# Distributed Coordination in Modular Precision Assembly Systems

Alfred A. Rizzi, Jay Gowdy and Ralph L. Hollis

*The International Journal of Robotics Research* 2001; 20; 819
DOI: 10.1177/02783640122068128

**Alfred A. Rizzi**
**Jay Gowdy**
**Ralph L. Hollis**
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
arizzi@ri.cmu.edu
jayg@ri.cmu.edu
rhollis@ri.cmu.edu

# Distributed Coordination in Modular Precision Assembly Systems

## Abstract

*A promising approach to enabling the rapid deployment and reconfiguration of automated assembly systems is to make use of cooperating, modular, robust robotic agents. Over the past 5 years, the authors have been constructing just such a system suitable for assembly of high-precision, high-value products. Within this environment, each robotic agent executes its own program, coordinating its activity with that of its peers to produce globally cooperative precision behavior. To simplify the problems associated with deploying such systems, each agent adheres to a strict notion of modularity, both physically and computationally. The intent is to provide an architecture within which it is straightforward to specify strategies for the robust execution of potentially complex and fragile cooperative behaviors. The underlying behaviors use a runtime environment that includes tools to automatically sequence the activities of an agent. Taken together, these abstractions enable a designer to rapidly and effectively describe the high-level behavior of a collection of agents while relying on a set of formally correct control strategies to properly execute and sequence the necessary continuous behaviors.*

KEY WORDS—robotics, automated assembly, distributed systems, sensor-based control

## 1. Introduction

Since the introduction of programmable industrial robots at the end of the 1960s and beginning of the 1970s, a significant industry based on these machines developed steadily to a point in the mid-1980s, when it was poised for (and many predicted and expected) an explosion of huge proportions. This explosion failed to happen as a robotics backlash took hold, with the strong perception held by many that programmable machines did not, and would not, live up to the high expectations and demands held by their users.

While the interest in robotics by the industrial sector waned, it continued to grow tremendously within the academic community, resulting in the aggressive investigation of many difficult and fundamental problem areas. The primary research emphasis has been on key component technologies (e.g., kinematics, dynamics, control, 3-D vision, planning, etc.) but only rarely on complete systems, and the integration of the vast majority of these results into industrial practice has been slow.

One result of this history, in our opinion, is that academic robotics researchers—perhaps as a result of the difficulties of technology transfer to industry—have turned their backs on industrial robotics in favor of working in other, more exploratory, areas such as mobile, field, medical, space, and service robotics. Meanwhile, worldwide sales of industrial robots has steadily grown during the past decade. To enable further growth, there must be significant changes in the way robotic systems are deployed in manufacturing environments. It would appear that there are now significant opportunities for applying increased *intelligence* and *autonomy* to industrial robot systems. This observation derives from several factors, including the following:

- increasing demand for ever smaller and more complex products whose product lifetimes are constantly shortening;
- the increasing need to remove humans from the immediate vicinity of the manufacturing process both because of product scale/precision and cleanliness requirements;
- the relatively recent widespread and ubiquitous availability of significant computing power at reasonable costs;
- the explosive deployment of high-performance information and communications technology—most notably in the application of local and wide-area networking within the manufacturing arena.

The International Journal of Robotics Research
Vol. 20, No. 10, October 2001, pp. 819-838,
©2001 Sage Publications

### *1.1. Practical Difficulties with Automated Manufacturing Systems*

The fundamental problem with modern robotic manufacturing systems is that the individual components (robots, part feeders, conveyor systems, etc.) are generally designed as stand-alone devices. As a result, little or no explicit effort is dedicated to enabling the integration of these factory components into a complete manufacturing system. Similarly, information about the design of most complete systems is scarce, as there are few incentives for system integrators to document their work, and it is difficult to extract durable truths from case studies. There remain prohibitively high economic and technical costs associated with the factory integration process, which in turn severely limit the utilization of robotic elements in many practical applications.

Meanwhile, work on programming robotic assembly systems has progressed at both the task level (e.g., "place part A on part B") and at the manipulator level (e.g., "move joint three 10.15 inches, close gripper, etc."). These efforts are, in general, based on standard computer languages, with the addition of special primitives, constructs, and libraries to support the physical control of a robot (Lozano-Perez 1983). The resulting languages are designed to enable the effective programming of a single robot and do not inherently support distributed automation systems. More abstract programming models have recently appeared (Berry and Gonthier 1992; Harrigan 1993), but typically these are either task or process based and are applied to the programming of statically configured work cells (which might contain multiple robots). To date, these task-level systems have not moved beyond laboratory study, while the more basic manipulator-level systems, despite the enormous programming complexities required to produce functional behavior, have become widely accepted.

### *1.2. Related Efforts*

A number of academic and industrial groups have attempted to provide partial solutions to these fundamental problems over the past decade.

A leading-edge benchmark, which attempts to address some of these issues, is Sony's SMART flexible assembly line. It makes use of SCARA robots equipped with indexing multiple grippers and modular product and part transport systems to simplify the mechanical problems of factory reconfiguration. Unfortunately, the individual modules are physically large, and the problems of programming and tuning a complete factory system are still daunting.

A key study was the DARPA microfactory demonstration (Foslien and Nibbe 1990), developed as part of the Defense Department's Intelligent Task Automation program. This work emphasized operation in unstructured environments, recognition and grasping of overlapping parts, semiautomatic planning, and geometric reasoning. The system that resulted used parts kitting and sensor-moderated motion to assemble a precision microswitch. While meeting many objectives, the system required 18 minutes to complete the microswitch assembly task.

A recent significant trend is the notion of programming and operating robots over the Internet. For example, a concept of virtual laboratories was recently demonstrated (Gertz et al. 1994), showing that a robot in one laboratory can be programmed and controlled from another laboratory thousands of miles away. In another case, exploration and tele-gardening (Goldberg et al. 1995) were demonstrated. Both of these studies show that it is now possible to allow meaningful remote (Internet-based) interaction between a robot and a programmer or operator. However, neither demonstrates the level of expressiveness required to undertake a significant practical manufacturing task.

Sandia National Laboratory has developed its Agile Manufacturing Prototyping System (AMPS), composed of robotic cells supplied by various vendors.[1] Simultaneously, industrial robot producers have begun to service the demand for increased flexibility and precision. Adept has developed a concept of "rapid deployment automation" (Craig 1997) that embraces key elements of modularity and offline programming. Megamation and Yaskawa have produced systems of small, modular, easily programmed robots capable of moderately precise assembly.

Structurally more similar to our approach is the notion of Holonic Manufacturing Systems, which would be composed of "holons," wherein each holon is an active information-processing entity (Christensen 1994; Brussel et al. 1999). Sometimes a holon is associated with a specific computer and mechanism (e.g., a specific robot holon); sometimes a holon is associated with a specific physical entity, but its processing moves from computer to computer (a holon that represents a product); and sometimes a holon can be associated purely with an abstract expertise (a holon that represents a resource management expert). All of these holons would negotiate with each other within an all-encompassing "holarchy" that attempts to model the heterarchical and hierarchical aspects of robust biological organizations (Koestler 1967). Holonic systems are essentially a particularly ambitious class of agent-based manufacturing systems. They seek to encompass the entire manufacturing enterprise, from human resource management to putting parts together.

## 2. The Agile Assembly Architecture

Our vision of an agile manufacturing system is one that provides a large pallet of modular robotic processing and product transport systems from a wide variety of vendors, with each module presenting a standard mechanical, computational, and algorithmic interface enabling their simple and rapid

1. See http://www.sandia.gov/isrc/RMSEL.html.

integration (both physical and programmatic) into a complete factory system. In contrast to much academic research on agile manufacturing systems, we are not striving to provide a "universal assembly machine" but rather wish to adhere to the industrially accepted model of flow-through (assembly line) processing while providing for the rapid deployment and reconfiguration of such systems. We foresee this being achieved through the use of compact, mechanically simple elements whose customizable combined behaviors provide the specific complex capabilities required for a specific application. Furthermore, we do not foresee these modules being used to form a "lights-out" factory, but rather we expect them to act smoothly in concert with humans, serving as highly capable and intelligent tools for their operators.

In our laboratory,[2] we have developed new hardware and software technologies along with programming and integration strategies for just such a class of automated assembly system. This new class of automation systems is suitable for the manufacturing of a wide range of precision high-value products such as magnetic storage devices, palm-top and wearable computers, and other high-density equipment (Hollis and Quaid 1995). Our approach draws extensively on high-speed, wide-area communication and intensive distributed computation. The Agile Assembly Architecture (AAA), as we have termed this system, supports the creation of miniature manufacturing systems (minifactories), built from small modular robotic components, which will occupy drastically less floor space than today's automated assembly lines. Our goals are to reduce assembly system changeover times, facilitate geographically distributed design and deployment of assembly systems, and increase product quality levels.

We are developing AAA as a distributed system of tightly integrated mechanical and computational robotic modules endowed not only with information about their own capabilities but also with the ability to appreciate their role in the factory as a whole and negotiate with their peers to participate in flexible factory-level cooperation (Rizzi, Gowdy, and Hollis 1997). A unified interface tool will allow a user to select and order these mechanisms over the Internet and to assemble, program, and monitor them in both a simulated factory environment and the real factory environment.

AAA relies on factory-wide standard procedures and protocols and well-structured autonomy to simplify the process of designing and programming high-precision distributed assembly systems. The architecture makes use of agents' self-knowledge and ability to explore their environments to make the transition between simulation and reality as painless and seamless as possible.

Our sample instantiation of these ideas is a modular tabletop factory that we refer to as a *minifactory* (see Fig. 1). The key technical aspects to note about this system include the use of distributed low degree-of-freedom (DOF) robotic agents[3]

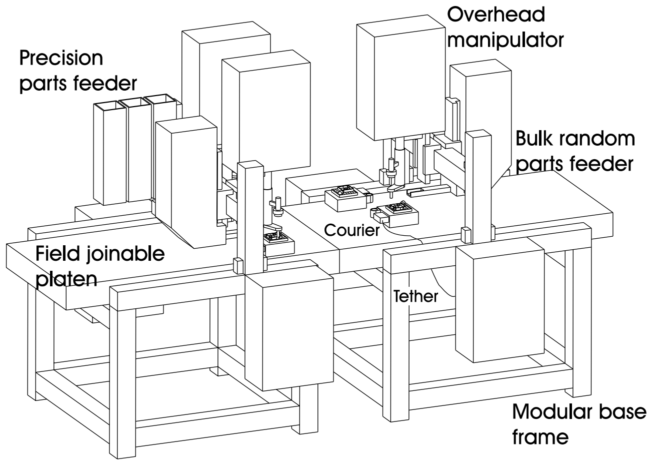


Fig. 1. A segment of a typical minifactory.

and the integration of product transport and manipulation subsystems. The entire system is composed of compact elements with standardized mechanical and electrical interconnects, allowing for the rapid setup and adjustment of a factory system.

