Pancakes: A Software Framework for Distributed Robot and Sensor Network Applications

Patrick Martin, Jean-Pierre de la Croix, and Magnus Egerstedt

Abstract The development of control applications for multi-agent robot and sensor networks is complicated by the heterogeneous nature of the systems involved, as well as their physical capabilities (or limitations). We propose a software framework that unifies these networked systems, thus facilitating the development of multi-agent control across multiple platforms and application domains. This framework addresses the need for these systems to dynamically adjust their actuating, sensing, and networking capabilities based on physical constraints, such as power levels. Furthermore, it allows for sensing and control algorithms to migrate to different platforms, which gives multi-agent control application designers the ability to adjust sensing and control as the network evolves. This paper describes the design and implementation of our software system and demonstrates its successful application on robots and sensor nodes, which dynamically modify their operational components.

1 Introduction

The increasing use of wireless sensor networks in distributed control applications, such as unmanned surveillance or building automation, results in the deployment of heterogenous, mobile computing platforms into new environments. These systems are usually connected with wired or wireless interfaces, such as Ethernet, Wi-Fi, or ZigBee, to enable the sharing of local information among the devices comprising the network. One important development for utilizing these distributed control networks is the incorporation of mobile robots, as noted by LaMarca *et al.* [13]. Allowing robots to interact with sensor networks provides new functionality in military, industrial, and consumer applications. In [13], the authors deployed a robot to

Patrick Martin · Jean-Pierre de la Croix · Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30312 USA
e-mail: {patrick.martin, jdelacroix}@gatech.edu, e-mail: magnus@ece.gatech.edu

maintain an office garden and its wireless sensors. The authors developed their own software framework that couples their robot with the sensor nodes embedded into the office garden. The robot successfully maintained energy resources of the sensors as well as detected failures.

To make multi-agent robotics applications, such as the prior example, work across different types of robots and wireless sensors, developers need software frameworks that help manage the complexity introduced by the heterogeneity of computational platforms and communication interfaces. Furthermore, the robotic and sensing devices on the network need to respond dynamically to physical changes (i.e. battery power). Providing the ability to dynamically adjust the framework at runtime opens up the possibility of extending operational lifetime, as well as adapting the system to reflect changes in the environment.

In this paper, we propose and demonstrate a software framework that unifies robotic and sensor networks in a seamless way. This framework, called *Pancakes*, gives developers several key features that facilitate the design of multi-agent control applications. First, Pancakes *abstracts sensing, actuation, and networking capabilities* such that high-level controllers can be implemented without worrying about low-level hardware management. Furthermore, this framework provides a structured way to *dynamically adjust* the runtime behavior of the sensor and robotic platforms according changes on the local system, as well as the operational environment. Complementary to this dynamic adjustment feature, Pancakes allows for the *migration of executable components* (i.e. sensing and control algorithms) from one platform to another. This software framework was inspired by the current literature in distributed and software control middleware, e.g. [6, 9, 16], robotics control software, e.g. [8, 10, 12, 15, 5], and actor-oriented design principles, e.g. [7, 11, 14].

In [6], Abdelzaher *et al.* developed a software framework that enabled the dynamic adjustment of a web server using feedback control. Their middleware exposed software "knobs" that could be adjusted to get better quality of service. Moving beyond this idea of modifying parameters of software components is the idea of reflective middleware [16]. This work describes a system where the pieces of the middleware dynamically adapt their capabilities as changes occur within the software. The work in [9] proposed a larger distributed embedded system framework that enables the development of software across many different types of computing platforms, from embedded controllers to desktop systems. In a similar manner, Pancakes gives robot and sensor network application designers the ability to dynamically change how their system operates at runtime. Furthermore, it allows for the migration of system components across deployed platforms.

The work in robotics software architectures made the control of heterogeneous systems easier by abstracting the sensors and actuators. For example, [10] created a common interface to the sensors and actuators of the robots so that users could write control software that works on different types of robots without having to know every detail of the robot's implementation. The authors of [8] took this idea a step further by separating the capabilities of a robot into discrete, re-usable components that can be assembled into a larger robot control application. Additionally, the work in [12] applied multi-agent software design to create a platform for developing

distributed robotics applications. The newer software package, ROS [5], provides an operating system-like framework in an attempt to standardize robotics software development for many robotics platforms. Pancakes provides the same sensor, actuator, and network abstraction that are commonplace in recent robotics software frameworks, which allow it to unify distributed robots with sensor networks.

