A Service Paradigm for Reconfigurable Agents*

Gary Holness Deepak Karuppiah Subramanya Uppala Roderic Grupen
Laboratory for Perceptual Robotics

S. Chandu Ravela
Center for Intelligent Information Retrieval

Department of Computer Science
University of Massachusetts, Amherst, MA 01003 USA
{gholness, deepak, uppala, ravela, grupen}@cs.umass.edu

ABSTRACT

Applications of multiple processors embedded in the systems involved with entertainment, informatics, climate control, communication, transportation, and food preparation are already commonplace. A network of these embedded processors presents application development challenges since the state space grows exponentially as new devices attach to the network. The programming model for such systems will need to change if reliable systems are to be realized. By observing information about sensorimotor activity, such systems can gather information useful to programming the network. Through such interaction, a network can build hierarchies of shareable computational structures for representing various activities. This application domain presents additional challenges due to disparate sensor and actuator bandwidth, hardware disparities and partial system failure. We present an architecture that fuses sensing and control with the Jini Network Technology in a distributed sensorimotor network. This network can grow dynamically over time and facilitates the automatic creation of persistent data structures.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

Keywords

Distributed Objects, Dynamic Service Discovery, Ubiquitous Computing, Reconfigurable Agents, Networked Sensory-Motor systems

*The authors gratefully acknowledge support for this work from NSF CDA-9703217 (Infrastructure), DARPA/ITO DABT63-99-1-0022 (SDR), and DARPA/ITO DABT63-99-1-0004 (MARS)

1. INTRODUCTION

A wide range of applications, from climate control to navigation employ an ever increasing number of sensorimotor subsystems [10, 4]. The premise behind this approach is that multiple distributed sensorimotor components provide a robust basis for interacting with the environment. For example, large physical environments such as buildings can benefit from networks of articulated sensors and robotic agents that interact with human users to provide a plethora of services including maintenance, navigation, security, monitoring, hazard-recovery and communication. While constructing agents that reason within such environments is an active area of research, significant issues still need to be addressed.

The first issue is complexity. Computational processes for inference must be able to handle an increasing variety of sensors and motors. The design and use of custom device interfaces yields agents that are neither flexible nor reusable and can be expected to scale poorly.

The second issue is one of fault-tolerance. An environment containing multiple sensorimotor systems is subject to various failure modes. This includes failure in the hardware site, network link, or in the algorithms attached to the devices. Since the environment is instrumented with multiple (possibly redundant) devices, it is possible to switch devices or change process modes when devices and associated algorithms fail. Fault-tolerance for a distributed computation is not feasible without a reliable architecture for detecting and reporting such faults[14].

The third issue is one of timing and synchronization. In general, an agent must process observations and generate control commands with heterogeneity in sensors, signal bandwidths, dynamic range, features and control actions. Any inference regarding the environment or internal state fundamentally relies on synchronization between disparate device timings in the network. While there are several approaches to synchronization, it is important that such mechanisms neither be ad-hoc nor embedded somewhere deep within the agent's architecture. That is, there has to be a consistent notion of an event's occurrence across the network.

This paper presents an architecture for constructing flexible agents from networks of sensor and motor devices. The

proposed paradigm is based on a service model. In this model, physical devices adopt a consistent interface and present themselves as services to the network. These service interfaces expose failure modes in a consistent manner and maintain a means of publishing information to clients at necessary rates. This model is recursive in that higher level services can be constructed from other services in addition to those grounded directly in physical devices. In addition to having the same interface properties, higher level services have a built in notion of fault tolerance. When parameterized with a set of services, an algorithm, and a state machine specification whose transitions are driven by an objective function, higher level services change modes (if necessary) to maintain the property of the system for which they are constructed.

The proposed service architecture uses Java to develop consistent service interfaces. The network time protocol and first order extrapolation methods are used to achieve synchronization. The Jini network technology is used to dynamically post and discover services across a network, as well as to establish interfaces between the service site (where the device is) and the agent[16]. This core substrate also facilitates fault reporting by combining structured exception handling with a periodic data push model.

In this paper, we present the service architecture and demonstrate its utility in an example agent for tracking a human subject by orchestrating multiple sensorimotor services. This system is tolerant to multiple physical faults and run-time context changes. In this example, the agent maintains a tracking task using the best service resources as determined by an objective function for network reachability and algorithm performance criteria. Objective functions optimizing for other metrics such as minimum algorithmic complexity, energy cost, minimum response time, and quality are also possible.