The minifactory incorporates planar robot couriers that travel on connected tabletop platen surfaces. These robots are derived from planar linear motors that float on air bearings and translate along the platens in two directions with micron-level precision. Other devices, including 2-DOF overhead manipulators, are mounted on modular bridges above the platens (see Fig. 1). The couriers are responsible for both carrying the product subassemblies from one overhead agent to another and cooperating with the overhead agents to execute assembly operations. Limiting the robots to 2 DOFs has advantages in terms of modularity, reliability, and performance (Quaid and Hollis 1996) but allows the minifactory to perform 4-DOF assembly tasks through the use of robot cooperation. Each low-DOF device has integrated high-performance computing resources and serves as an agent in the AAA context. This eliminates the need for central resources that would degrade the modularity and scalability of the system.

Adding or deleting functionality to a minifactory is relatively straightforward. New agents can be added to the system almost anywhere (imagine new overhead agents or couriers inserted into Fig. 1), while removing agents has little or no effect on the existing ones. Central to this model of factory integration is that a module's internal functionality never be subject to modification by a factory designer but rather only by the module's vendor. Adhering to this stricture ensures that modularity, and hence the system's inherent agility, will not be broken.

Because each factory agent is fairly simple and limited in functionality, the cost of entry for module suppliers is kept

2. See http://www.cs.cmu.edu/~msl.

3. By agent, we explicitly mean a mechanically, computationally, and algorithmically modular manufacturing entity (e.g., robot) capable of both communication and physical interaction with its peers.

low. In this way, each modular component can employ the latest and best technology from that particular company. We argue that this sort of approach can respond more quickly to changing market opportunities than can today's state-of-the-art modular systems.

### *2.1. Underlying Challenges*

A fundamental component of our long-term goal is to elevate the design of automated assembly systems from the detailed technical problems associated with designing and integrating independent mechanisms to the more salient problem of designing the factory as a whole. We see this as a complementary effort to that provided by the industrial engineering and operations research communities, but it can provide a natural mechanism for the widespread application of new factory design methodologies. Ideally, the collections of machines that follow the framework presented here will form a natural template onto which the more abstract ideas of factory design and optimization can be applied.

From our perspective, several key barriers currently stand between current best practice and a more agile and open manufacturing infrastructure. Fundamentally, these barriers all relate to the need for standard mechanisms to support interaction between agents, designers attempting to integrate the agents into a system, programmers developing control software for a factory involving the agents, and operators whose task is to monitor performance and provide support when an agent is unable to cope with its environment.

As mentioned in Section 1, the current practice in the robotics and automation community is to focus on the engineering of individual robots and mechanisms, with little or no consideration for how they are later integrated into a complete factory system. Only by designing robotic agents that are explicitly prepared to participate in a larger factory system can we begin to provide the types of tools necessary to move toward fundamentally more useful systems of machines. This represents what we see to be an important goal of the robotics field: the construction of mechanisms capable of both physical and "social" interaction. Physical manipulation has been the province of both the academic and industrial robotics communities from their inception, while social interaction has been an academic goal (Minsky 1986) that has produced several novel and interesting systems (Brooks and Stein 1994; Lenat 1995). There have been, however, few practical applications for these systems.

We believe our efforts to design rapidly reconfigurable and user-friendly factories represents a modest step toward achieving this goal. The scope of social interaction has been explicitly limited to two well-defined domains: interagent interaction for factory coordination and interaction between agents and the factory personnel. Given even this limited scope, the details underlying the definition of these agent interfaces are not obvious and comprise a significant portion of the AAA.

#### *2.1.1. Factory Interaction*

The definition of a suitable "machine language," for interagent communication is a central issue in enabling the type of interaction under discussion between both multiple robotic agents and humans and agents. For interagent communication, the basic requirements include the following.

*Extensibility*. Whatever the actual format of messages, the underlying media must efficiently allow for the introduction of not only new message formats but also the negotiation of completely separate communication modalities. In principle, these allow the natural growth and development of new methods for interagent coordination, and with responsible classification of which protocols are to be considered "required" and which are "optional" for an agent, it is reasonable to expect long-term compatibility through the use of the required basic protocols.

*Real-time coordination*. Sufficient communications capability (enough bandwidth with sufficiently low latency) is essential to allow the tight coupling and integration of agents that are incapable of performing complete manufacturing tasks in isolation. In the sample system described in Section 4, it is clear that there are significant advantages both in terms of flexibility and simplicity inherent in supporting such distributed mechanisms. This issue is mitigated by the fact that in general, an individual member of a factory is likely to only interact with a well-defined "neighborhood" of peers—for example, an insertion mechanism need only perform precise coordination with a part feeder (providing the part to insert) and with a product transport mechanism (presenting the subassembly to be operated on)—greatly limiting the scope of high-performance communication by nature of its locality.

*Factory communication*. Conversely, there is a need to provide a standard means for factory-wide control and monitoring and hence the need for a standard interface to join every robotic agent in a factory together with each other and an arbitrary number of control and monitoring workstations. The intent is to provide a common medium and basic interchange format for the most rudimentary forms of factory control while providing a means by which agents can negotiate for the use of more application-specific interchange formats. By requiring every element of a factory system to "understand" this basic level of interaction, we strive to ensure that each and every agent is capable of participating in the factory at a minimum level.

#### *2.1.2. Integrated Design, Simulation, and Evaluation Tools*

Not only is it necessary to require a facility for interaction between agents, but it is equally important to support interaction between humans acting as factory designers and the agents. In a traditional design process, there are three major classes of interaction between the designer and a component under consideration.

- *Preliminary selection*: Initial evaluation of a manufacturing component for its suitability to a problem.
- *Detailed evaluation*: Iterative validation and discard of candidate solutions and components based on analysis, simulation, and mock-up of proposed designs.
- *Integration and refinement*: Detailed analysis, design, construction, and test of a working system.

While the above distinctions are somewhat arbitrary, they highlight fundamentally different forms of inquiry performed on candidate components by a designer and provide a model under which we can explore the interactions necessary to reduce the designers' uncertainty about the factory system they are working on.

Given the widespread acceptance and rapid development of high-performance computation and communication systems, particularly as embodied in the Internet, we have sought the integration of such capabilities with robotic agents to enable new relationships between the designer and the component. In stage 1 of the design process, when traditionally decisions would be made based primarily on vendor-provided catalogs, it now becomes possible for the designers (or some agent acting on their behalf) to directly interrogate an actual mechanism (probably at a vendor's facility) for relevant properties (Gowdy and Rizzi 1999). There are myriad options for exactly what remote entity answers such queries, depending on the nature of the component under scrutiny. In the case of "brainless" components, this would be similar to a catalog search, but for full-fledged factory agents, the designer could interact directly with the specific agent under consideration, potentially providing a significantly more accurate representation of the actual mechanism and its capabilities.

The implications of this model on phase 2 of the design process are more significant. It now becomes possible for the component under evaluation to provide a number of different "renderings" of its physical and behavioral models for use by a designer. With the careful integration of tools for either retrieving downloadable representations from factory components or remotely involving the component in a distributed simulation or analysis process, it becomes possible for the item under review to provide a model with an appropriate level of fidelity to its actual performance. It is easy to imagine a broad range of models ranging from trivial kinematic representations to highly detailed physics-based distributed simulations or even remote experimental environments being made available to a designer through a single and consistent set of design and simulation tools capable of allowing the construction and interrogation of a highly accurate "virtual" factory identical to the design under evaluation.

Finally, in phase 3, as a design is refined and physical experiments are undertaken, it is through these same tools that the designer will continue to interact with the evolving factory design. The ultimate goal is the truly seamless transition from factory simulation to operation, but with enough expressiveness and flexibility in the underlying components and representations so as not to unduly constrain the behavior and performance of the final system.

### 2.1.3. Programming Interaction

The key goal in simplifying human-machine interaction is one of providing a simple and natural language for specifying complete machine behavior. Further complicating the problem is that given a large collection of disparate agents, it is necessary to distribute the "factory program" among the various agents. In contrast to current practice, however, it is also necessary to provide tools and highly expressive, yet convenient, languages that aid a factory programmer in developing and debugging the agent-level programs that instantiate a specific solution to a manufacturing problem. Most of our effort in this domain is focused on understanding and developing appropriate representations for machine behavior in a factory setting, as detailed in Section 3.2.

### 2.1.4. Operator Interaction

Finally, as we do not see the near-term future of automated manufacturing to be "lights out," it is important to consider the role played by factory operators and their interaction with the agents that make up the factory. Predominantly, we see operators serving as aids to the factory, acting to help agents recover from and avoid situations that they are unable to manage in an automatic manner. This includes such mundane tasks as managing factory supplies by refilling part hoppers and removing finished products, possibly for additional processing by a more traditional factory system. Furthermore, we foresee operators being called to the aid of agents that recognize factory difficulties that they are unable to recover from. This form of interaction should include the ability of an agent to notify an operator of the difficulty, allowing the operator to remotely (from across the room or facility) interact with the agent in question and its peers via a set of agent "front panels" or "dashboards" (remotely rendered presentations of the agent's status) to diagnose the problem and choose a corrective course of action.

## 3. Instantiation

Minifactory is our instantiation of the concepts of AAA. A minifactory consists of a potentially large collection of mechanically, computationally, and algorithmically distributed agents, with each element in this collection adhering to the guidelines for modularity imposed by AAA.

The most obvious departure from traditional automation systems and one of the most obvious embodiments of our philosophy of factory-level integration can be seen in our choice to integrate product transfer and local manipulation. As such, we have eschewed the traditional use of SCARA manipulators coupled with part conveyor systems and local fixtures. Alternatively, as depicted in Figure 1, minifactory makes use of 2-DOF manipulators and 2-DOF planar couriers moving over a high-precision platen surface. The couriers are thus responsible both for product transport within the factory and for transiently forming cooperative 4-DOF manipulators when they present subassemblies to stationary manipulators.

The manipulator agents are capable of vertical ($z$) and rotational ($\theta$) movement. The $z$ range of motion is approximately 125 mm and achieves a resolution of 2 $\mu$m, while the range of motion in $\theta$ is approximately $\pm 270$ degrees with a resolution of 0.0005 degrees (Brown et al. 2001). The manipulator also incorporates a field-rate frame grabber for image acquisition, and its end effector contains a monochrome camera, a vacuum suction pickup device, and a force sensor. The result is a device capable of participating in a wide variety of sensor-guided assembly operations (Brown et al. 2001).

The courier agents provide motion in the $x$, $y$ plane—both translation and a limited range of rotation. They incorporate a magnetic position sensor device, providing 0.2 $\mu$m resolution ($1\sigma$) position measurements (Butler, Rizzi, and Hollis 1998) and closed-loop position control (Quaid and Hollis 1998).

## *3.1. Runtime Coordination and Communication*

Any element of a minifactory, be it a courier, a manipulator, or some custom-designed module, must provide a minimal level of capability to participate in the minifactory "society." Currently, there are three general classes of capability every agent must reliably provide: *basic trustworthiness*, *self-initialization*, and *interagent coordination*, the latter of which includes facilities for resource negotiation.

