses sonar.

We have implemented a system to move furniture in our lab that includes the asynchronous online reorientation protocol. Our experiments suggest that the system is very robust.

3. MPMD Manipulation

Section 2 presents an implemented and tested SPMD manipulation protocol. We now present an MPMD protocol called the *Pusher/Steerer system*. A detailed description and analysis of this sytem is given in [2, 3].

Despite conventional wisdom regarding the complexities of programming a multi-robot system, a key feature of the Pusher/Steerer system is its ease of use – the actual robot code is simple and elegant, and yet there remains great flexibility in methods of path specification. We present the following properties of the Pusher/Steerer system:

- The code that implements the Pusher/Steerer strategy is simple because it relies on the "natural" kinematic and dynamic interactions between the robots and the manipulated object to achieve the goal. Clearly, an understanding of the relevant dynamics is fundamental to research in manipulation, whether the manipulation is performed by mobile robots or other devices.
- The Pusher/Steerer system is easily adapted to employ either an offline path planner, or an online navigation system in which the path to the goal is not known *a priori*. This is possible because of the decoupling of the steering and pushing components of the strategy.
- Finally, an information invariant analysis (see [7]) of the Pusher/Steerer strategy reveals several formal properties, which we may express informally here:

    1. The Pusher/Steerer system is exactly a redistribution of the same resources (computation, state, sensors, etc.) of a comparable single-robot manipulation system.
    2. There is no explicit communication between the Pusher and the Steerer.

3. The addition of a clock and some state to each robot (the Pusher and the Steerer) increases the power of the system significantly. With clocks, the Pusher and Steerer may exchange roles in an online fashion, and thus execute complex paths, such as "parallel parking" maneuvers.

3.1. Details of Pusher/Steerer

We are using cooperative manipulation protocols in which two robots move a box (or other object) using what we call a Pusher/Steerer configuration, in which the robots take on the role either of the Pusher, in which

- Torque-controlled translations push the object in front of the robot,
- the robot follows the object by continually turning to align its front bumpers with the rear face of the object (the rotational and translational motions here are decoupled and occur in parallel), and
- the robot does *not* know the path that the object is supposed to follow.

or the Steerer, in which

- The robot knows a path that it is supposed to follow,
- the robot is translationally compliant (it controls the heading of its wheels, but does not control their rate of rotation), and
- the robot moves forward as a result of being pushed by the object (which is itself being pushed by the Pusher).

The manipulated object sits between the robots. We use *no direct communication* between the two robots, but employ only *indirect* communication through the mechanics of the robots-and-box system. The configuration is conceptually similar to a rear-wheel-drive automobile that has been sliced into three sections: the rear wheels push the passenger compartment forward in a direction determined by the front wheels. The challenge to this configuration is that the pieces *are* separate – the robots have to be programmed to allow flexible trajectory following, while keeping the object between them. The advantage is that the robots can trade roles – the Pusher can become the Steerer and vice-versa. This increases the flexibility of the protocol by allowing such maneuvers as the "back-and-fill" that automobile drivers use for turning cars around on narrow roads. It also permits the robots-and-object system to reverse direction without the robots having to move to different faces on the object. Figure 6 shows two robots moving a rectangular object through a circular arc.

Our analyses of protocols for translational manipulation only, circular-arc following, and more general trajectory-following are omitted here. We do note, though, that for analysis purposes, we treat each robot as a point mass with ideal wheel kinematics – the wheels do not slip relative to the floor under any application of force. We have torque control over the wheels and the robots can rotate in place. Our analysis uses Coulomb friction and quasistatic dynamics, and treats the world as planar – the objects we manipulate are 3D right extrusions of planar objects.