The remainder of this paper is organized as follows. In Section 2 the proposed architecture is described. In Section 3 implementation and run-time performance for the tracking application are discussed.

2. THE ARCHITECTURE

Situated agents, interacting with the physical world through their sensors and motors, garner information integrated over time to form the basis of their understanding of the world [8]. We factor an agent into basic parts including an algorithm, an objective function, sensors and motors. The agent relies on the services of its sensors and motors in order to perceive and act in its environment. Whether coupled tightly through a system backplane or coupled loosely through a network, the agent makes use of its sensors and motors through a communication medium. To the agent, its sensors and motors are services accessed through a communication network.

A sensor or motor may be co-located with and share a physical interface with an embedded processor. The processor executes a low level driver that samples raw signal data or produces control signal data. The processor also executes an arbiter that interacts with the driver and the communication medium. Through the arbiter, a client poses requests to the sensors and motors. If the arbiter exports sensorimotor interfaces and is the point of contact on the network, it becomes the sensorimotor service. This approach follows the tenets of object-oriented systems in that a structure exports an interface whose implementation details are hidden from clients. This provides an elegant mechanism for system decomposition into flexible reusable components which may be shared across the network. Moreover, the implementation details within the component include the protocol for contacting the sensorimotor device. This frees clients from embedding custom protocols.

A higher level sensorimotor service can be assembled by combining a set of sensorimotor services. The resulting service appears identical to every other sensorimotor service. Repeated construction results in a hierarchy of services. This gives rise to a network of sensors and motors which may be reconfigured dynamically [9] by fitting together various components and adding computation. An agent endowed with sensorimotor services can also mode change across different services in its hierarchy or change its repertoire by adding or deleting services.

In dealing with systems that export objects across a network, the distinction between local and distributed computation is crucial [14]. A distributed computation consists of a number of nodes connected through a network which must collaborate or reach consensus by exchanging messages. Such systems may fail on account of node failure or link failure. These failures may be transient or may persist over time. The Jini Network technology is a distributed computing infrastructure well suited for building reliable distributed systems which deal with partial failures [16].

Also in a distributed system, a computation must deal with issues of clock skew and synchronization. This is important if the understanding of an event's occurrence is to be kept consistent across the network. The Network Time Protocol (NTP) is a simple mechanism useful for nodes on a network to synchronize their clocks [13, 15].

2.1 Sensor Service

Figure 1 depicts the processes that a service attaches to a physical sensor. Raw signal information is passed through a bank of filters to produce feature vectors. Queries for the delivery of these features will be posed to the sensor service from across the network. To properly satisfy its task, an agent is interested in such features in a timely fashion in order to build a good representation of what is happening in the world. In a real-time scenario, the driver and filters may be implemented to produce features at characteristic rates. In a system that satisfies real-time guarantees, both the network and remote procedure call (RPC) mechanism used to deliver these features must have a timing guarantee that satisfies an upper bound. Our current implementation does not make strict real-time guarantees.

A client request consists of a feature type, the duration of interest , the period defining feature delivery rate, and a target where the client can be contacted. A mechanism is needed to manage feature propagation at client specified rates. Together, the lease engine and the extrapolator serve