### *3.1.1. Basic Trustworthiness*

For an agent to be a successful member of a factory community, its peers must be able to trust it to reliably represent itself. Practically, this manifests itself in the form of three fundamental capabilities:

- All agents must advertise their basic capabilities and the protocols they understand to their peers.
- Every agent must be capable of reporting its current status and its understanding of its environment.
- Each agent must implement reliable and safe failure detection and recovery schemes.

The first two of these requirements are essential to address the issues of Section 2.1.1 and support the graceful coordination between minifactory components, their peers, and factory monitoring tools. Furthermore, the ability to advertise capabilities addresses the need for a predefined extensible protocol suitable for the exchange of such information between agents. The next two capabilities may well be the most important and quite possibly the most difficult to precisely define and implement. The assertions demand that agents be capable of constantly monitoring the state of the factory available to them. Furthermore, when an agent detects conditions outside the norm, it must be capable of independently correcting the aberration, negotiating with its peers to recover from the fault, or broadcasting its inability to proceed, thus bringing the factory to an orderly stop. Although it is potentially difficult to guarantee this level of capability in an arbitrary system, we feel that through judicious use of a combination of traditional AI reasoning (Musliner, Durfee, and Shin 1995) and reactive behaviors (Burridge, Rizzi, and Koditschek 1999), it can be achieved in the highly structured domain of the minifactory.

### *3.1.2. Factory Calibration/Initialization*

Integral to the rapid deployment of an automation system is the need for precise and reliable calibration and initialization whenever a factory is "turned on." There are three interrelated tasks that must be collectively undertaken by the minifactory components to successfully initialize a factory system. This process begins with agents identifying their peers through the use of messages broadcast to the factory at large. Following this, couriers explore their environs to discover both the exact geometry of the platen surfaces, as well as the positions of any stationary agents within their range of motion (Butler 2000). Finally, through a careful exchange of this information between agents, a complete map of the minifactory can be constructed both in the agents and in a monitoring interface tool.

### *3.1.3. Robotic Agent Coordination*

Since individual elements of the minifactory are rarely capable of performing "useful" tasks alone, they each include standard mechanisms for orchestrating their coordination. Fortunately, the locality of action performed by individual agents provides a natural locality of communication and coordination. To help alleviate the problems associated with manually coordinating the motions of all of these machines, we have chosen to make use of a *geometry reservation system*. Under this system, factory elements that are potential competitors for a specific predefined segment of the factory floor (platens) are grouped and required to negotiate for the use of that shared resource. In principle, an individual agent may well be a member of several different groups of agents sharing myriad resources associated not only with physical resources but potentially with more abstract factory goals. It is the neighborhood groupings of agents that form the basic fabric for cooperation between the elements of the factory system.

The most fundamental form of this cooperation happens whenever a courier and manipulator transiently form a 4-DOF system to perform a part placement task. The most basic mode for such cooperation takes the form of a virtual linkage between two agents, in which one agent is effectively slaved to the state of the other, allowing for simple coordinated movement. Other modes of cooperation include coordinated behavior changes and cooperative sensor-based action. Behavior changes are used to encode the sequence of operations necessary for a high-precision force-controlled insertion task (e.g., manipulator exerts low vertical force while the courier "finds" the hole, followed by the courier becoming compliant while the manipulator exerts higher forces to perform the insertion), as described in Section 4.2.

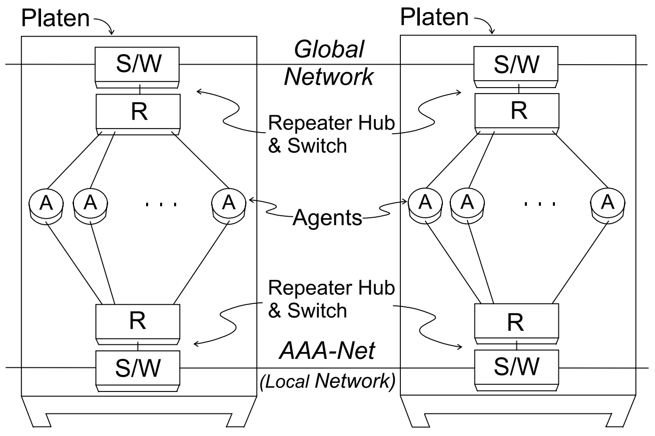


Fig. 2. Physical network structure.

### 3.1.4. Communications Infrastructure

To support seamless cooperation between physically distinct agents in the minifactory, each agent is equipped with two network interface devices. These provide a scalable communications infrastructure throughout the minifactory system. The first interface connects to a network, which carries non-latency-critical information, such as user commands or information destined for the factory interface tool. This network uses standard IP protocols (Postel 1981). The second interface connects to a network (AAA-Net), which carries real-time information critical to the timely coordination of activity between agents. This includes the coordination described in Section 4 as well as other activities performed by multiple agents transiently acting as a single *machine*.

Unlike the majority of industrial field networks (e.g., Profibus (www.profibus.com), ControlNet Online (www.controlnet.org), and WorldFIP (www.worldfip.org)) AAA-Net makes use of commercial off-the-shelf 100 Mbps Ethernet hardware. This provides access to a wide variety of low-cost and compact hardware options. The result is (1) reduced investment in network infrastructure hardware, (2) simplified installation and maintenance, and (3) direct access to rapidly improving Ethernet technology. Unfortunately, as a result of the CASM/CD media access rule used by IEEE 802.3 Ethernets, the timing of packet delivery is not deterministic. Thus, transmission latency cannot be absolutely bounded, and packet delivery cannot even be guaranteed. However, our specification for agent coordination only requires the delivery of 100 byte data packets at a maximum rate of 1 kHz between agents that are physically collocated. Given the maximum agent density, we can conservatively bound the local bandwidth needs of the entire minifactory system at roughly 10 Mbps. By choosing to use 100 Mbps fast Ethernet technology ("IEEE-802.3u-1995 10" 1995), we are able to provide an infrastructure that operates well below its total capacity and thus minimizes the risk of packet collisions and the associated delays and losses in communication.

Figure 2 depicts the communications infrastructure of the minifactory system. As can be seen, both the AAA-Net and the global IP network are configured as a chain of star topology local networks, with a fast Ethernet repeater hub at the center of each star and fast Ethernet switches forming the connections between the local network segments. The repeater hub allows each agent in a network segment to directly communicate with its immediate neighbors, while the frame relay switches allow for arbitrary daisy chaining of the network, overcoming the topology limitations that are fundamental to fast Ethernet. The switches also serve to localize communications within the factory system by not transmitting data packets destined for local agents to the remainder of the factory and by selectively transmitting those packets destined for other network segments toward their destination, yielding a scalable communications infrastructure.[4]

To facilitate a wide variety of local interactions between agents, the AAA-Net protocol supports both a connection-based guaranteed data transmission scheme as well as a connectionless nonguaranteed data transmission scheme. These services are not unlike the familiar TCP and UDP services commonly used on IP networks, although they are significantly simplified to improve performance in the highly structured network environment of AAA. Figure 3 depicts the simplified header used by the AAA-Net protocol. For nonguaranteed communication, the TYPE field in Figure 3b is set appropriately, and the body of the message is placed in the DATA field. The packet is then immediately placed on the Ethernet network, with no effort made to guarantee its delivery. This form of communication is appropriate for the exchange of rapidly changing values, such as the velocity commands sent from the manipulator to the courier described in Section 4, where the loss or delay of a single packet is insignificant since additional data will follow shortly.

This same packet format is also used to implement a connection-based guaranteed delivery protocol. The packet

4. Note that in AAA/minifactory, the bulk of high-bandwidth communication will be between agents that are attached to the same network segment, while the remainder will involve agents that are typically attached to neighboring segments (Rizzi, Gowdy, and Hollis 1997).

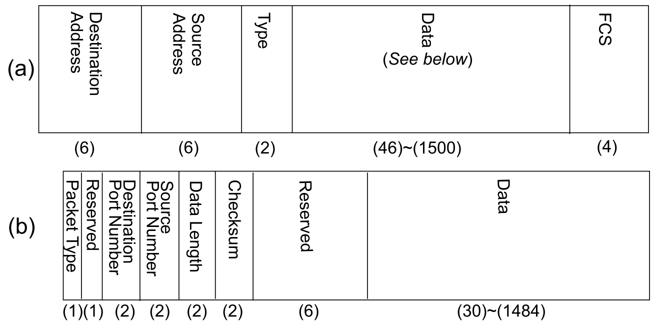


Fig. 3. Ethernet frame format used by AAA-Net. (a) Standard IEEE-802.3 frame format. (b) AAA-Net data format.

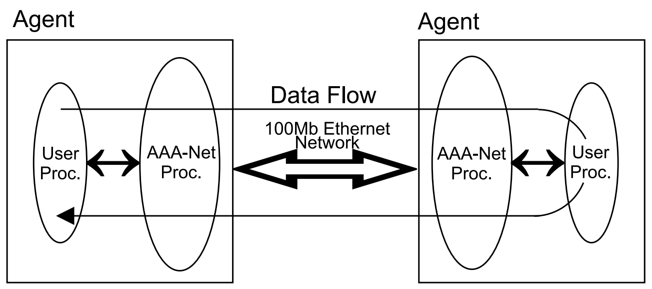


Fig. 4. Test setup for AAA-Net latency measurements.

TYPE is again set, and the fields marked as "reserved" are used by the protocol to ensure reliable delivery of the data stream. This protocol is appropriate for the exchange of larger, more complex data structures or messages that will only be sent once and whose receipt must be guaranteed.

In addition to the interagent cooperative behaviors demonstrated in the remainder of this paper, we have independently evaluated the performance of AAA-Net protocol and underlying network hardware (Kume and Rizzi 2001). A collection of agents connected to the same network segment exchanged messages, as shown at 1 kHz, and the round-trip time was measured. To evaluate the impact of network contention, this test was performed with one, two, and three pairs of agents communicating simultaneously. Messages of both 100 and 1000 bytes in length were transmitted, and the results are shown in Table 1. From our past experience, we estimate that between 45% and 50% of the average round-trip time is spent passing through the AAA-Net daemon process the four times a message requires to make the complete circuit. The results demonstrate the viability of using this relatively low-cost network infrastructure to reliably perform real-time coordination with under 1 ms latency.

### *3.2. User-Level Design, Programming, and Monitoring Tools*

In the absence of a centralized controller, a minifactory has instead a centralized *user interface tool* capable of supporting the design, programming, simulation, and runtime monitoring and control of a minifactory. Thus, the overall behavior of a minifactory results from the interaction between its agents, their programs, and the environments in which they operate. The central challenge for the minifactory simulation and programming environment is to provide the services discussed in Section 2.1.2, facilitating the development of well-debugged distributed programs, while easing the difficult transition from the simulated world of bytes and pixels to the real world of actuators and sensors.

The distributed and cooperative nature of minifactory programs has implications about the program form: an agent program is not simply a script but rather defines an instance of a class that implements a number of specific methods. The program may define a new class to be instantiated or a subclass from a preexisting standard one, but the class must implement a standard interface. This concept is very similar to the Java applets that are used in World Wide Web programming. For various reasons, including ease of porting and licensing issues, we have chosen Python (van Rossum 1998), another object-oriented language that can be interpreted, or byte compiled, to program our robotic agents rather than Java.