It is sometimes possible to navigate an object in a straight line or along a circular arc using a single robot. If the object's center of friction (COF) is known and fixed, and the robot has little control error, then the robot can plan a control

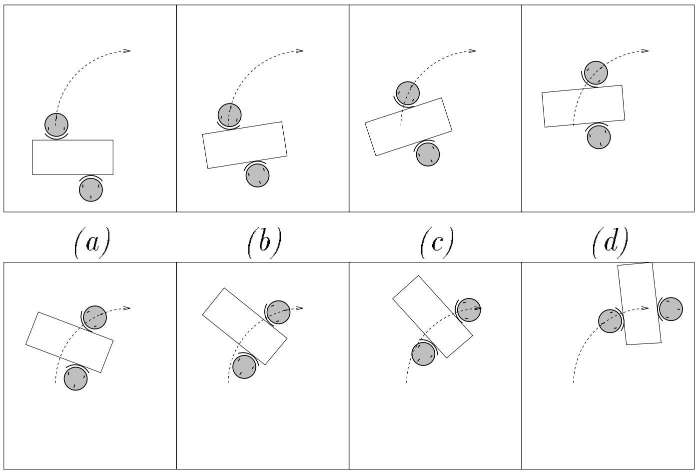

Figure 6. This series of figures depict a box being guided through a 90 degree arc by a steering robot (in front, following the arc), and a pushing robot. The box begins with its front and rear faces approximately perpendicular to the path. In parts *(b)* and *(c)*, the box rotates in the wrong direction, due to poor initial placement of the Pusher relative to the Steerer. By part *(d)*, the Pusher, with no model of the box or the path and with no communication, has compensated for the poor initial configuration. By part *(h)*, the box has traversed the arc and rotated until its front and rear faces are approximately perpendicular to the path.

---

strategy to manipulate the object along a desired trajectory with acceptably small error. However, if the location of the COF is not known precisely, or changes with time (generally the case), then the robot cannot simply plan and execute a reliable trajectory, but must continually sense the object's relative orientation and position and compensate for drift. On the other hand, the forces exerted by the robots in the two-robot Pusher/Steerer system can constrain the object reliably in the presence of greater uncertainties, e.g. in the location of the COF, and under coarser control and sensing.

We summarize our analyses of the Pusher/Steerer system with respect to information invariants. We are working on a precise description of the motion of the robots-and-object system as a function of the robots' programs and parameters of the object such as geometry, mass distribution, and frictional properties. Objects of varying size, mass distribution, and surface friction may be manipulated by our system over a wide range of paths.

For the purpose of illuminating the information and resource requirements of our manipulation system, we turn to the framework of information invariants [7], which defines formal reductions between protocols. We say "Protocol B reduces to Protocol A" when we can use the resources (state, communication, computation, sensors, effectors) of Protocol A to build Protocol B. In writing a reduction, we list explicitly any new resources (including communication) added to Protocol A to assemble Protocol B. We use reductions to make formal comparisons between different protocols that achieve the same task; for instance, we can calculate equivalence between protocols. While a presentation of the formal reductions between single-robot pushing and our 2-robot Pusher/Steerer system is beyond the scope of this report, we will pause to discuss those results informally.

In his monograph on information invariants [7], Donald claims that the spatial distribution of resources has a critical effect on the capabilities of a system. The Pusher/Steerer system validates that claim. Consider a *single-robot* manipulation algorithm such as, e.g., [11]. As implemented on the Cornell mobile robots, the execution system consists of the following skills:

- a *pushing* primitive, `(prim-push)`;
- `(align)` (see Section 2)
- a *steering* primitive, `(steer)`; and
- *a priori* path information.

The Pusher's (entire) control system `(Pusher)` is obtained by the *parallel* composition of `(align)` and `(prim-push)`. The skill `(push-track)` used in Section 2 consists of the *sequential* composition of `(align)` and `(prim-push)`. In the language of information invariants, Brown [3] shows that

$$\texttt{(Pusher)} =_0 \texttt{(prim-push)} + \texttt{(align)}, \tag{2}$$

and hence, from Equation(1) it follows that

$$\texttt{(push-track)} =_0 \texttt{(Pusher)}. \tag{3}$$

We can synthesize the Pusher/Steerer system by redistributing items from the list above into *two separate physical locations*. The *pushing* and *alignment* primitives become the Pusher, and the *steering* primitive and path information comprise the Steerer. Clearly we have added a second robot to the system. But did we actually add resources, or just move them around? The Steerer gets the rotation subsystem from our single-robot strategy; the Pusher gets the translation subsystem. The Steerer gets the path information; the Pusher gets the alignment subsystem (a relative orientation sensor and rotation capability). So we did add a resource! Both the Pusher and Steerer need to rotate.