To the best of the authors' knowledge, the combination of dynamic adjustment and migration of system components with hardware abstraction is a novel contribution to the distributed robotics community. These features allow us to create dynamic applications that leverage the powerful capabilities of these heterogeneous robot and sensor networks. In Section 2 we provide a high-level description of how Pancakes works using an example application. Following this overview, we discuss the architecture design and implementation in Section 3. In Section 4 we deploy the Pancakes architecture onto mobile robots that must encircle a region monitored by a sensor node. We conclude with some final remarks in Section 5.

2 Pancakes Overview

Each system deployed with Pancakes is treated at its highest level as a software agent. However, Pancakes is not a general purpose agent-based software framework, such as CybelePro [1] or JADE [3]. Instead, Pancakes focuses on providing an infrastructure for the distributed control of robots and sensor networks. The result is a Java-based system that can be deployed on embedded computers, such as ARM-based platforms, as well as full desktop environments.

Pancakes provides the necessary hardware and network abstractions that have become a common practice in current robotics software frameworks. These abstractions let users utilize the system devices that interact with the environment, such as actuators and sensors. Also, the networking services let each agent share local information by passing messages over the network interface without having to micromanage the low-level communication protocols.

Internally, each Pancakes agent is composed of the Pancakes *kernel* and a collection of actor-like software components that communicate with each other using input and output channels provided by the Pancakes kernel. The two types of components in Pancakes are *tasks* and *services*. Tasks carry out a particular function for the Pancakes agent, such as reading sensor data or performing agent discovery. They publish their results onto their output channels for other tasks or services to use.

Services spawn and manage tasks that the agent requires for execution. Services submit periodic tasks to the kernel's *scheduler* for execution. Additionally, event-driven tasks are configured to listen to their input channels for new messages, use these messages to carry out their computation, and eventually send results to an output channel. The services also enable the dynamic reconfiguration of the middleware by starting new tasks, adjusting task schedules, stopping current tasks, migrating tasks across platforms, or shutting down an entire service. Dynamic reconfigura-

tion is especially important when we construct power- and communication-aware applications.

Consider the mobile robots and wireless motion sensor shown in Figure 1, which are deployed in a building for security monitoring. The two mobile robots, Agents 1 and 2, need to communicate between themselves and the sensor node in order to share information necessary for completing the desired monitoring mission. Some important questions an application designer needs to consider are: what happens when a mobile robot is low on power? and how can robot tasks be transferred from one agent to another?

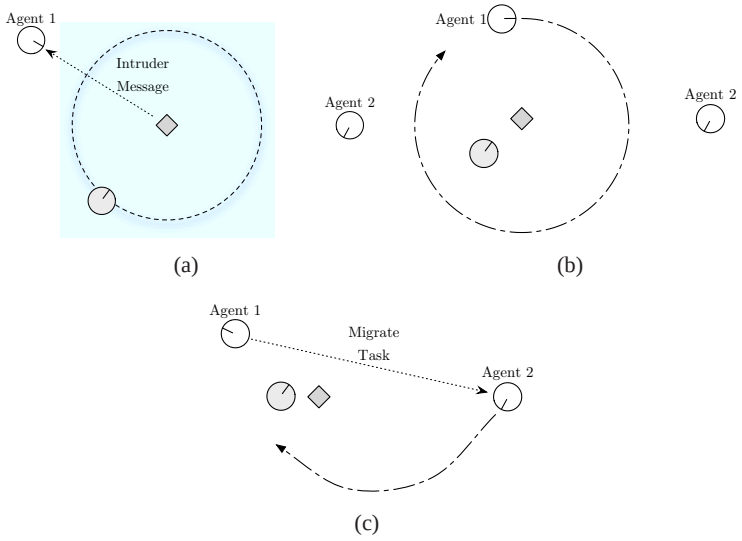

Fig. 1 An illustrative example of our desired control application for mobile robots (white circles) working with a sensor node (grey diamond) to isolate an intruding agent (grey circle).