Every robotic agent program must provide two methods: `bind` and `run`. They encapsulate the two major conflicting requirements of an agent program—it must specify behavior in reference to external entities but must run in a completely distributed fashion and cannot rely on any centralized resource or database during execution. For example, a courier must be able to know it will be interacting with a particular manipulator, but the information on how to contact that manipulator must reside with the courier at runtime. Similarly, a manipulator may need to know it will get parts of a specific type from a specific parts-feeding device without having to contact a central database at runtime to get the geometric and product-specific information it needs to perform its operations.

In the AAA environment, an agent program has two distinct phases in its life cycle. First, it is written and simulated within a centralized interface and design tool. This tool provides a factory developer with a global view of the factory system under development (Gowdy and Butler 1999). When executing within the centralized simulation environment, the `bind` method simply causes the relevant items to be looked up in the simulation database before proceeding to execute the `run` method. The second phase occurs when the factory developer downloads an agent program from the simulation environment to the physical agent. At this point, the agent program must be "bound" with all of the global factory information the agent will require while executing the program. To bind a program, the interface tool

**Table 1. Measured AAA-Net Round-Trip Times (mean and standard deviation) for Varying Network Loads**

| | Round Trip ($\mu$sec) | | | |
|---|---|---|---|---|
| | 100 Byte Message | | 1000 Byte Message | |
| Number of Agents | Mean | Standard Deviation | Mean | Standard Deviation |
| 2 | 566 | 20 | 804 | 22 |
| 4 | 562 | 19 | 816 | 34 |
| 6 | 567 | 21 | 826 | 38 |

executes that program's bind method and uses the results to construct a small database containing the information necessary for the agent to locate, both geometrically and logically, all of the factory elements it will interact with. This small database serves as a starting point for an agent's self-initialization and exploration of its environment. For example, in the sample courier program (Fig. 5), the bind method calls `bindAgent(''FeederManip'')`, which declares that the agent program wants to know about the manipulator named *FeederManip* and assigns the result of that binding to a local member variable for use in its run method. As a result of the invocation, the interface tool will add the relative position of *FeederManip* in the courier's frame of reference as well as the network address of *FeederManip* to the local database, which is sent to the courier along with the program text.

The run method contains the "script" that actually runs during execution, implementing the discrete logic of the agent that is responsible for initiating and coordinating the behavior of this agent. For example, the run method in Figure 6 causes the agent to loop, transferring parts from a parts feeder to couriers that request them. The run method is written using convenience methods defined by the program's superclasses, which themselves cause the exchange of messages between agents using AAA protocols and the deployment of hybrid controllers (described in Section 3.3 below). For example, the convenience method invoked by `self.getPartFromFeeder` is implemented in the parent class, `ManipProgram`. This convenience method extracts information from the product prototype and feeder instance passed into it and sets up and monitors control policies that will robustly pick a product of that type from that feeder.

### 3.3. Real-Time Control

As we have already described, an agent program in AAA has two distinct but related runtime responsibilities: (1) it must carry out semantic negotiations with its peers to accomplish work on behalf of the factory, and (2) it must properly parameterize and sequence the application of low-level control strategies to successfully manipulate the physical world. The programming model we are using simplifies the relationship between these two responsibilities and minimizes their

```
# Agent class definition
class Program(CourierProgram):
  # Binding method
  def bind(self):
    # superclass has some binding to do
    CourierProgram.bind(self)

    # Bind to a particular manipulator
    self.source = self.bindAgent
      (''FeederManip'')
    # Bind to a particular factory area
    self.corridor = self.bindArea
      (''CorridorA'')

  # Execution method
  def run(self):
    # initialize the movement
    self.startIn(self.corridor)

    # block until manipulator is ready
    self.initiateRendezvous(self.source,
      ''Feeding'')

    # move into the workspace
    self.moveTo(self.sourceArea)

    # coordinate with manipulator to
    # get product from it
    self.acceptProduct()

    # The coordinated maneuver is done
    self.finishRendezvous(''Loading'')

    # move out of the workspace
    self.moveTo(self.corridor, blocking=1)

# instantiate the applet
program = Program()
```

Fig. 5. A simple courier program.

```
# Agent class definition
class Program(ManipProgram):
  # Binding method
  def bind(self):
    # bind a bulk feeder
    self.feeder = self.bindDescription
      (``ShaftFeeder'')
    # bind product information
    self.product = self.bindPrototype
      (``ShaftB'')

  # Execution method
  def run(self):
    while 1:
      # convenience function for getting a
      # product from a feeder
      self.getPartFromFeeder(self.product,
      self.feeder)

      # Wait for a courier to rendezvous
      # with the manipulator for feeding
      partner = self.acceptRendezvous(``Feeding'')

      # and transfer the product to the courier
      self.transferGraspedProduct(partner)

# instantiate the applet
program = Program()
```

Fig. 6. A simple manipulator program.

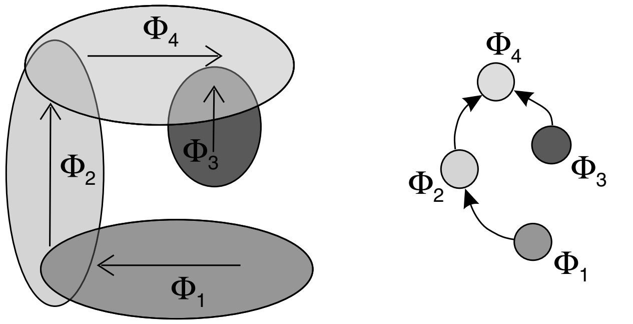


Fig. 7. Example decomposition of a trivial planar configuration space and the associated induced graph relating the control policies.

impact on one another. Specifically, to reduce the complexity associated with writing agent programs, the low-level control system has assumed responsibility for the details associated with switching and sequencing the various control policies available for execution at any one moment.

To simplify the development of agent programs, the process of deciding exactly when and how to switch between low-level control strategies is removed from the agent program and isolated from the high-level semantic negotiations that are the primary domain of the agent program. However, it is important to note that the high-level agent program continues to maintain explicit control over the precise policies that are available for use at any given moment. The fundamental model we use for the execution of control strategies was presented in Rizzi (1998). Briefly, rather than relying on the agent program to generate trajectories through the free configuration space of the agent, the program decomposes the free configuration space into overlapping regions and parameterizing control policies associated with each region. The left side of Figure 7 shows a simplistic cartoon rendering of this approach, where $\Phi_i$ represent the control policies that are guaranteed to safely move any state from anywhere in the associated shaded domain into the domain of the next control policy. A *hybrid control* system is then responsible for switching or sequencing between the control policies associated with this decomposition to achieve a desired overall goal, inducing a monotonically convergent finite state automata over the control policies, such as that depicted on the right of Figure 7.

This scheme describes the behavior of any one agent in terms of a collection of feedback strategies based on the state of the system as perceived by the individual agent. The result is a hybrid online control policy (one that switches between various continuous policies), which makes use of the collection of control policies that have been passed to it by the higher level agent program. By leaving the selection of goals and the associated prioritized decomposition of the state space to the agent program, it remains possible to describe (at the program level) arbitrarily complex behavior without constructing code to undertake the complex real-time management of those behaviors.

Given this model for executing physical action, it remains the responsibility of the agent program (specifically the script defined by its `run` method) to create, parameterize, and manage the currently active set of controllers along with the associated sets of goals and domains. Thus, the script is only responsible for choosing the current overall goal, along with appropriate intermediate subgoals, and providing parameterizations of control strategies to accomplish those goals. The complex and potentially error-prone problem of making real-time changes to the underlying control system is left to the hybrid control system.

The interface between the script and this *controller manager* is quite straightforward. The class from which a particular agent program instance is derived provides standard tools for creating and parameterizing controllers and their associated domains. These resulting controllers are then, at the direction of the script, placed into an ordered list of active controllers. Finally, the controller manager will select the appropriate control policy (from this list) to execute in real time. The details of high-bandwidth monitoring and coordination of an agent and its peers' state are performed by these lower levels, using a dedicated local communications network (AAA-Net) to share information between agents. This local network is used to pass relevant information between agents only about those variables that affect their execution, resulting in efficient utilization of the available communication bandwidth in a manner that is transparent to the agent program.

Communication of progress and completion of tasks back to the script are accomplished by use of either call-back functions or direct polling of the actual state of the agent. In general, the expectation is that scripts will submit a moderately sized list of control actions along with a set of fail-safe and fall-back strategies capable of responding to the most dire circumstances, then *sleep* (wait for a call back) until either progress has been made or a failure has been detected. When appropriate progress has been made, the script will, while motion is still executing, append additional control actions to the top of the active controller list indicating new goals and delete those control actions that are no longer useful. If a failure has been detected, the program will proceed in a similar fashion; only the actions added to the list will most likely attempt to recover from the problem.

By both parameterizing the specific controllers (setting the goal, defining the domain of applicability, specifying gains, etc.) and ordering their placement on the list of active controllers, a script is able to specify complex and efficient physical motion that is fundamentally robust. This provides a rich and expressive method for programs to specify physical motion while reducing the risks associated with writing those programs.

In practice, the details of this interface are hidden from the programmer by a set of standard "convenience functions." For example, the `moveTo(...)` call in Figure 5 would actually expand to the code fragment shown in Figure 8. It is here that a specific resource reservation protocol is implemented to ensure safe operation (Rizzi, Gowdy, and Hollis 1997) and where a standard set of controllers are parameterized and placed on the list of active controllers. In this particular instance, a geometric region in the factory is reserved by the call to `self.reserve`, and a call-back method is registered to execute as the agent enters the destination area. The particular call-back method used here, `Unreserve(...)`, frees the reservation held on the current area as soon as the agent departs it, thus allowing its use by other agents in the system.

```
# submit actions to move from self.current
 to area
def moveTo(area):
  # get the goal at boundary of area
  # and self.current in self.current
  x,y = self.getBoundaryGoal(area)

  # create and submit action
  controller = self.goTo(x,y)
  domain = self.inArea(self.current)
  self.submit(controller, domain)

  # reserve area, blocking if necessary
  self.reserve(area)

  # get goal at boundary of area and
  # self.current in area
  x,y,overlap = self.getOverlapGoal(area)

  # create and submit action to cross into
  # the new area
  self.submit(self.goTo(x,y), self.inRegion
   (overlap))

  # create and submit action to drive to the
  # goal in area
  # note that a callback class is invoked when
  # this action starts which unreserves
   self.current
  self.submit(self.goTo(x,y), self.inArea(area),
  start=Unreserve(self.current))

  # keep track of current area
  self.current = area
```

Fig. 8. Code fragment for `moveTo`.

## 3.4. Agent Interaction

Thus far, we have focused on describing how individual agents can be programmed to accomplish specific tasks. Construction of a useful AAA system, however, requires that agents successfully interact with one another to undertake cooperative behaviors. Not surprisingly, our approach to performing these cooperative actions uses the same protocols and programming constructs we have been describing. The notable difference is that the execution of the specific control strategies may require sharing significant state information between the participating agents for their behavior to be tightly coordinated.