The Pusher/Steerer system consists of a redistribution of the resources of the single-robot manipulation system described above, plus one rotate motor. Yet, if we substitute a slightly different robot for the Pusher, we find that we do not need to add the rotate motor at all. The Cornell mobile robot CAMEL has a flat bumper instead of a semi-circular one, and when CAMEL is the Pusher, the *alignment* resource is not needed. CAMEL passively maintains correct alignment with the object face due to rotational compliance and the mechanics of line-contact or "blade" pushing. Thus, the Pusher is *rotationally compliant*, but controls translations, while the Steerer is *translationally compliant*, but controls rotations. There is an explicit tradeoff between the choice of robot bumper geometry and the need for an active *alignment* primitive.[6]

In summary, if we choose an appropriate pushing robot, we can build the Pusher/Steerer system by redistributing exactly the resources that would be used in a single-robot manipulation system. There is thus a de facto equivalence, in terms of resource usage, between the two strategies: Pusher/Steerer and single-robot manipulation. It appears, however, that the Pusher/Steerer system admits a larger class

[6]These tradeoffs are precisely quantified in the information invariants theory.

of executable paths; moreover, the system may have other advantages which are less easy to quantify. The benefits of Pusher/Steerer do not derive from an addition of resources, but rather from the spatial redistribution of existing resources.

The combined internal state of the Pusher and Steerer is no greater than that of the single-robot manipulator described above. (It consists of the path information.) Similarly, no extra computation is needed, and there is no explicit communication between the Pusher and Steerer. They "communicate" only through the task mechanics. Implicit communication through the manipulated object confers synchrony. Thus we may describe the Pusher/Steerer system as a distributed manipulation system in which the individual robots are synchronized automatically by the task mechanics – no explicit synchronization or communication between the robots is necessary.

3.2. MPMD Manipulation Experiments

We have performed over one hundred constrained manipulation experiments using the Pusher/Steerer protocol running on several pairs of Cornell mobile robots. In these experiments, boxes and similar objects of varying size, mass, mass distribution, and material properties were manipulated along complex paths up to 50 feet in length. On the basis of these experiments, along with others which are described in [3], we have observed the system to be quite robust in practice. Additional experiments using online navigation methods (human guidance in one case, and visual landmark recognition in another) have demonstrated the flexibility of the system.

The first set of tests we ran on circular arc following was a simple endurance test: how far around a circle, on average, could TOMMY and LILY carry each of a set of test objects? For these experiments, LILY was the Pusher, and TOMMY the Steerer. We ran the protocols at each of a number of turning radii on each of several boxes 5 times, and present here (table 1) the average arc distance traversed before the Steerer loses control. The maximum distance traversed for any test is 1440 degrees (four complete circumferences). We also wanted data on how well the Pusher/Steerer system behaves when we let the Steerer determine on-line what the minimum feasible turning radius is, so we ran the following test: The Steerer alternately executes a counterclockwise semicircle and a clockwise semicircle, for a total of four semicircles, or nine meters, whichever came first. The Steerer monitors its contact mode[7] with the manipulated object, and follows the specified arc only when the contact mode is good (*i.e.*, when the difference in heading between the robot and the box is a sufficiently small angle); when the contact mode is bad, the Steerer goes in a straight line when pushed (*i.e.*, it temporarily suspends the monitoring of odometry and holds its heading constant.) The net effect is that the Steerer traverses an approximate semicircle whose turning radius is determined by the behavior of the Pusher/object/Steerer system. We ran this test a number of times, varying the object and the allowed contact-modes, performing 5 repetitions at each combination. Table 2 summarizes the result for each combination of object and allowed contact mode: The minimum turning radius for a given execution is measured as the distance between the starting position of the object and the finishing

[7]The robots use their bumpers to detect a finite set of contact modes (configurations) with the flat face of a manipulable object. Different modes correspond to different orientations of the object face relative to the robot's heading. See Figure 7.