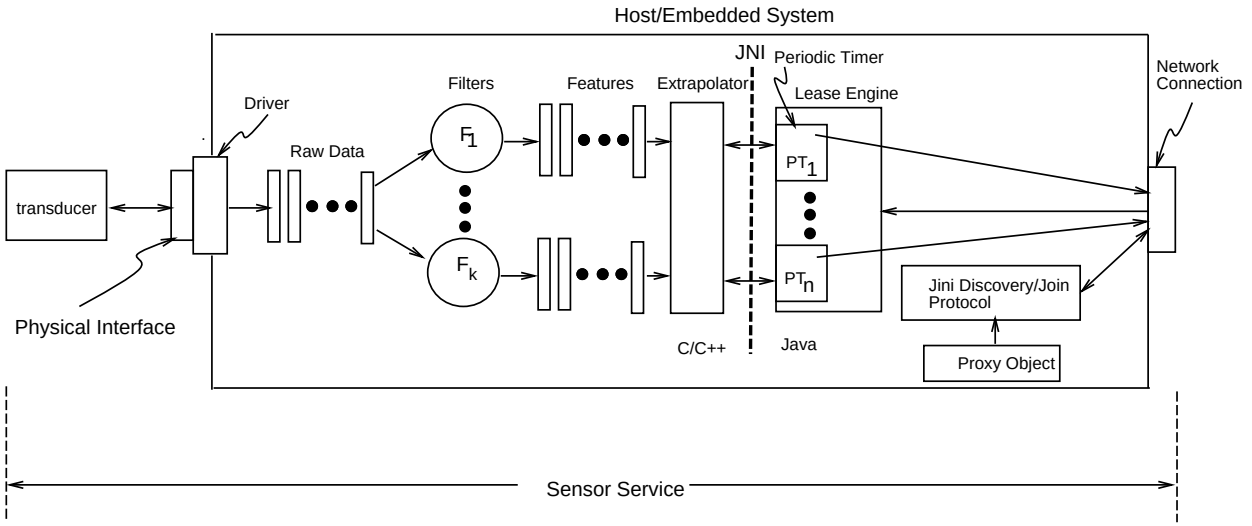


Figure 1: Sensor service using Jini Network Technology

this purpose. Caching context from feature streams, the extrapolator computes features for generation times between a real feature and its successor in the stream. This process can use statistical models or other function approximation methods. Error in extrapolation is governed by the length of the interval over which a computed feature must be produced. Our current approach uses first order spatio-temporal feature statistics for extrapolation.

Filter algorithms, drivers, and extrapolation techniques optimized in C and C++ are integrated into networked services using the Java Native Method (JNI) interface. Java objects corresponding to the computed features are constructed and handed to the Java process responsible for delivery to the client.

A sensor service exports objects that satisfy the following interface:

```
public interface Sensor extends Device {
   DeviceRegistration reportFeature(Type type,
        long duration,
        long period,
        RemoteEventListener target)
        throws LeaseDeniedException, RemoteException
}
```

Where a `Device` is an interface which serves as the root of the type hierarchy for sensors and motor services and a `DeviceRegstration` is:

```
public class DeviceRegistration
             extends EventRegistration {
}
```

If a request is deemed unachievable by a service, the request for feature delivery is not granted. This is signalled by throwing a `LeaseDeniedException`. If the request is achievable, using its lease engine, service grants the leasing of a periodic timer satisfying the given duration and period. The timer also includes code responsible for retrieving features and delivering them to a client through a target `RemoteEventListener` object.

Once instantiated, the periodic timer executes its action at regular intervals for the specified duration. The action requests a feature estimate for the current time from the extrapolator. If a feature sample is available, a feature object is constructed and delivered. If one is not available, the extrapolator computes an approximate feature and an object for the resultant feature estimate is constructed and delivered. A client is returned a `DeviceRegistration` object if its request is achievable. With this registration token, the client can extend or renew the requested duration. It is at the discretion of the service to grant such lease renewals. This approach uses Jini Distributed Events and Leases [2].

2.2 Motor Service

Figure 2 depicts the processes which a service attaches to a physical motor.

A motor service is composed of a set of concurrent controllers acting together to issue commands to a motor. These compositions act through an arbitration mechanism ensuring that a subordinate controller does not interfere with a dominant controller where their configuration spaces overlap[11]. Unaware of the service's constituent controllers, the client perceives the motor service as being governed by a single controller on which it may set goal references. Upon reaching a client initiated goal, the motor service generates an event signalling convergence.

The client packages its goal reference inside of a `ControlRef` object which it constructs and includes in its request along with a deadline, period and `RemoteEventListener` target object. The deadline represents the duration during for which the client is interested in convergence events. The