It makes sense to think of the two such cooperating agents as transiently forming an abstract machine consisting of their collective degrees of freedom and a single program dictating the behavior of this abstract machine—a term we will use to describe a collection of agents that are actively coordinating their continuous behavior with one another. Of course, in actuality, each agent will be executing its own program and associated control policies, and it is critical that the participating agents reliably enter into, execute, and dissolve the coordinated behavior. Clearly for this to happen, the agents must arrange to execute low-level control strategies that are compatible—that is, when executed in parallel, they must communicate the appropriate state information to their peer(s) to jointly perform the desired physical operation.

As already described in Section 3.2, our programming system includes primitive tools to perform the necessary semantic negotiation (e.g., the various forms of `rendezvous` statement in Figs. 5 and 6). One simple case, with extremely asymmetric interaction, is a transient master-slave relationship between two agents, with one agent abdicating control of its actuators to the other, which monitors the state of both and passes force and torque commands to its own actuators as well as those of the slave agent via the high-bandwidth AAA-Net. The details of this interaction are illustrated in Figure 9, which shows the expansion of the `accept Product` procedure from Figure 5, and Figure 10, which shows the expansion of the `transferGraspedProduct`

```
def acceptProduct(source):
  # slave to manipulator when it is ready
  controller = self.create(``Slave'')
  controller.master = source
  predicate = self.create(``WatchPartner'')
  predicate.is_grasping = True
  self.insert(ControllerAction(controller,
   predicate))

  # hold position after part placement and
  # manipulator withdrawal
  controller = self.holdPosition()
  predicate = self.create(``WatchPartner'')
  predicate.is_grasping = False
  predicate.min_height = source.offset
  action = ControllerAction(controller, predicate)
  action.addStartCallback(self.tag
   (``PartPlaced''))
  id = self.insert(action)

  # wait until part placed
  self.waitFor(``PartPlaced'')
  # and clean up action list
  self.truncate(id)
```

Fig. 9. Code for `acceptProduct`.

```
def transferGraspedProduct(partner):
  # from the product information and the partner
  # courier attributes calculate the courier
  # position and the manipulator position
  # necessary to place the product on the courier
  (cour_x, cour_y, manip_z, manip_th) = /
     self.findPlacementTransform(partner)

  # submit action to get ready to place
  controller = self.coordinatedMove(cour_x, cour_y,
     manip_z-self.offset, manip_th)
  self.insert(ControllerAction(controller,
     self.isGrasping()))

  # submit action to go down to place product when
  # both courier and manipulator have arrived at
  # pre-placement positions
  controller = self.goto(manip_z, manip_th)
  predicate = self.coordinatedAt(cour_x, cour_y,
     max_z=manip_z-self.offset, manip_th)
  self.insert(ControllerAction(controller,
   predicate))

  # submit action to release part when force
   sensed
  controller = self.releasePart()
  predicate = self.forceThreshold(min_force = 0.5)
  self.insert(ControllerAction(controller,
   predicate))

  # submit action to back off from courier
  action = ControllerAction(
     self.goto(manip_z-self.offset, manip_th),
        self.isNotGrasping())
  action.addStartCallback(``FinishedPlacement'')
  id = self.insert(action)

  # wait for placement to finish
  self.waitFor(``FinishedPlacement'')
  # register transfer
  self.registerTransferTo(partner)
  # clean up action list
  self.truncate(id)
```

Fig. 10. Code for `transferGraspedProduct`.

procedure. The `acceptProduct` method abdicates control of the courier's motions to a manipulator master until a product has been placed on it and the manipulator has backed off by a given offset. The `transferGraspedProduct` method coordinates the motions of the manipulator and courier and places the part it has already grasped onto the courier.

Here, the invocation of `acceptRendezvous` and `initiateRendezvous`, by the manipulator and courier, respectively, has indicated the willingness of these agents to participate in the cooperative behavior, as well as having forced them to synchronize their execution. Immediately following this, a set of control policies are submitted that cause the underlying control systems to synchronize and undertake the desired cooperative behavior. Implicitly, the submission and execution of these control policies (`Slave` and `coordinatedMove`) have created a high-bandwidth communications channel between the two agents. This communication channel is used to enable the supervisory controller manager on each agent to make transitions based on state variables contained within the peer and to allow relevant state and, in this case, command information to pass between the two control algorithms.

Note that throughout this process, the individual controller managers maintain authority over the specific policies that are executed by each agent, and it is only through the submission of a compatible set of policies by both agent programs that the desired behavior is realized. We do not foresee this extremely asymmetric form of cooperation as the typical behavior of agents, but it serves to illustrate the point: typically, the control policy will be segmented between the agents, with each individual computing those terms relevant to its own behavior. The manual specification of these interactions can be a complex process. Fortunately, a significant percentage of the interactions fall into stereotypic classes for which convenience functions can easily be provided.

## 4. Experimental Demonstrations

We have performed two demonstrations of interactions between agents using this framework. The first relies on visual servoing techniques to perform a precision alignment of parts measuring 3 mm.[2] The second uses a custom-built force sensor, with a resolution of roughly 24 mN ($1\sigma$), to perform force-guided insertion tasks.

### *4.1. Visually Guided Cooperation*

For our initial investigation of visually guided coordination in the minifactory environment, we have chosen to undertake a simplified position regulation task. The courier agent carries a subassembly of a small medical device measuring 3 mm on a side, and the manipulator agent observes this device through its vision system. The task is to keep the subassembly centered within the field of view of the camera while the manipulator undergoes a rotation. The only information transacted between the two agents is commands from the manipulator indicating the velocity at which the courier should move, based on the manipulator's visual observations.

Figure 11 shows the structure of the communication channels between the two agents and their computational processes after execution of a `rendezvous` by the scripts for the two agents—in this case, the `rendezvous` between the two agents has initiated communication and spawned additional control processes to guide their interaction. In this diagram, circles represent processes, and the two separate shaded areas represent the courier and manipulator agents. Internal communication between processes on the individual agents uses shared-memory structures, providing fast data sharing. For the visually guided coordination task, the manipulator uses three processes, in addition to the AAA-Net process. The first (Executor) executes low-level control, the second (Head) interprets user-level commands and scripts, and the third (Vision) analyzes images from the frame grabber. The courier has a similar structure but lacks the vision system and associated processes.

#### *4.1.1. Image Processing*

To accomplish the visually guided coordination described above, the manipulator, which manages the camera and frame grabber, must perform all the image-processing tasks. The camera system contained in the end effector has a monochrome CCD camera and an adjustable two-lens optic. The resulting magnification of this system was set at 0.75 so that one pixel corresponded to approximately 10 $\mu$m of motion. Tsai's (1986) coplanar method was used to calibrate the intrinsic parameters of the camera system, as well as the extrinsic parameters. Wilson's implementation was used to numerically optimize the parameter values from measurements of a calibration fixture (see www.cs.cmu.edu/gw/Tsai-method-v3.0b3.tar.Z).

Performing visual servoing requires both global and local feature localization strategies. The global scheme finds the top corners of the subassembly without any initial position estimates, while the local scheme tracks the subassembly at the field rate given recent position estimates. The global search begins by locating the centroid of the subassembly's image, providing an initial estimate for the location of the part. Performing a Hough transform eliminates pixel noise and allows the use of linear regression techniques to recover the location of lines representing the part edges. As seen in Figure 12, the intersection of the lines provides accurate positions of the top two corners of the device.

Given these estimates of corner locations, local image-processing routines can track the corners at the field rate (60 Hz). We use the XVision package (Hager and Toyama

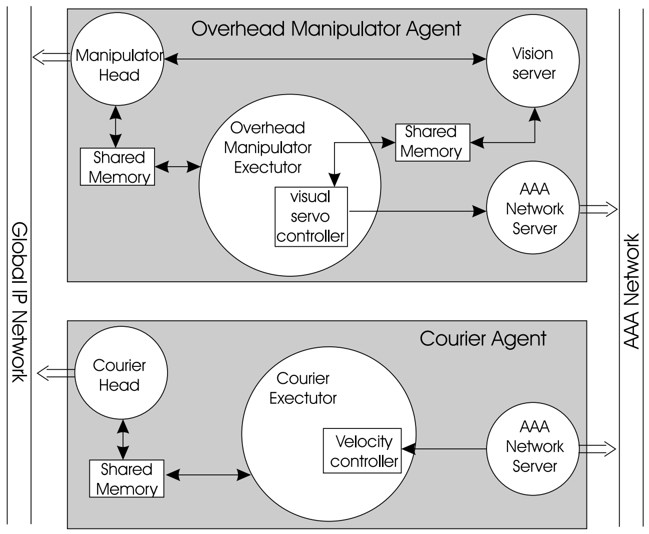


Fig. 11. Communications within and between manipulator and courier agents during coordination.

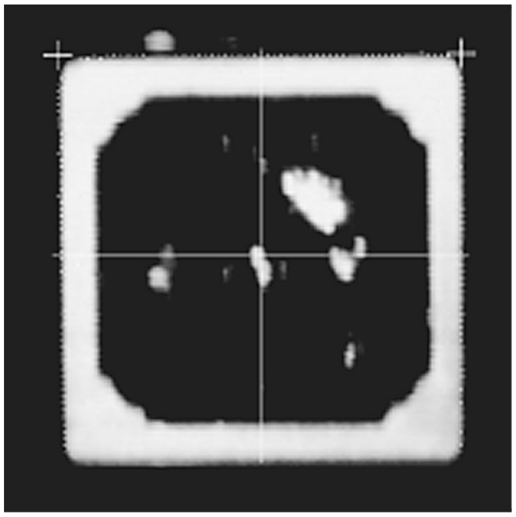


Fig. 12. Results of global search for the top corners of the subassembly—the recovered center and corner locations are marked.

1998) to perform these tracking tasks. The flexible nature of XVision allows programming constraints between the two corner trackers we used. This improves robustness of the system by allowing it to recover from transient occlusion events. For example, if one corner tracker fails, information from the other corner's position can be used to reinitialize the failed tracker. Once the occlusion passes, the failed tracker can relocate the appropriate corner.

### 4.1.2. *Visually Guided Control*

Visually guided control was accomplished by processing image information in the manipulator agent and sending velocity commands to the courier agent and its controller. The vision server, as shown in Figure 11, tracks the position of the subassembly at the field rate and in turn passes the image plane position information to the visual servo controller running as part of the executor process. The visual servo controller computes a desired velocity for the courier to keep the subassembly at the desired position in the image plane. These commands are finally sent to the courier controller at the field rate (60 Hz) via the AAA-Net.

A simplified model of the visual servoing control system is shown in Figure 13. This depiction emphasizes the three main components of the system: the visual servo controller, the courier controller, and the courier motor. The visual servo controller is implemented on the manipulator agent and can be classified as an image-based visual servo controller since control values are directly computed from image features (Hutchinson, Hager, and Corke 1996). In the block diagram, H encapsulates the camera and the image-processing routines used to locate and track the subassembly within an image. The block labeled $G_c$ represents the adjustable gains of the visual servoing controller—for improved performance, this includes a proportional term, as well as an integral term, which both operate on the error between the desired and current image plane position of the subassembly. The block labeled J represents the image Jacobian and depends on the rigid transform between the camera coordinate frame and the courier coordinate frame. Figure 14 shows our convention for frame placement in minifactory. Note that the frames attached to the courier and end effector depend on the agents' current positions and are constantly recalculated, while the remaining frames are precisely located during the factory self-calibration process.