Table 1. Degrees of arc traversed at given turn radius for several boxes. w is the box dimension between the contact faces; ℓ is the box dimension between the non-contact faces. The values presented are averaged over 5 runs.

	Turning Radius (mm)			
Box ($w \times \ell \times m$)	1000	1500	2000	2500
51cm × 58cm × 3Kg	1000	1050	1220	1440
35cm × 23cm × 2Kg	225	234	528	618
33cm × 58cm × 4Kg	153	342	475	656

Table 2. Turning radius subject to contact-mode restriction for S-curve test. For each combination of box and contact-mode restriction, the test was run 5 times. The first number in each box is the average turning radius; the second is the number of times out of the 5 that the task terminated successfully (either by completing four half circles or by running out of space in our test area).

	Allowed Contact Modes		
Box	3-13	4-12	5-11
51cm × 58cm × 3Kg	1.1m/4	1.4m/4	1.5m/5
35cm × 23cm × 2Kg	0.8m/2	1.1m/3	1.3m/2

position times 1/2 the number of 180-degree turns completed, for those runs which were completed or which terminated when the robots ran out of space. The second entry for each object/contact-mode combination is the fraction of trials which were completed successfully.

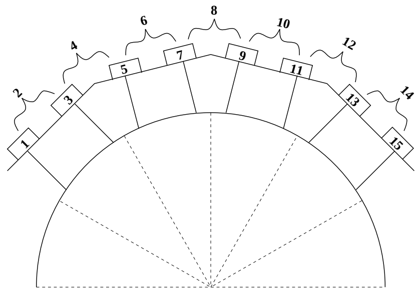

Figure 7. An overhead view of TOMMY's forward-facing bumper: Against a flat surface, TOMMY's bumpers enter one of 15 contact modes. Odd-numbered contact modes occur when a single button is depressed; even-numbered contact modes occur when two adjacent buttons are depressed. Contact modes 2 through 14 each indicate the position of a contacted object relative to the robot with a precision of roughly 7.5 degrees.

There are two main lessons we have learned from our experiments with distributed reorientation and with the Pusher/Steerer system.

1. *Information invariants theory indicates that we should be able to distribute a manipulation task across multiple robots with essentially no additional resource cost.*

This idea is borne out by an examination of the program code that we run on our *Pusher/Steerer* MPMD distributed manipulation system system: The *Steerer* has only the code that pertains to navigation, and the *Pusher* has only the code that pertains to pushing the object. We use no explicit communication, so no program code needs to be added for this purpose.

2. *A mechanics analysis of large-scale manipulation indicates that distributing a manipulation task across multiple robots using a Pusher/Steerer model will allow robots with limited control and sensing to perform that manipulation task in a manner equivalent to a single-mobot system with much more sophisticated control and sensing.*

   Using robots with simple 8-pushbutton bumpers and simple SCHEME control-loops, we were able to manipulate a variety of objects through complex paths; we were able to change and manipulate new objects without changing the programs, and with no re-tuning.

4. Manipulation with microelectromechanical actuator arrays

Next, we discuss our work on sensorless manipulation using massively parallel arrays of microfabricated actuators [1]. The single-crystal silicon fabrication process opens the door to building monolithic microelectromechanical systems (MEMS) with microactuators and control circuitry integrated on the same chip. Our actuators are servoed to uniquely orient (up to symmetries) an object lying on top, and require no sensing. We can also program the array as a sensorless geometric filter—to sort parts based on shape or size.

4.1. Device Fabrication and Properties

In recent years much progress has been made in microelectromechanical systems (MEMS). They consist of structures in the micrometer range which are usually fabricated with VLSI technology. Unlike conventional circuits, MEMS devices have an electrical *and* a mechanical component, i.e. moving parts that can be controlled or monitored with electrical voltage or current.