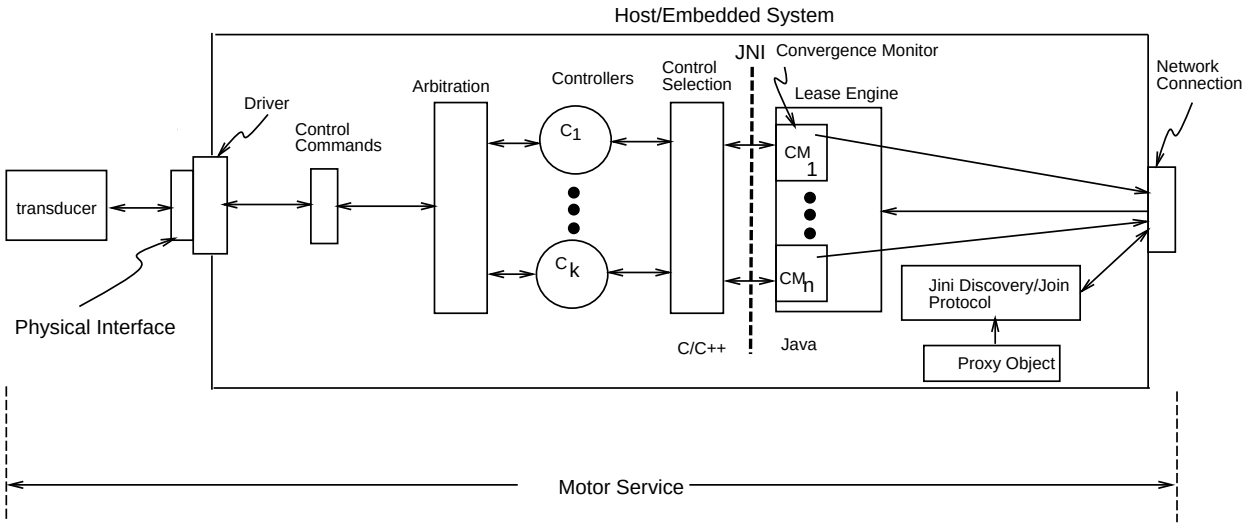


Figure 2: Motor Service using Jini Network Technology

period represents how aggressively convergence should be monitored.

The `RemoteEventListener` target serves as a proxy object used to contact the client with convergence events. A motor service also maintains a lease engine. Upon receipt of an achievable request, a lease is granted for a control monitor containing the target listener, period, `ControlRef` object, and deadline. The monitor sets the reference point on the control selection mechanism which, in turn, sets appropriate references on the proprietary controllers. At the specified rate, the monitor periodically checks for convergence status and sends an event to the client signalling the result.

A motor service may deem a request unachievable if it violates the boundary conditions in which the service operates. A client who poses an achievable request is returned a `DeviceRegistration` object. This registration token may be used to extend the duration. A motor service is not obligated to grant renewal requests. This approach uses Jini Distributed Events and Leases[2]. A motor service exports objects satisfying the following interface:

```
public interface Actuator extends Device {
   DeviceRegistration actuate(ControlRef ref,
        long duration,
        long period,
        RemoteEventListener target)
        throws LeaseDeniedException, RemoteException
}
```

A single sensorimotor service may export objects which implement both the `Sensor` and `Actuator` interfaces.

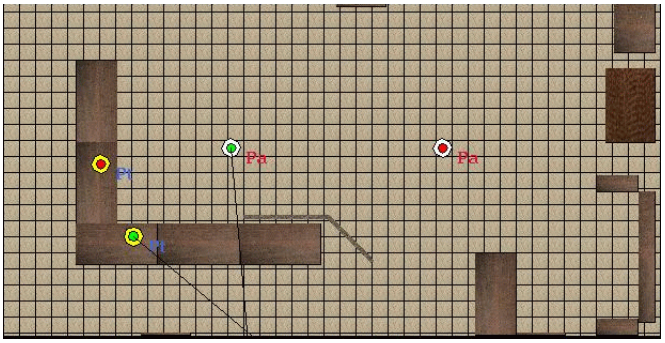
3. IMPLEMENTATION AND RUN-TIME PERFORMANCE

In the experiment described in this document, sensor services have been implemented for pyro-electric (thermal) sensors, panoramic cameras and pan/tilt/zoom (PTZ) cameras. Motor services have been implemented to move PTZ cameras. Our sensor and motor services are written using the Jini Network Technology and are dynamically discovered as they attach to the network. These services also associate attributes used to inform the agent where in the room the devices are located. We have also constructed higher level sensors which return headings from delivered features. We have constructed an agent consisting of a triangulation algorithm, a set of heading sensors, motor services for PTZ cameras, a finite state description for sensor pairs and an objective function. The objective function decides the state transition by selecting the best heading sensors based on network reachability and the number of missed periods for feature delivery.