The remaining major components of the simplified visual servo system model represent the courier and its controller. The dynamics of the courier motor are approximated by a double integrator, and the courier controller is replaced by a simplified proportional-derivative control scheme that has been modified to accept velocity commands from the manipulator agent. The details of courier behavior are beyond the scope of this paper and can be found in Quaid and Hollis (1998).

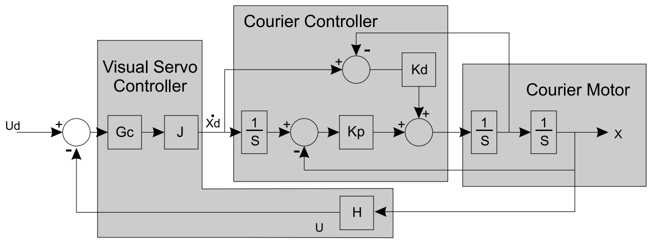
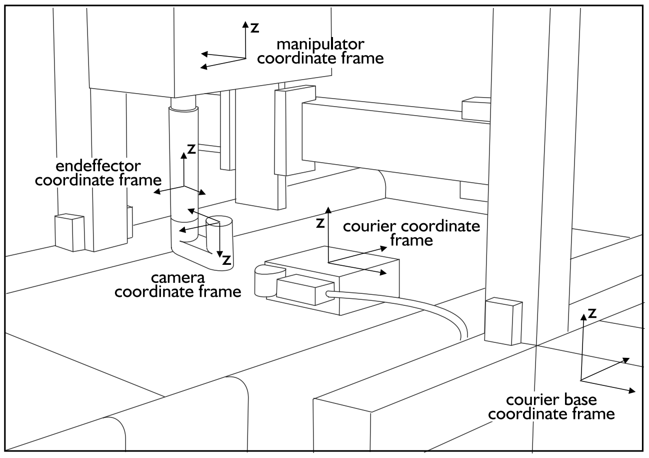


Fig. 13. Simplified model of visual servoing control system.


z
manipulator coordinate frame
endeffector coordinate frame
z
z
camera coordinate frame
z
courier coordinate frame
z
courier base coordinate frame
Fig. 14. Minifactory coordinate frames.

### 4.1.3. *Results*

The visually guided coordination experiment was composed of four major steps: automatic calibration of the factory, presentation of a subassembly to the manipulator, initialization of the visual servoing system, and movement of the manipulator's $\theta$-axis as a disturbance input. Calibrating the factory provided precise coordinate transforms between various parts of the minifactory, as shown in Figure 14.

After calibration, the courier moved to present a subassembly to the manipulator. An image was acquired and searched to initialize a pair of corner trackers. To guarantee that the trackers had time to settle, both agents were prevented from moving for 1 second. Figure 15 shows that the desired and measured positions of the subassembly, as measured by the vision system during a typical experiment, were different at this point in the experiment. While keeping the manipulator fixed, the courier was allowed to move, and as can be seen, the initial error in the position of the subassembly was quickly accommodated.

After the vision system was initialized, the manipulator's $\theta$-axis was rotated clockwise at a constant rate of 0.052 rad/s. This served as a disturbance to the visual servoing system starting at 2 seconds. Table 2 shows mean and standard

**Table 2. Steady-State Image Plane Position Error for the Given Visual Servo Controller: Proportional and Integral Gains**

| Gains $G_c$ | | $u$ Error (mm) | | $v$ Error (mm) | |
|---|---|---|---|---|---|
| | | | Standard | | Standard |
| $K_p$ | $K_i$ | Mean | Deviation | Mean | Deviation |
| 7.0 | 0.0 | −0.286 | 0.012 | 0.007 | 0.013 |
| 7.0 | 0.005 | 0.000 | 0.013 | 0.001 | 0.014 |
| 12.0 | 0.0 | −0.167 | 0.017 | 0.005 | 0.026 |

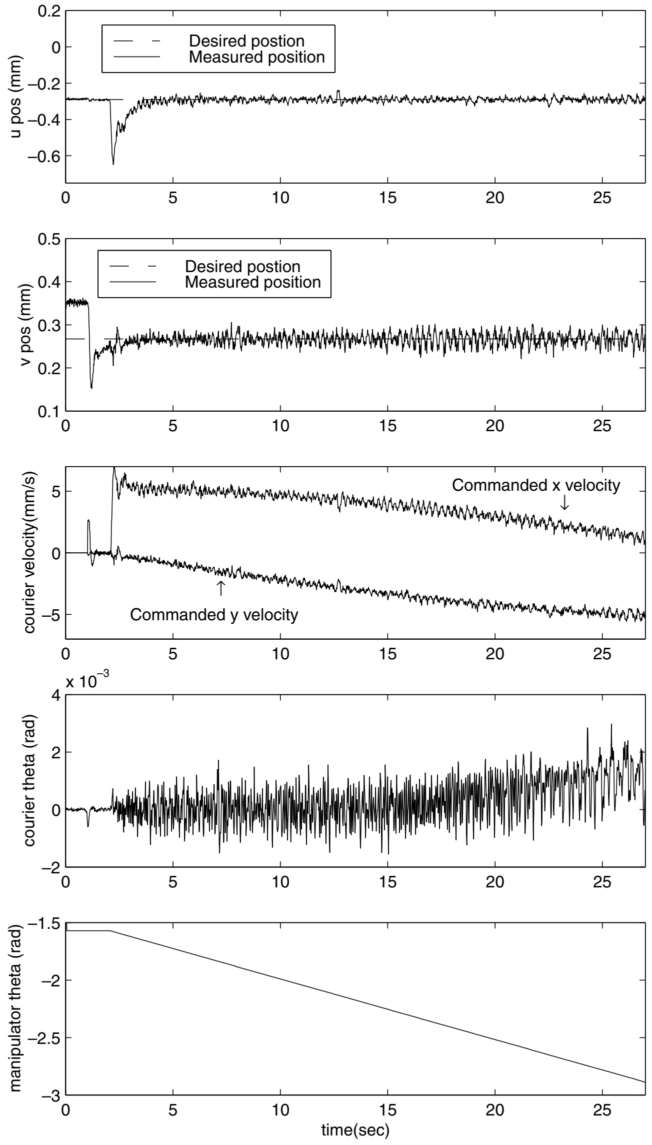


Fig. 15. Visual servoing results for $K_p = 7.0$ and $K_i = 0.005$, including measured image plane location of the subassembly ($u$ and $v$), commanded courier velocities ($\dot{x}$ and $\dot{y}$), and measured angular position of the courier and manipulator.

deviation of the visual error signals (both $u$, lateral, and $v$, vertical, directions) for three different settings of the proportional and integral gains. Note that with the integral term disabled ($K_i = 0$), the resulting $u$-axis error was much greater than the $v$-axis error (note that 1 pixel corresponds to roughly 0.01 mm). The difference in error magnitude between the axes is a direct result of the camera configuration, since $\theta$ motion of the manipulator maps directly into a $u$ motion on the image plane. Setting $K_i = 0.005$ significantly reduced the error during motion, as shown in Figure 15 and Table 2, at the cost of slightly slower transient performance.

The overall results shown in Figure 15 were encouraging. The manipulator's $\theta$-axis rotation caused the courier's tangential velocity to be approximately 5 mm/s, while image plane measurements showed a standard deviation of less than 15 $\mu$m in both axes of the vision system. However, the peak-to-peak error was approximately 0.04 mm in the $u$-axis and 0.05 mm in the $v$-axis, corresponding to image movements of nearly 5 pixels. The courier angle plot in Figure 15 provides one possible cause for this problem. As can be seen, the recorded peak-to-peak motion was approximately 0.002 rad, even though the courier was commanded to hold its orientation throughout this experiment. This "wobbling" is believed to result from miscalibration of the courier position sensor (Butler, Rizzi, and Hollis 1998) and contributed to most of the error.

### 4.2. *Force-Based Interagent Coordination*

To explore the applicability of force-guided coordination in the minifactory environment, we chose to undertake a collection of insertion ("peg-in-hole") tasks. In our sample task, the courier robot carried a plate bearing a chamfered hole, and a manipulator observed forces on a peg as it was inserted into the hole. By performing a `rendezvous` operation, the two agents were able to exchange state, sensor, and command information with one another and act as a single 4-DOF device to reliably perform the insertion task.

The details of the particular force sensor carried by the manipulator (shown in Fig. 17) were described in DeLuca, Rizzi, and Hollis (2000).

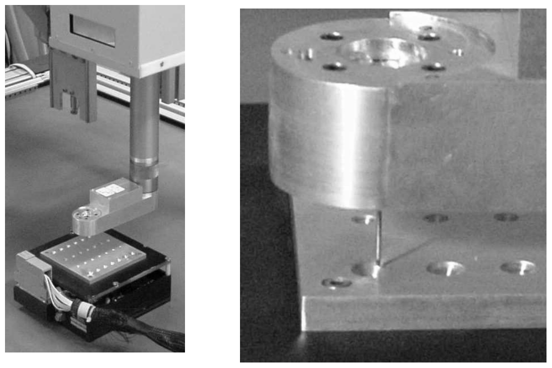


Fig. 16. (a) +View of courier (lower agent), configured for an insertion task, collaborating with an overhead manipulator (upper agent). (b) Close-up view of the manipulator's end effector preparing to insert a peg into a hole.

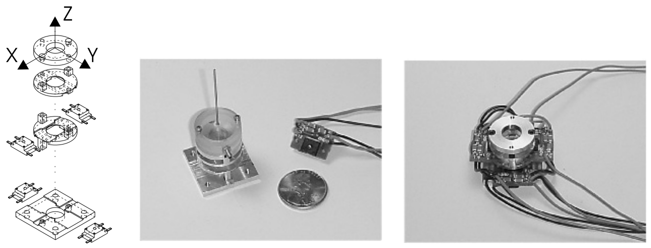


Fig. 17. Three-axis force sensor: (a) exploded assembly drawing of the flexure and single-axis load cells, (b) photograph of the flexure with clear Lexan$^{TM}$ vacuum chamber and one load cell, and (c) flexure with four load cells in place.

### 4.2.1. Force-Guided Control

We have undertaken a number of experiments to evaluate the performance of the distributed agent pair performing this task. To accomplish this, we have deployed a control architecture that minimizes interagent communication by making use of algorithmically simple control schemes. Impedance control provides one such simple class of control policies and accomplishes stable interaction with an environment by converting the system to a form that naturally performs the task (Hogan 1985; Anderson and Spong 1988; Chiaverini, Siciliano, and Villani 1999).