Fabrication of our devices consists of a sequence of depositions and etches (called SCREAM process, for Single Crystal Reactive Etching and Metallization [13]) which define and partially release the structures from the silicon substrate. Thus, the devices consist of a single crystal silicon core, which is usually covered with aluminum, isolated by a thin film of dielectric silicon oxide. Typically these devices resemble grid or truss structures, because only beams up to a few μm wide (but up to 1 mm long) can be released with the SCREAM process. The fabrication process is compatible with conventional VLSI. Control logic can be integrated on the same chip or even within the silicon structures. Figures 8 through 10 show such actuators at different magnification levels. Each of them consists of a grid structure suspended on a torsional beam. Electrostatic forces cause the device to rotate out of plane by several degrees. When applying an AC voltage the actuator oscillates, with resonance in the kHz range. The design of the grid is asymmetric, with tips only on one side of the grid (see Figure 10). This ensures that when the actuator is in contact with an object, it will generate a lateral force.

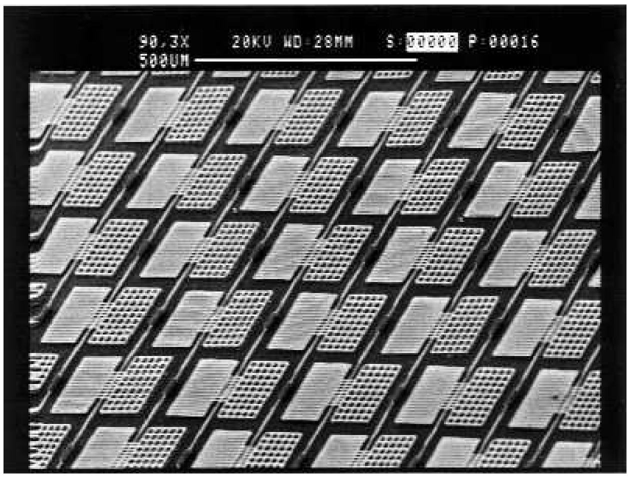

Figure 8. A prototype M-Chip fabricated in 1995. A large unidirectional actuator array (scanning electron microscopy). Each actuator is $180 \times 240\,\mu m^2$ in size. Detail from a 1 in^2 array with more than 11,000 actuators.

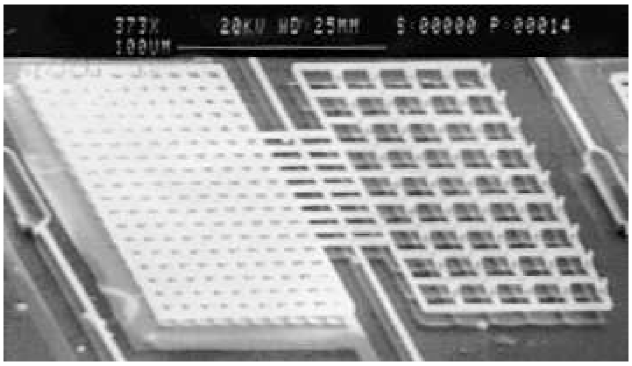

Figure 9. Released asymmetric actuator for the M-Chip (scanning electron microscopy). Left: Dense grid (10 μm spacing) with aluminum electrode underneath. Right: Grid with 5 μm high poles.

Recently we have fabricated arrays with up to 7000 individual actuators for massively parallel manipulation. There is a huge potential of applications. Such MEMS actuator arrays can be used as bulk-fabricated (cheap), ultra-thin transport mechanisms e.g. for paper in copy machines or printers. At the other end of the spectrum, recent advances have brought within reach arrays equipped with tips that can probe and move single atoms [18]. Such devices, employed in a massively parallel fashion, will yield tremendous data storage capacities.

The MEMS array that we present here is designed for "medium size" applications in which objects in the millimeter range are moved, e.g. for an automatic stage of a microscope, or for the assembly of small parts.