Using Pancakes, we can create a control application that allows us to address these questions in the following way. When the sensor node (grey diamond) detects motion from an intruder (Figure 1(a)), it sends a message to Agent 1. This robot initiates a task that encircles the region where the intruder was detected, as illustrated in Figure 1(b). While executing this task, Agent 1 can use Pancakes to monitor its power consumption and actively adjust its speed or communication rates to conserve power. If Agent 1 consumes too much energy, it needs to ensure that the region is still monitored by *migrating* its currently running task to Agent 2 (Figure 1(c)). This capability would effectively lengthen the operational time of the network by letting Agent 2 wait until absolutely necessary before executing a task. The details on how Pancakes is designed to facilitate the implementation of this application is the subject of the next section.

3 The Pancakes Architecture

In this section, we describe the architecture of the Pancakes software framework and how they work together to facilitate the development of multi-agent robotics applications. Each Pancakes agent is composed of a collection of executable components, or *tasks*, which are the "workhorses" of Pancakes. These tasks are associated with a *service* that maintains a collection of related tasks. Since we adopt the actor-oriented model of programming [7, 11, 14], tasks and services communicate with each other through a collection of channels, the *information stream*. Combining these pieces with a scheduler allows for the construction of parallel and dynamic control applications for multi-agent systems.

3.1 Information Stream

The information stream sets up the communication channels that services and tasks use to publish new information or subscribe to receive information from other Pancakes components. This stream contains five core channels: `system`, `sysctrl`, `ctrl`, `network`, and `log`. Additionally, services can create specialized channels at runtime that are used to pass service specific information among tasks within the service.

The `system` channel provides a channel for services and tasks to publish system information to user-made and other system tasks. For example, a mobile robot's sonar sensor task would publish its most recent data points to the `system` channel, which is subscribed to by a control task. The `sysctrl` channel serves as a control messaging channel among the services and tasks. Messages sent over this channel facilitate the dynamic rescheduling, shut-down, or migration of tasks. To issue control commands to actuators, tasks send messages over the `ctrl` channel. A task or service that requires network communication publishes its network messages to the `network` channel. Finally, the `log` channel allows any Pancakes component to perform error, debug, or data logging, which helps in the post-run analysis and debugging of complex distributed applications.

3.2 Tasks

Task components are the main "actors" in Pancakes: they produce and consume information in order to affect a change in the deployed system. Tasks can execute in time-driven, event-driven, or a combination of both modes depending on the desired functionality set by the designer. At startup, a time-driven task is submitted to the scheduler and is executed at its specified frequency. The event-driven tasks wait for a message to arrive on one of its incoming channels. Since tasks communicate via the Pancakes stream channels, there is no need to synchronize on shared variables.

Instead, the data necessary for execution is transmitted through the channels and delivered to subscribing Pancakes components.

Tasks are a natural way to abstract how different pieces in the system should interact. For instance, as shown in Figure 2, a robot can have several sensing tasks, such as sonar, IR, or local pose, and a control task that takes the output from these sensors and computes a control input for the actuation system. Furthermore, there can be a supervisor task that executes in parallel, monitors the output of all of the other tasks, and makes higher level decisions. In this example, the Supervisor Task examines the sensor input *and* the output from the control task in order to adjust sampling rates of different sensor managed by the Device Service. The advantage of this approach is that the application designer can focus on developing the input/output behavior of each individual task, rather than deal with complicated thread management.