To deploy an impedance control scheme, we begin by specifying the desired behavior of the system. Given a desired mass ($M_d$), stiffness ($K_d$), damping ($B_d$), and force ($F_d$), the desired system behavior is given by

$$\begin{aligned} &M_d\ddot{q} + K_d(q - q_0) + B_d\dot{q} + \\ &G_i \int_0^t (F_e - F_d)d\tau = F_e, \end{aligned} \tag{1}$$

where $q$ represents the generalized configuration of the system and $F_e$ the applied environmental force acting on the system. This system behaves like a mass attached to a spring and damper about a nominal target $q_0$, with the added integral term driving the system to the desired contact forces ($F_d$).

To realize the behavior defined by (1), we developed distributed control policies for the two independent robotic agents. The courier can be modeled as a mass with essentially ideal actuators in the $x$ and $y$ directions. This is a direct result of the simple actuator mechanism and the frictionless nature of the air bearing that supports the courier (Quaid and Hollis 1998; Sawyer 1973). The overhead manipulator's $\theta$-axis is directly driven, resulting in negligible friction and an equally simple model. The $z$-axis is driven by a ball screw so a friction term was included to offset the effects of friction in that axis. Thus, the dynamic model of the system takes the form

$$M_a\ddot{q} + B_a\dot{q} + f(\dot{q}) = \tau_a - F_e, \tag{2}$$

where $M_a$ and $B_a$ are $4 \times 4$ diagonal matrices that describe the overhead manipulator's and courier's mass and damping parameters, $f(\dot{q})$ contains the friction terms, $\tau_a$ represents the applied actuator forces, and $F_e$ is the applied environmental forces. It is this last term that will be measured by the force sensor mounted on the manipulator's end effector (DeLuca, Rizzi, and Hollis 2000). Given the system model and desired impedance, applying inverse dynamics yields a control law of the form

$$\begin{aligned} \tau_a = \quad & M_a M_d^{-1}[F_e + K_d(q_0 - q) - B_d\dot{q} + \\ & G_i \int_0^t (F_e - F_d)d\tau] + B_a\dot{q} + f(\dot{q}) + F_e. \end{aligned} \tag{3}$$

This control law is implemented separately on the two agents, with each agent responsible for its actuated degrees of freedom. By choosing to only allow diagonal matrices for $M_d$, $K_d$, $B_d$, and $G_i$, this control policy is completely diagonal, which implies that the only information that must be shared between the agents are the sensed environmental forces.

Critical to implementing the above control law is the communication infrastructure described in Section 3.1.4. Messages sent to the courier from the overhead manipulator at real-time rates (roughly 500 Hz) contain force sensor information, velocity commands, and controller mode. The velocity commands and controller mode information allow the manipulator agent to command a variety of different behaviors from the courier agent.

### 4.2.2. Results

In the experiments described, the overhead manipulator performs contact tasks with a 0.813 mm (0.032 in.) diameter hypodermic tube attached to the force sensor (see Figs. 16b and 17b). Mounted on the courier is a plate containing several sets of holes 6.35 mm (0.25 in.) in depth and ranging

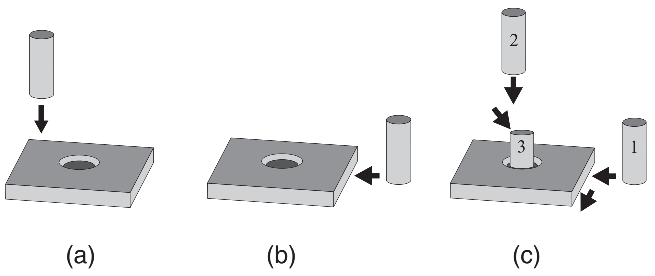
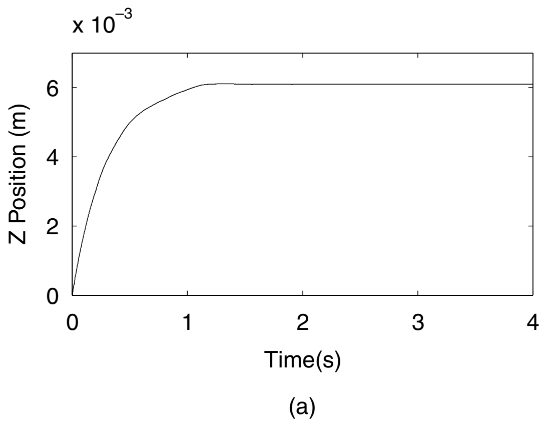


Fig. 18. Graphic depiction of the three classes of experiments performed: (a) vertical contact, (b) lateral contact, and (c) peg-in-hole.

in diameter from 2.54 mm (0.1 in.) to 0.838 mm (0.033 in.) (see Fig. 16). Each hole has a 45° chamfer. Three types of tasks were performed to characterize overall system performance: vertical contact, lateral contact, and peg-in-hole insertions (see Fig. 18). In addition, repeatability and reliability experiments were performed to verify overall system performance.

*Vertical contact*. The vertical contact experiment involved the courier maintaining a fixed position, while the overhead manipulator made contact and maintained a constant force with the plate. Specifically, the courier performed a move (under position control) to place it under the manipulator and hold its position. Meanwhile, the manipulator found the top of the plate on the courier by executing a constant-velocity, force-guarded move. With the exact position of the plate top registered, the manipulator servoed to a $z$ position above the plate by a height equal to the depth of a hole. Once the tool tip had arrived at this position, an impedance controller was activated with the desired position located appropriately below the surface of the plate to obtain the desired contact force at equilibrium. For this experiment, the desired $z$ force was $-1$ N. Results from a typical experiment are shown in Figure 19. Note the slightly underdamped response of the $z$ force and the steady-state value of $-1$ N. The observed high-frequency noise in the force information is attributed to unmodeled dynamics in the gripper tube. The settling time from impact is approximately 1 second. Response rates faster than this were difficult to achieve without higher impact forces. Friction in $z$ appears to be the limiting factor.

*Lateral contact.* The goal of the lateral contact experiment was to position the tool tip below the plane of the top of the plate while the courier made contact and maintained a constant lateral force with the side of the plate. The top of the plate was located with a constant-velocity, force-guarded move as in the vertical contact experiment. Controllers were then deployed to reposition the courier and bring it into lateral contact with the tool tip. When contact was made, the manipulator executed an impedance controller, which in turn issued desired position commands to the courier controller.



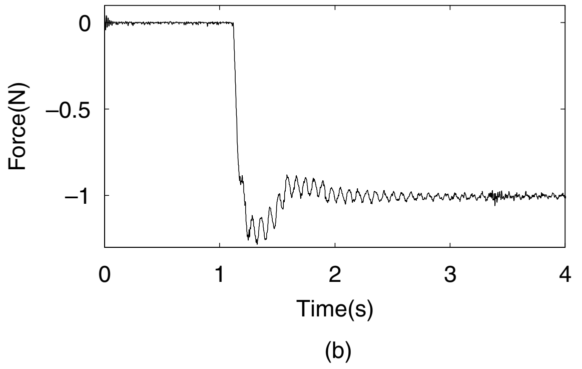


Fig. 19. Position (a) and force (b) measurements during vertical contact.

The desired contact force was set to $-0.4$ N in $y$. Results of a typical experiment can be found in Figure 20. Notice that the force applied in the $y$ direction reaches a steady-state value of $-0.4$ N. A nonzero steady-state value for the $x$ force can be attributed to small misalignments of the gripper tube, misalignments of the courier plate, or compliance in the manipulator's $\theta$-axis. The observed settling time was roughly 0.3 s. It is expected that faster settling times can be achieved for lateral contact in which the stiffness of the $\theta$-axis of the manipulator is increased.

*Peg-in-Hole*. The peg-in-hole experiment consisted of the courier bumping into and sliding along the manipulator gripper to register the plate corner and thus the entire plate geometry relative to the manipulator's tool tip. The initial bump-and-slide maneuver involved a hybrid position-force control sequence implemented through the use of the described impedance controllers. Once lateral contact was made, the manipulator commanded a velocity and a mode change to the courier such that it maintained a constant force and slid along the plate edge. When the courier lost contact with the gripper, the courier's position was noted immediately, and the plate geometry was registered to determine the location of the hole in tool tip coordinates. The plate top location was then

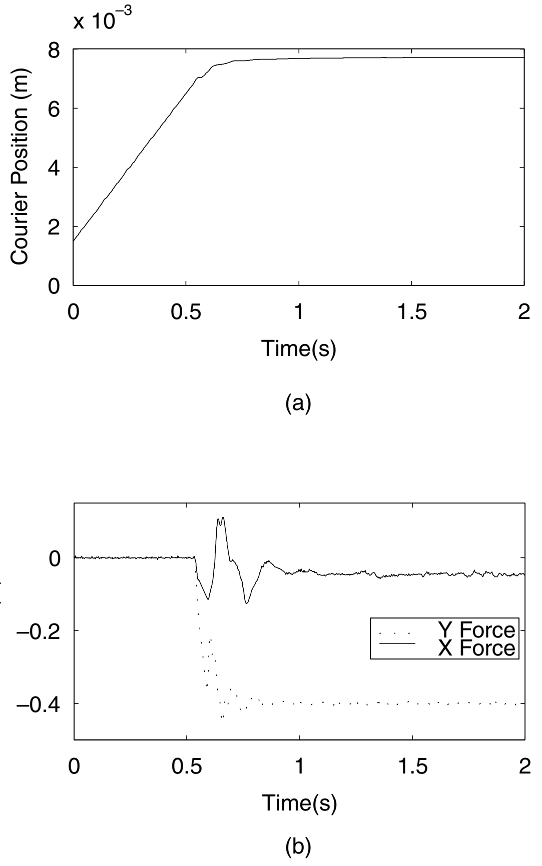


Fig. 20. Position (a) and force (b) measurements during lateral contact.

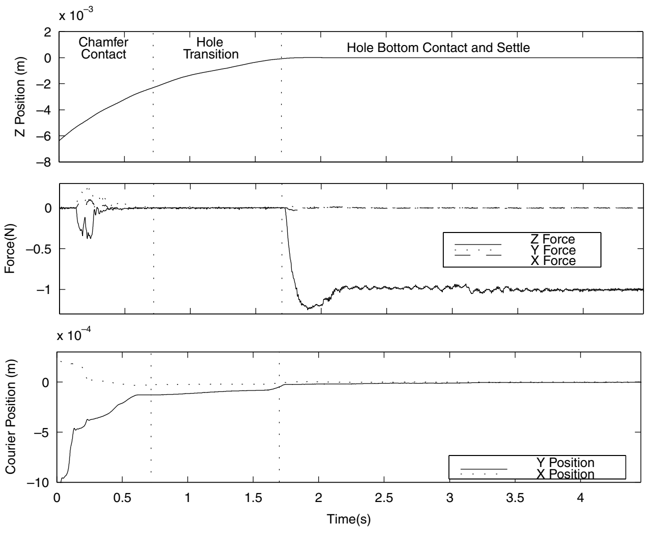


Fig. 21. Overhead manipulator position, force, and courier position measurements during an insertion experiment.

calibrated as described earlier, and the courier was positioned so that the gripper was directly over a chamfer of a hole. The manipulator was servoed to a position at the height of the plate top, at which point an impedance controller was activated with a desired position located at the center and appropriately below the bottom of the hole. The desired force was $-1$ N in $z$ and 0 N in $x$ and $y$. Figure 21 shows typical results for this type of experiment. The insertion data presented are from an insertion performed on a hole with an exaggerated chamfer of diameter 6.35 mm (0.250 in.). The oversized chamfer was used to produce data with longer chamfer contact regions to aid in analysis. The exaggerated chamfer explains the lengthy duration of the insertion event. During chamfer contact, one can see that the tool tip follows the chamfer down into the hole as the courier moves in response to the $x$ and $y$ forces. The settling time for the $z$ force is comparable to that of the vertical contact experiment.