4.2. Part Positioning and Orienting

We want to use arrays of up to hundreds or thousands of microactuators to manipulate and orient parts in the millimeter to centimeter range. Each individual actuator is approximately 200 μm $\times$ 300 μm in size and can generate a force of

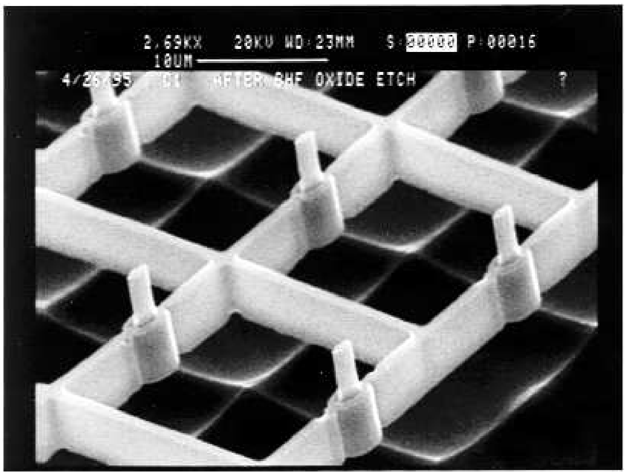

Figure 10. Released M-CHIP actuator (detail) consisting of single-crystal silicon with 5 μm high tips (described in [5].)

up to 10 μN. Clearly this is a strongly distributed manipulation task, as it cannot be achieved with an individual actuator. However, by distributing the task among cooperating actuators the joint force is sufficient (the weight of paper per actuator area is approximately 60 nN, more than two orders of magnitude less than the force generated by an actuator).

Let us consider the task in which a flat polygonal part $\mathcal{P}$ has to be oriented to a specific angle on the array, starting from an arbitrary initial configuration. A global control strategy could act as follows:

(1) Determine the current position of $\mathcal{P}$.
(2) If $\mathcal{P}$ is in the goal configuration, stop.
(3) Compute a motion that brings $\mathcal{P}$ closer towards the goal.
(4) For each actuator, compute the force necessary to induce this motion, and tell the actuator to generate this force.
(5) Repeat.

This strategy is rather complex, and requires individual communication with each actuator, as well as sensing. In the following we show how a simpler and more effective distributed manipulation strategy can perform the same task without sensing, and with the use of only very limited communication resources.

Suppose the array generates a "squeeze" pattern in which all actuators push perpendicularly towards a straight line l. A polygonal part $\mathcal{P}$ on the array will experience a force that causes translation towards the line l. When l intersects $\mathcal{P}$, it also experiences a torque. This can be modeled as forces acting on the respective centroids of $\mathcal{P}$ on either side of l (see Figure 11). These forces are proportional to the surface area of each section of $\mathcal{P}$.

In earlier work we have shown [1] that every polygonal part has a small finite number of stable equilibrium configurations in such an actuator pattern. In equilibrium the forces and moments balance out, such that l becomes a bisector of the part, and the line connecting the centroids is perpendicular to l (see Figure 12).

All possible equilibrium configurations can be predetermined from the geometry of the part. For now let us assume that the part $\mathcal{P}$ has only one equilibrium. $\mathcal{P}$ will

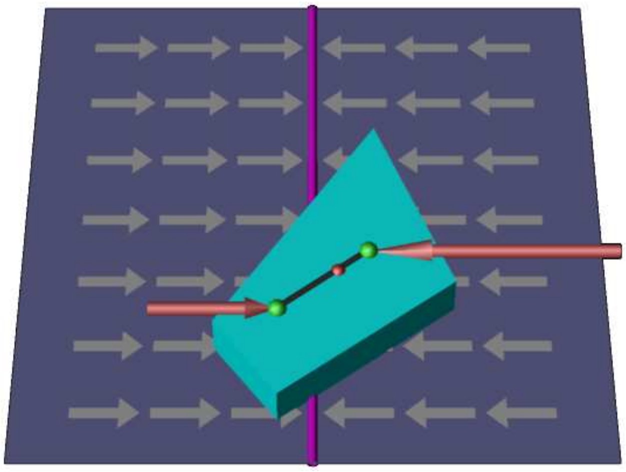

Figure 11. Part in a force field generated by an actuator array. The resulting forces for the left and the right section of the part are shown acting at the respective centers of mass: the part experiences a translational force and a moment.