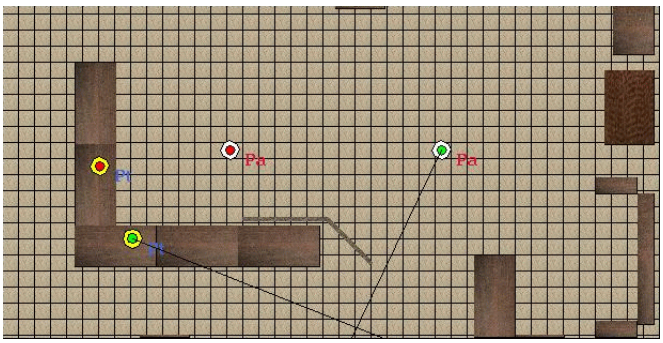
When the agent is run, it constructs heading services from dynamically discovered sensor services for panoramic and PTZ cameras. A triangulation algorithm is initialized with the heading service pair identified by an initial state. The agent then obtains the human's spatial location from the triangulation algorithm. As the human walked around the room, a number of faults were introduced. Faults included walking behind an occlusion, unplugging a camera, covering a camera, walking too swiftly the PTZ camera's motors to maintain tracking, and walking collinear with the two panoramic cameras.

If a sensor was unable to maintain the track, features from it would no longer be available causing it to miss feature delivery periods. The faulting sensor's rating was downgraded and the objective function induced a state transition by selecting a new best pair. A similar situation occurred when the human moved too quickly for the PTZ camera.

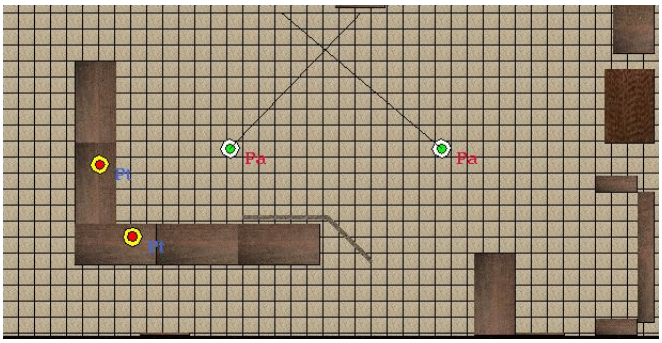
When a collinearity fault occurred when using the pair of panoramic camera heading sensors, a new best pair consisting of a PTZ and panoramic heading sensor pair was selected by the objective function. When the state transition was induced by the objective function, our algorithm forwarded the last known spatial location computed by triangulation



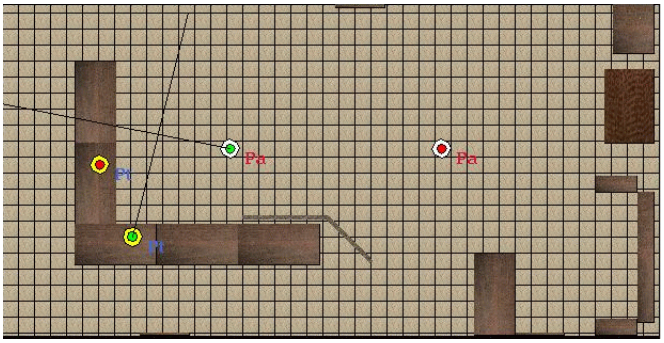
(a) tracking with left panoramic and lower PTZ



(b) switch to right panoramic and lower PTZ



(c) tracking with two panoramics



(d) collinear fault induced, switch to lower PTZ and left panoramic

Figure 3: Frames from a recorded tracking task demonstrating agent reconfiguration

as a goal reference for the PTZ camera's motor service. The PTZ camera saccaded to the last known human target location. The PTZ and panoramic heading sensor pair then provided heading information and tracking continued.

We have implemented a user interface which accepts human target locations from the agent and records and can re-play tracking data. We can run our agent consistently for a 24 hour period.