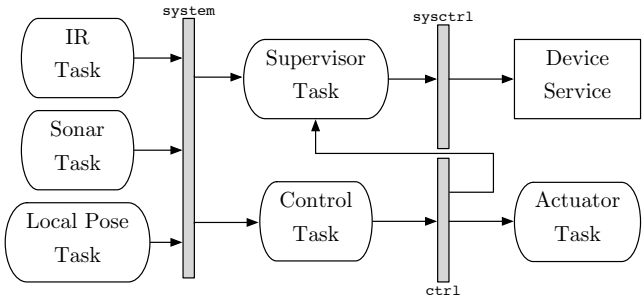

Fig. 2 In this example Pancakes application, a robot is deployed with sonar, IR, and local pose sensors. This sensor information is communicated to the system via the `system` channel, which is subscribed to by the Control Task and Supervisor Task. The Control Task computes a command for the actuators and sends it over the `ctrl` channel. Additionally, the Supervisor Task sends commands over `sysctrl` to the Device Service to adjust runtime components (e.g. the sampling rate of the IR task).

3.3 Services

The duties of services are 1) to maintain a registry of its tasks that are currently running, 2) to manage the startup, shutdown, or migration of tasks, and 3) to manage the shutdown or restart of the service itself. Pancakes has four default services that are loaded at startup: the Device Service, the Network Service, the Log Service, and the Client Service. Furthermore, developers can create new services that can supply additional functionality for their system.

The Device Service creates the system device tasks, which are the hardware abstractions for sensors and actuators, and schedules any that require timed execution.

The Network Service enables communication with other Pancakes agents on the network. This service creates a network client task that listens to the `network` channel for any outgoing network messages and transmits them to the intended target. The Log Service listens to the `log` channel and displays error and debug messages to the console; additionally, it can record messages to a file for later analysis. Finally, the Client Service spawns user tasks and channels for inter-task communication. Users implement tasks to carry out communication and control algorithms, which are then loaded into the Client Service at system startup.

3.4 Dynamic Adjustment and Migration of Components

One of the key features of Pancakes is the ability to adjust services and tasks as an application executes on a deployed system. This feature lets the application adjust the capabilities within the architecture according to dynamic effects from software (i.e. logic statements, software controllers) or the physical environment (i.e. power consumption, sensing data). For example, a task can request that network discovery be slowed down to reduce the number of network transmissions; therefore, it can reduce the rate of power consumption of the application. Also, a more drastic power savings could be achieved by requesting the Network Service to shut down temporarily.

We enable this feature by establishing a messaging protocol for tasks and services to request a change the runtime behavior of other tasks and services. The currently supported control operations are *stop*, *restart*, *start*, or *reschedule*. For a task or service to initiate one of these controls, it must send a control message over the `sysctrl` channel within Pancakes. All services subscribe to this channel and inspect the message to determine if it has to change its behavior or that of one of its tasks. Once the message is received at the target service, the service calls on the scheduler to stop, start, or reschedule the task. Additionally, if the service is requested to stop or restart, it shuts down all of its currently running tasks and requests the scheduler to stop or restart itself.

A complementary feature to dynamic adjustment is the ability to migrate tasks among deployed Pancakes systems. As illustrated in the example of Section 2, mobile robots and sensor networks can use this capability to achieve a longer mission lifetime. Task migration is managed by the Task Migrator task in the Network Service, since it must communicate with neighboring agents to find a suitable candidate for migration. The migration protocol involves sending the task and its dependencies for execution (i.e. required sensors and/or actuators) to all neighbors of the current agent. Once candidate agents are found, the migrating agent chooses the one that has the lowest execution “cost.” In its current implementation, our cost metric is based on the system load of the candidate agent, for example, the number of tasks running on the deployed system.

3.5 Implementation

To enable the concurrent operation of multiple hardware and networking devices, Pancakes makes use of the actor-oriented programming model as described in [7, 11, 14]. This model creates software components that focus on concurrency and communication rather than interface methods, such as remote method invocation: a common technique in object oriented software [14]. By using an actor-oriented approach, Pancakes avoids the common issues of thread blocking in concurrent applications, since information is shared via message passing among the components, rather than through direct function calls.

We implemented Pancakes in Java to ensure its operation on several types of computational platforms and operating systems. In particular, the robots and sensors nodes in our lab use low-power ARM processors and an embedded Linux/GNU OS. We use the open source virtual machine JamVM [2], which can be compiled for several processor architectures. Another reason we use Java as our implementation language is the existence of robust Java libraries that enable concurrency and message passing. The Java SE 6 standard library has new concurrency tools that efficiently handle multiple threads using specialized thread pools. Complementary to this library, we make use of the Jetlang [4] library, which provides messaging services for multi-threaded applications.