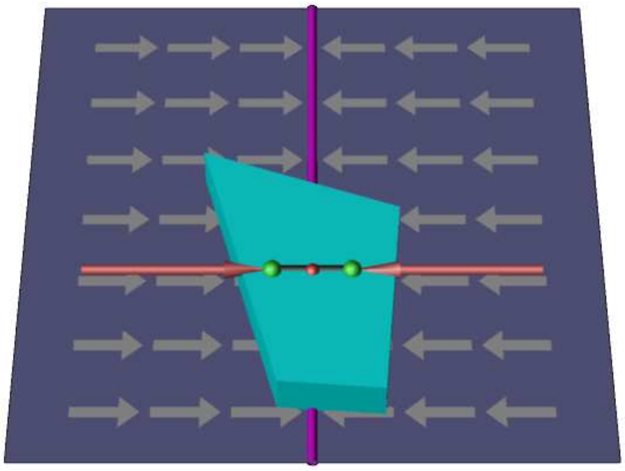

Figure 12. Part in equilibrium: The resulting forces for the left and the right section of the part are of equal magnitude and opposite direction, and the resulting moment is zero.

reach this equilibrium when put on an array with a squeeze pattern. Thus, to orient $\mathcal{P}$ up to symmetry it is sufficient to generate a squeeze pattern with the center line l at the appropriate place.

Note that this strategy does not require any sensing, and that it is not necessary to have individual communication with each actuator. If each actuator knows its relative place in the array, a single broadcast of the location of l is sufficient for each actuator to determine the direction of pushing. We see that by distributing computing resources and state information we can significantly reduce the amount of communication necessary. This SPMD approach simplifies both the control strategy (software) as well as the communication circuitry (hardware) for MEMS actuator arrays.

4.3. Multi-step part alignment

In the previous section we made the assumption that the part $\mathcal{P}$ has only one stable equilibrium. For parts that have multiple equilibria we can still use the same basic idea to align parts up to symmetry. We employ a multi-step strategy in which we successively reduce the number of possible configurations in which $\mathcal{P}$ can be. This is achieved by a sequence of squeeze patterns at specific angles. As Goldberg [8] has shown for the related problem of aligning parts with a robot gripper, there always exist efficient multi-step strategies. We have extended his results for general manipulation in force fields [1].

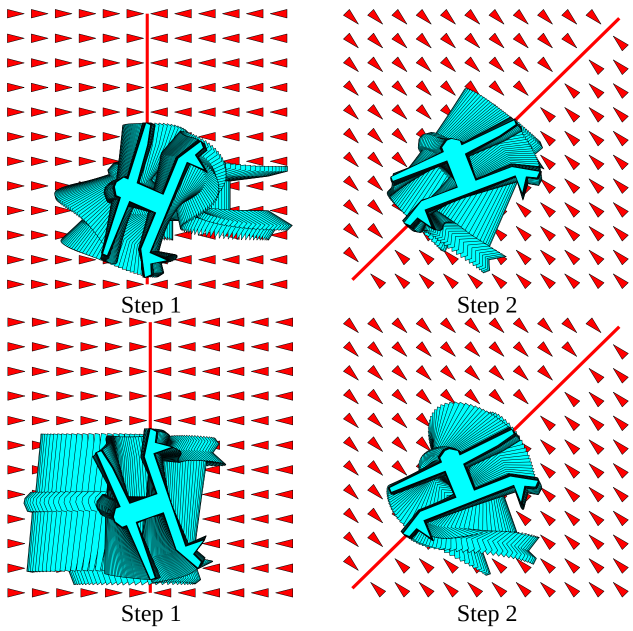

Figure 13. Sensorless parts alignment using force vector fields: Parts reach unique poses after two subsequent squeezes.