Figure 3 shows a replay of a recorded track. In the figure the two panoramic camera devices are labeled Pa and two PTZ cameras are labeled Pt. Scan lines are drawn in the user interface for the headings returned by the heading sensors for visualization purposes. As can be seen, the system tracks the moving target using the best pair of sensorimotor services. This figure demonstrates robust tracking of a human subject and must be read in order a,b,c,d. In frame a, heading services for the left panoramic and lower PTZ camera track the subject as he approaches an occluding office partition. As the target walks behind the partition, the left hand panoramic can no longer view the subject and begins missing deadlines for periodic feature delivery. The panoramic's rating by the objective function is downgraded, a state transition is induced, and the lower PTZ and right hand panoramic pair are selected in frame b. The subject emerges from behind the partition, but is no longer visible to the lower PTZ. A fault occurs and the objective function selects the panoramic pair which continue to track the subject in frame c. The target moves collinear with the the panoramic cameras. A fault occurs and the objective function selects the left panoramic and lower PTZ pair in frame d. We would like to stress that this is not a simulation. Our agent is a real implementation grounded in physical systems which we have used for successful tracking.

Using NTP and first order extrapolation techniques, we were able to produce features from cameras which bound uncertainty in headings to features to under a foot. Figure 4 depicts two subjects being tracked, namely the target on the left and the target on the right. The ground truth for the actual paths along which the two subjects walk is plotted along with the positions reported by the sensorimotor services. Note that each tile is a foot in length.

4. FUTURE WORK

In the future, we plan to implement sensor and motor services for audio, sound, and mobile robots. We also plan to increase the capabilities in the `DeviceRegistration` object which allows a client to switch features and to adjust the periods. Our higher level sensorimotor elements are implemented in the Java programming language. Since Java