4 Experimental Results

In this section, we use Pancakes to implement the example application described in Section 2. The platforms used in our experiment are shown in Figure 3. Pancakes is deployed on two Khepera III robots, which perform the target encirclement behavior with a BUG sensor node as described in the example scenario of Section 2. To determine the local pose of each system we use a Vicon motion capture system. Since this local pose data is produced off-board by the motion capture system, it is a "virtual" local sensor on each robot. The Vicon system tracks the reflective points on each robot and transmits the local pose data to each robot, where the data is received and handled by a Local Pose task in Pancakes.

The following experiment uses two robots to perform surveillance, and a BUG sensor node for motion monitoring. The robots have two tasks available: 1) a `ScanTarget` task, which implements the boundary tracking algorithm of [17] and 2) a `GoHome` task, which drives the robot back to its home station. The BUG sensor has a task, `MotionDetection`, that monitors its motion detection sensor and transmits its location to its neighbors when motion is detected. Initially, the BUG is set near the center of the monitored area and both robots are initialized. Agent 1 starts with its `ScanTarget` task initialized and agent 2 is held idle for reserve.

Figure 4 shows how the systems execute during the initial phase of the experiment. Agent 1, the $\triangle$ symbol, starts from its initial position in the top right of the area. It converges to a circle around the BUG sensor, denoted by the $\square$ near the ori-

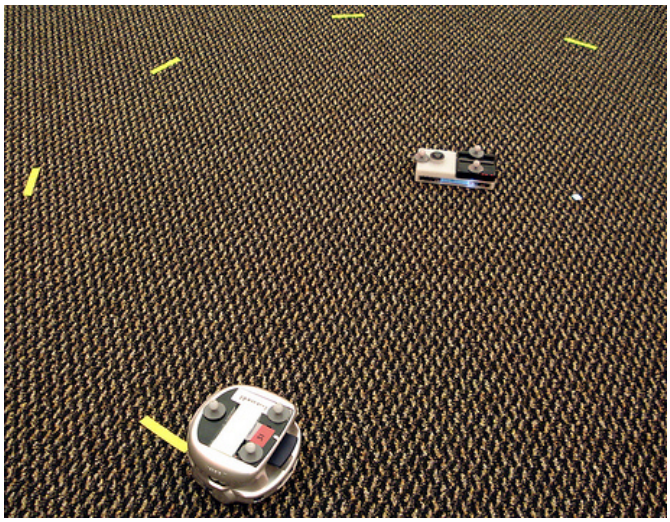

Fig. 3 This figure shows the hardware devices used in our experiment. The figure shows one of the two K-Team robots circling a BUGLabs BUGBase, which is our sensor node to detect intruders. These systems were provided with indoor localization data from the motion capture systems shown.

gin. Agent 2, denoted by the ○, waits in the bottom right of the region to be assigned a task. Once agent 1's battery level drops below a particular threshold, it begins its *task migration*, such that agent 2 can take over the `ScanTarget` task.

Agent 1 sends a migration message to all of its neighbors, which includes the task itself and a list of dependencies the task needs to execute. For now, these dependencies are the types of devices the platform supports (i.e. sonar, local pose, motors). When a neighbor receives the message, it checks for dependencies and, if compatible, returns a positive reply with a "cost" value. This cost is currently calculated by counting the number of active tasks running on the platform; however, the framework is flexible enough that a more complicated cost could be calculated from power levels, communication rates, or other important properties of the platform. Agent 1 inspects all of its valid replies, chooses the agent with the lowest cost, and migrates the task to that agent.

Figure 5 shows the trajectories of the systems after migration has taken place. Agent 1 has started its `GoHome` task and agent 2 is encircling the BUG sensor node using the migrated `ScanTarget` task. Agent 2 continues to circle the sensor node, as shown in Figure 6, and agent 1 has returned to its home position for the duration of the mission. This experiment shows how the Pancakes framework enabled the creation of a dynamic control application for a small team comprised of mobile robots and a sensor node.