As an example consider the traces of a two-step strategy in Figure 13. The ratchet-shaped part is put on the array at two different random initial configurations. After two squeeze steps, the part ends up in the same orientation.

4.4. Summary

We have presented bulk-fabricated MEMS actuator arrays to perform massively parallel manipulation tasks, and described efficient control strategies for part manipulation. We discovered a trade-off between communication and computation resources and have shown how distributing resources can significantly reduce the complexity of parallel manipulation tasks.

Acknowledgements

We thank Jean-Claude Latombe for his great hospitality during our stay at the Stanford Robotics Laboratory.

References

[1] Böhringer, K., Donald, B., Mihailovich, R., and MacDonald, N., Sensorless manipulation using massively parallel microfabricated actuator arrays, In *Proc. IEEE Int.*

Conf. on Robotics and Automation, pages 826–833, San Diego, CA, May 1994.

[2] Brown, R., and Jennings, J., Manipulation by a pusher/steerer, In *Proceedings of Intelligent Robot Systems*, Pittsburgh, PA, August 1995.

[3] Brown, B., *Algorithms for Mobile Robot Localization and Building Flexible, Robust, Easy to Use Mobile Robots*, PhD thesis, Cornell University, Ithaca, NY, 1995.

[4] Cao, Y., Fukunaga, A., Kahng, A., and Meng, F, Cooperative mobile robots: Antecedents and directions, Technical Report, UCLA Department of Computer Science, 1995.

[5] Chandler, D., Atom by atom, *The Boston Globe*, May 5:37ff, 1995.

[6] Donald, B., Jennings, J., and Rus, D., Information invariants for distributed manipulation, *The First Workshop on the Algorithmic Foundations of Robotics*, eds. K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, A. K. Peters, pages 431–459, 1994.

[7] Donald, B., Information invariants in robotics, *Artificial Intelligence*, 72:217–304, 1995.

[8] Goldberg, K., Orienting polygonal parts without sensing, *Algorithmica*, 10(2/3/4):201–225, August/September/October 1993.

[9] Jennings, J., *Distributed Manipulation with Mobile Robots*, PhD thesis, Cornell University, Ithaca, NY, (forthcoming, January 1996).

[10] Jennings, J., and Rus, D., Active model acquisition for near-sensorless manipulation with mobile robots, In *IASTED International Conference on Robotics and Manufacturing*, pages 179–184, Oxford, England, September 1993.

[11] Lynch, K., and Mason, M., Stable pushing: Mechanics, controllability, and planning, In *Proceedings of the 1994 Workshop on the Algorithmic Foundations of Robotics*, San Francisco, CA, 1994.

[12] Mason, M., Manipulator grasping and pushing operations, *International Journal of Robotics Research*, 5(3):53–71, 1995.

[13] Mihailovich, R., Zhang, Z., Shaw, K., and MacDonald, N., Single-crystal silicon torsional resonators, In *Proc. IEEE Workshop on Micro Electro Mechanical Systems*, pages 155–160, Fort Lauderdale, FL, February 1993.

[14] Raibert, M., and Craig, J., Hybrid position/force control of manipulators, *Journal of Dynamic Systems, Measurement, and Control*, 102, 1981.

[15] Rees, J., and Donald, B., Program mobile robots in scheme, In *Proc. of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, 1992.

[16] Rus, D., Donald, B., and Jennings, J., Moving furniture with mobile robots, In *Proceedings of Intelligent Robot Systems*, Pittsburgh, PA, August 1995.

[17] Rus, D., *Fine motion planning for dexterous manipulation*, PhD thesis, Cornell University, Ithaca, NY, August 1992.

[18] Xu, Y., Miller, S., and MacDonald, N., Microelectromechanical scanning tunneling microscope, *Bulletion of the American Physical Society*, 40(1):63, 1995.