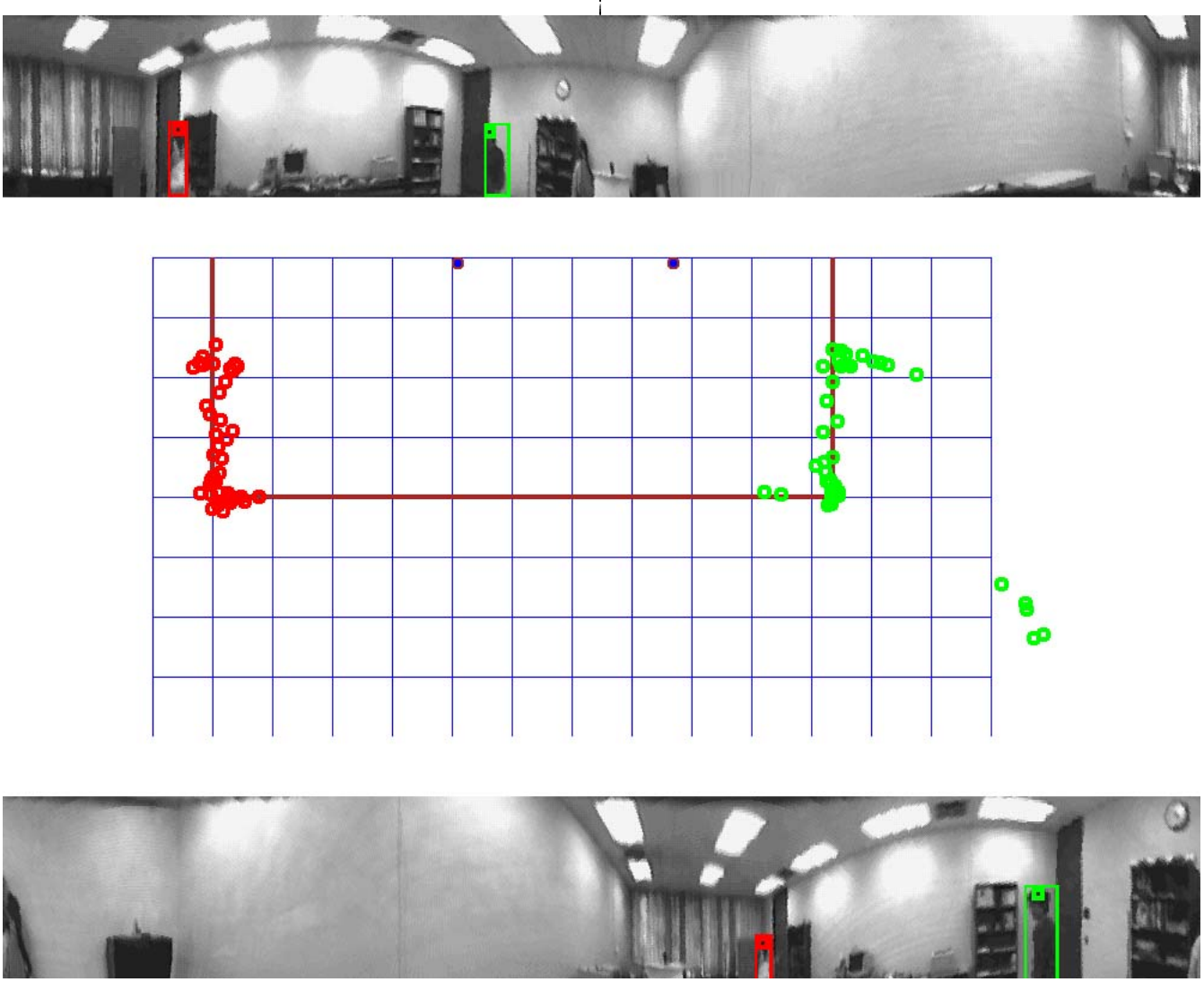


Figure 4: This figure demonstrates a tracking uncertainty bounded within 1 foot for camera services

objects may be serialized, we would also like to make virtual sensorimotor compositions persistent so that an automated process may index and recall previously created higher level services instead of constructing them on line each time. We plan to harden our service implementations with the goal of conducting experiments which run continuously for weeks at a time. Such a long running sensorimotor network will yield the kind of information necessary to begin building rich representations of activity as we look toward implementing activity maps and algorithms for learning activity in the room.

5. ACKNOWLEDGEMENTS

We would like to thank Dr. Zhigang Zhu, Prof. Edward M. Riseman, Prof. Allen R. Hanson for their involvement in the development of this research. We would also like to thank Yuichi Kikuchi for many contributions to this work.

6. REFERENCES

[1] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present and future reseach in ubiqitous computing. *ACM Transactions on Computer-Human Interaction, Speial Issue on HCI in the new Millenium*, 7(1):29–58, March 2000.

[2] K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath. *"The Jini Specification"*. Addison Wesley, 1999.

[3] D. Chakraborty C. Pullela, L. Xu and A. Joshi. A component based architecture for mobile information access. Tech. Report TR-CS-00-05, University of Maryland, Baltimore County, March 2000.

[4] H. Chen. Developing a dynamic distributed intelligent agent framework based on the jini architecture. Masters thesis, University of Maryland, Baltimore County, January 1999.

[5] Michael H. Cohen. Building brains for rooms: designing distributed software agents. Tech. report,

MIT Artificial Intelligence Laboratory, 1997.

[6] Michael H. Cohen. Design principles for intelligent environments. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Madison, Wisconsin, July 1998.

[7] Y. Ge. Jini smart sensor application in mobile interactive data acquisition systems(midas). Masters thesis, Oregon State University, March 2000.

[8] T. Henderson and R. Grupen. Logical behaviours. In *Journal Of Robotic Systems*, volume 7, pages 309–336, Santa Barbara, California, June 1990.

[9] T. Henderson and E. Shilcrat. Logical sensor systems. *Journal of Robotic Systems*, 1(2):169–193, March 1984.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceesings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, Massachusetts, November 2000.

[11] M. Huber and R. A. Grupen. Learning robot control-using control policies as abstract actions. *NIPS'98 Workshop : Abstraction and Hierarchy in Reinforcement Learning*, 1998.

[12] D. Karuppiah, P. Deegan, G. Holness, Z. Zhu, B. Lerner, R. Grupen, and E. Riseman. Software mode changes for continuous motion tracking. In *International Workshop on Self Adaptive Software*, Oxford, England, April 2000.

[13] D. Karuppiah, Z. Zhu, P. Shenoy, and E. Riseman. A fault-tolerant distributed vision system architecture for object tracking in a smart room. In *International Workshop on Computer Vision Systems*, Vancouver, Canada, July 2001.

[14] S. Kendall, J. Waldo, A. Wollrath, and G. Wyant. A note on distributed computing. Tech. Report TR 94-29, Sun Microsystems Laboratory, November 1994.

[15] D.L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Transactions on Networking*, 3, 1995.

[16] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, July 1999.