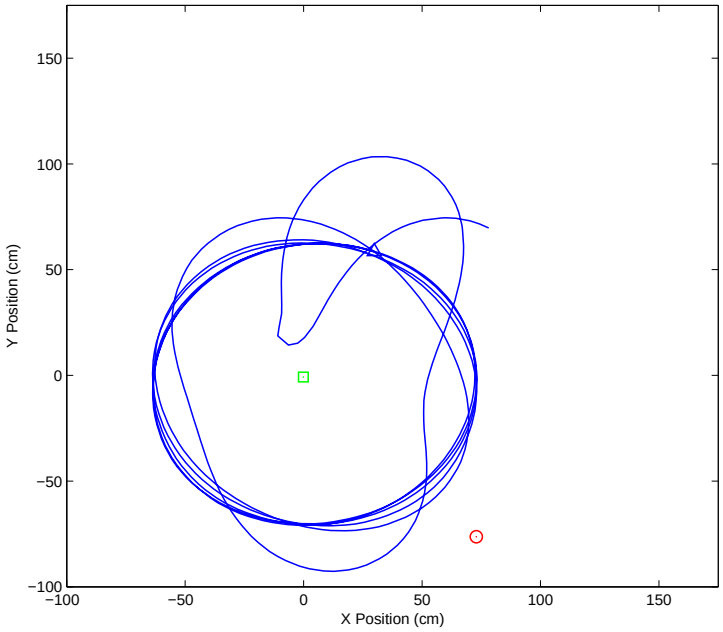

Fig. 4 This figure shows the agent 1, denoted by △ with trajectory, encircling the BUG sensor (□ near the origin). Agent 2, denoted by the ○, is idle to the bottom right.

5 Conclusion

In this paper we designed and demonstrated a new software infrastructure for developing control applications for mobile robots and sensor networks. The benefits of this system are its ability to abstract the sensors, actuators, and network devices as well as the ability to dynamically change how these components operate. Furthermore, the systems allows for the migration of tasks to other agents that have the necessary capabilities to execute them. Our experimental results show that this framework facilitates the development of dynamic control and sensing applications that can incorporate heterogeneous distributed systems.

Acknowledgement

This work was sponsored by the US Office for Naval Research through the grant MURI HUNT.

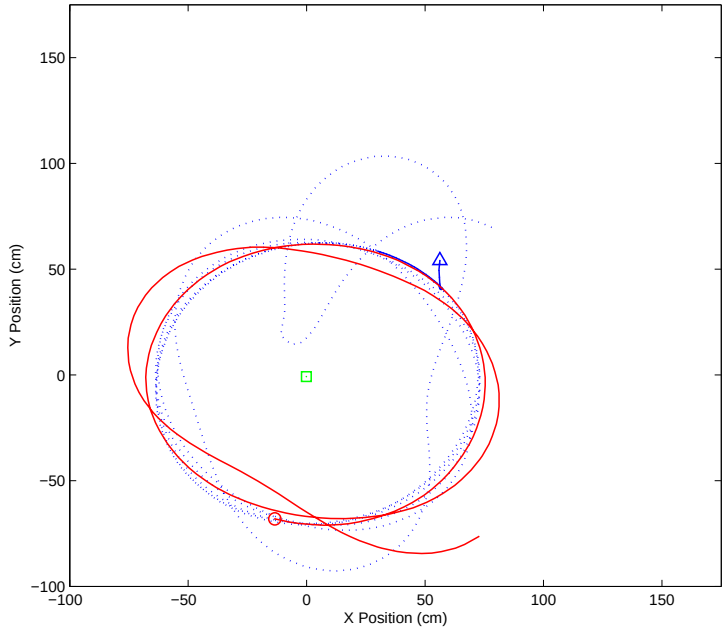

Fig. 5 Once the first agent (△) runs low on its battery level, it sends out a migration message to the other two agents. Since agent 2 (○) has the necessary devices needed to execute, it accepts the task and starts the same encirclement algorithm. At the same time, agent 1 begins its `GoHome` behavior.