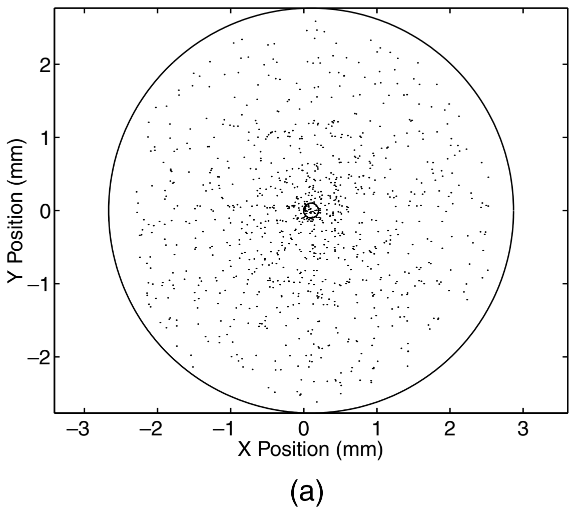
*Repeatability*. Results are presented from two repeatability tests in which insertion tasks were initiated from uniformly distributed random points located above the chamfer. Repeatability was tested on a 1.016 mm (0.04 in.) hole with an exaggerated chamfer of diameter 6.35 mm (0.25 in.) and a 0.838 mm (0.033 in.) hole with a typical chamfer of diameter 1.52 mm (0.06 in.). Figure 22 shows the start and stop positions of the tool tip in configuration space from a typical experiment. Finish positions shown outside of the hole in Figure 22b are attributed to slight system miscalibration and our failure to account for small rotations of the courier. Table 3 summarizes these experiments. The difference in average insertion time is due to differences in the size of the chamfers. It is important to note that although only 93% of the 0.838 mm (0.033 in.) hole insertion attempts made it successfully into the hole, 63 of the failed attempts are attributed to system-level errors. Discounting system-level failures that prevented the experiment from beginning, the overall success rate rises to 99.3% for the 0.838 mm (0.033 in.) holes. Smaller clearance holes present a higher chance of wedging, which explains the larger RMS error in $z$ force for the smaller clearance hole.

## 5. Conclusion

This AAA project began in November 1995. Over the past 5 years, we have concentrated most of our efforts on developing the engineering technologies needed to build the prototype minifactory and the software infrastructures required to design and program the resulting systems. This has led us well into the tasks of developing comprehensive environments for modeling, simulation, and programming. A prototype 3-D interactive user interface package has been implemented that allows the user to look at a detailed running minifactory simulation and interact with the agents. These same tools can be synchronized with agents operating in our laboratory to serve as a "virtual window" on an operating factory system (Gowdy and Butler 1999).

**Table 3. Repeatability Experiment Results**

| Hole Clearance | Average Time | Attempts | Success | RMS $z$ Force Error |
|---|---|---|---|---|
| 0.2032 mm | 2.43 s | 1000 | 100% | 0.0663 N |
| 0.0254 mm | 2.36 s | 1000 | 93% | 0.1091 N |



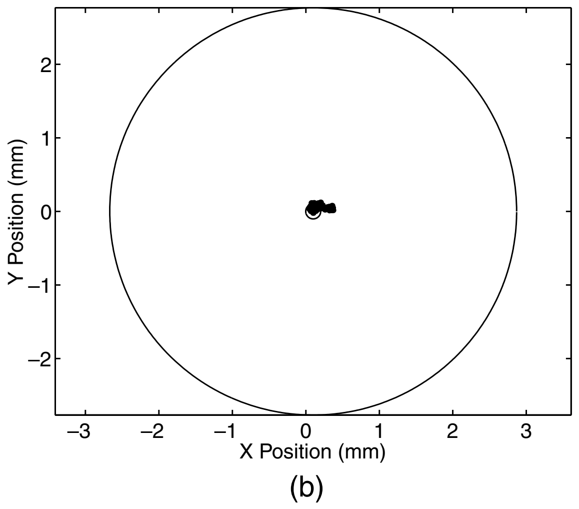


Fig. 22. Tool tip start (a) and finish (b) positions plotted in configuration space for a repeatability test involving a hole with 0.02032 mm (0.008 in.) clearance.

Looking to the future, we foresee AAA and minifactory serving both as a research test bed and as an exemplar of one potential path for the future of automated assembly systems. Whereas our focus has been on rapidly reconfigurable assembly systems for precision assembly, this approach may also have utility for agile parts fabrication, chemical synthesis, pharmaceutical manufacturing, and other such applications.

We are encouraged by the positive response received to date from our colleagues in both the research and the industrial communities. In particular, it would seem the notions of increasing both intelligence and autonomy through the use of well-crafted modular building blocks to realize rapid deployment is very attractive—provided that it can be made to work in the real world and that it can provide realizable benefits in the marketplace.

## Acknowledgments

This work was supported in part by NSF through grants DMI-9523156 and CDA-9503992. The authors would like to thank all the current and former members of the Microdynamic Systems Laboratory, particularly Ben Brown, Zack Butler, Mike Chen, Rich DeLuca, Jimmy Ma, Patrick Muir, Arthur Quaid, and Maria Sensi for their invaluable work on the project and support for this paper.

## References

Anderson, R. J., and Spong, M. W. 1988. Hybrid impedance control of robotic manipulators. *IEEE Journal of Robotics and Automation* 4(5):549–556.

Berry, G., and Gonthier, G. 1992. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming* 19:87–152.

Brooks, R., and Stein, L. 1994. Building brains for bodies. *Autonomous Robots* 1(1):7–25.

Brown, H. B., Muir, P. M., Rizzi, A. A., Sensi, M. C., and Hollis, R. L. 2001. A precision manipulator module for assembly in a minifactory environment. *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, HI, October-November, pp. 1030–1035.

Brussel, H. V., Bongaerts, L., Wyns, J., Valckenaers, P., and Van Ginderachter, T. 1999. A conceptual framework for holonic manufacturing: Identification of manufacturing holons. *Journal of Manufacturing Systems* 18(1):35–52.

Burridge, R. R., Rizzi, A. A., and Koditschek, D. E. 1999. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research* 18:534–555.

Butler, Z. J. 2000. Distributed coberage of rectilinear environments. Ph.D. thesis, Carnegie Mellon University.

Butler, Z. J., Rizzi, A. A., and Hollis, R. L. 1998. Precision integrated 3-DOF position sensor for planar linear motors. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3109–3114.

Chiaverini, S., Siciliano, B., and Villani, L. 1999. A survey of robot interaction control schemes with experimental comparison. *IEEE Transactions on Mechatronics*, pp. 273–285.

Christensen, J. H. 1994. Holonic manufacturing systems: Initial architecture and standards directions. *Proceedings of the First European Conference on Holonic Manufacturing Systems*.

Craig, J. J. 1997. Simulation-based robot cell design in Adept-Rapid. *Proceedings IEEE International Conference on Robotics and Automation*, pp. 3214–3219.

DeLuca, R. T., Rizzi, A. A., and Hollis, R. L. 2000. Force-based interaction for distributed precision assembly. In *Experimental Robotics VII*, ed. D. Russ and S. Singh, 141–150. New York: Springer-Verlag.

Foslien, W., and Nibbe, V. 1990. A robotic workcell for small-batch assembly. *Robotics Today* 3(2):1–5.

Gertz, M. W., Stewart, D. B., Nelson, B. J., and Khosla, P. K. 1994. Using hypermedia and reconfigurable software assembly to support virtual laboratories and factories. *Proceedings of the 5th International Symposium on Robotics and Manufacturing*, Maui, Hawaii, August.

Goldberg, K., Mascha, M., Gentner, S., Rothenberg, N., Sutter, C., and Wiegley, J. 1995. Desktop teleoperation via the World Wide Web. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Gowdy, J., and Butler, Z. J. 1999. An integrated interface tool for the architecture for agile assembly. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3097–3102.

Gowdy, J., and Rizzi, A. A. 1999. Programming in the architecture for agile assembly. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3103–3108.

Hager, G. D., and Toyama, K. 1998. X Vision: A portable substrate for real-time vision applications. *Computer Vision and Image Understanding* 69:23–37.

Harrigan, R. W. 1993. Automating the operation of robots in hazardous environments. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1211–1219, Yokohama, Japan, July.

Hogan, N. 1985. Impedance control: An approach to manipulation: Part I—theory, part II—implementation, part III—applications. *Journal of Dynamic Systems, Measurements, and Control* 107:1–24.

Hollis, R. L., and Quaid, A. 1995. An architecture for agile assembly. *Proceedings of the American Society of Precision Engineering*, Austin, TX, October.

Hutchinson, S., Hager, G. D., and Corke, P. I. 1996. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation* 12:651–670.

IEEE 802.3u-1995 10 & 100 Mb/s management. 1995. Section 30. Supplement to IEEE std. 802.3.

Koestler, A. 1967. *The Ghost in the Machine*. London: Arkana.

Kume, S., and Rizzi, A. A. 2001. A high-performance network infrastructure and protocols for distributed automation. *Proceedings 2001 ICRA, IEEE International Conference on Robotics and Automation*, Seoul, South Korea, May, pp. 3121–3126.

Lenat, D. 1995. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM* 38:32–38.

Lozano-Perez, T. 1983. Robot programming. *Proceedings of IEEE* 71(7):821–841.

Minsky, M. 1986. *The Society of Mind*. New York: Simon & Schuster.

Musliner, D. J., Durfee, E. H., and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74:83–127.

Postel, J. 1981. Internet protocol: DARPA Internet program protocol specification—RFC 791. Technical report, USC/Information Sciences Institute.

Quaid, A., and Hollis, R. L. 1996. Cooperative 2-DOF robots for precision assembly. *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, MN, May.

Quaid, A. E., and Hollis, R. L. 1998. 3-DOF closed-loop control for planar linear motors. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2488–2493.

Rizzi, A. A. 1998. Hybrid control as a method for robot motion programming. *IEEE International Conference on Robotics and Automation*, pp. 832–837.

Rizzi, A. A., Gowdy, J., and Hollis, R. L. 1997. Agile assembly architecture: An agent-based approach to modular precision assembly systems. *IEEE International Conference on Robotics and Automation* 2:1511–1516.

Sawyer, B. A. 1973. Linear magnetic drive system. U.S. patent 3,735,231.

Tsai, R. Y. 1986. An efficient and accurate camera calibration technique for 3D machine vision. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 364–374.

van Rossum, G. 1998. *Python Tutorial*. Reston, VA: Corporation for National Research Initiatives.