References

1. Cybelepro (2010). URL http://products.i-a-i.com
2. Jamvm (2010). URL http://jamvm.sourceforge.net
3. Java agent development framework (2010). URL http://jade.tilab.com
4. Jetlang (2010). URL http://code.google.com/p/jetlang
5. Ros (2010). URL http://www.ros.org
6. Abdelzaher, T., Stankovic, J., Lu, C., Zhang, R., Lu, Y.: Feedback performance control in software services. IEEE Control Systems Magazine pp. 74–90 (2003)
7. Agha, G.: Concurrent object-oriented programming. Communications of the ACM **33**(9), 125–140 (1990)
8. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A.: Towards component-based robotics. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS), pp. 163–168 (2005)
9. Cornea, R., Dutt, N., Gupta, R., Kreuger, I., Nicolau, A., Schmidt, D., Shukla, S.: Forge: A framework for optimization of distributed embedded systems software. In: Proceedings of the 17th IEEE/ACM International Parallel and Distributed Processing Symposium (2003)
10. Gerkey, B., Vaughan, R., Howard, A.: The player/stage project: tools for multi-robot and distributed sensor systems. In: Proceedings of the International Conference on Advanced Robotics (ICRA), pp. 317–323 (2003)

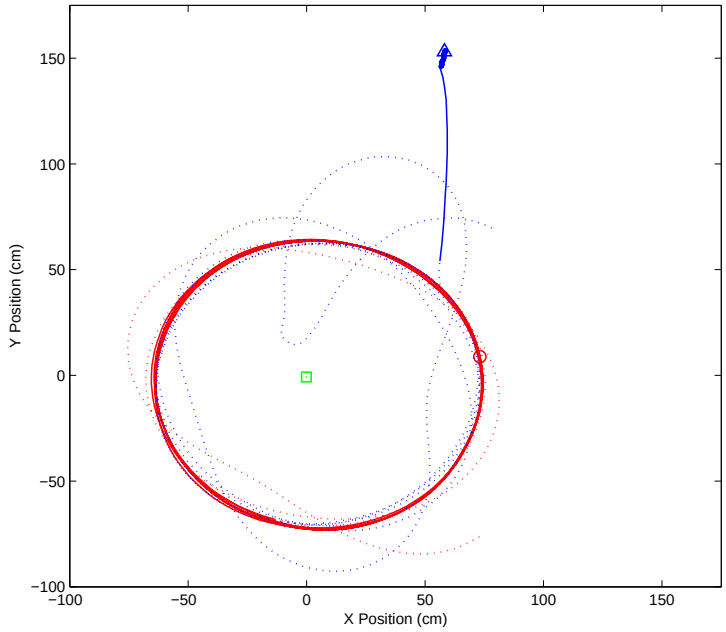

Fig. 6 At this point, agent 1 has finished its `GoHome` behavior. Agent 2 continues to encircle the BUG sensor.

11. Hewitt, C.: Viewing control structures as patterns of passing messages. Journal of Artificial Intelligence **8**(3), 323–363 (1977)
12. Kulis, Z., Manikonda, V., Azimi-Sadjadi, B., Ranjan, P.: The distributed control framework: A software infrastructure for agent-based distributed control and robotics. In: Proceedings of American Control Conference (2008)
13. LaMarca, A., Brunette, W., Koizumi, D., Lease, M., Sigurdsson, S.B., Sikorski, K., Fox, D., Borriello1, G.: Making sensor networks practical with robots. In: International Conference on Pervasive Computing, pp. 615–622 (2002)
14. Lee, E.A.: Model-driven development - from object-oriented design to actor-oriented design. In: Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (a.k.a. The Monterey Workshop) (2003)
15. Montemerlo, M., Roy, N., Thrun, S.: Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS), pp. 2436–2441 (2003)
16. Wang, N., Kircher, M., Schmidt, D.: Applying reflective middleware techniques to optimize a qos-enabled corba component model implementation. In: Proceedings of 24th Annual International Computer Software and Applications Conference, pp. 492–499 (2000)
17. Zhang, F., Justh, E., Krishnaprasad, P.: Boundary following using gyroscopic control. In: Proceedings of the 43rd IEEE Conference on Decision and Control (2004)