Speaking Swarmish: Human-Robot Interface Design for Large Swarms of Autonomous Mobile Robots

James McLurkin[1], Jennifer Smith[2], James Frankel[3], David Sotkowitz, David Blau[4], Brian Schmidt[5]

{jamesm[1], scrbl[4], bschmidt[5]}@csail.mit.edu
MIT Computer Science and Artificial Intelligence Lab
Cambridge, MA 02139

{jsmith[2],jfrankel[3]}@irobot.com
iRobot Corporation
63 South Ave, Burlington, MA 01803

Abstract

Human-robot interfaces for interacting with hundreds of autonomous robots must be very different from single-robot interfaces. The central design challenge is developing techniques to maintain, program, and interact with the robots without having to handle them individually. This requires robots that can support hands-free operation, which drives many other aspects of the design.

This paper presents the design philosophy and practical experience with human-robot interfaces to develop, debug, and evaluate distributed algorithms on the 112-robot iRobot Swarm. These human-robot interaction (HRI) techniques fall into three categories: a **physical infrastructure** to support hands-free operation, **utility software** for centralized development and debugging, and **carefully designed lights, sounds and movement** that allow the user to interpret the inner workings of groups of robots without having to look away or use special equipment. The end result is a useable Swarm, with develop-run-debug cycle times approaching that of a simulation.

1 Introduction

The task of interacting with hundreds of autonomous robots presents unique challenges for the user interface designer. Traditional graphical user interfaces, data logs, and even standard power switches fail to provide the user with a practical, efficient interface. The core issue is one of scale: in a system of *n* robots, any task that has to be done to one robot must be done to the remaining *n* - 1. Our solution is a swarm that can operate largely without physical interaction, using an infrastructure that allows remote power management and autonomous recharging, and has software for centralized user input and techniques for global swarm output.

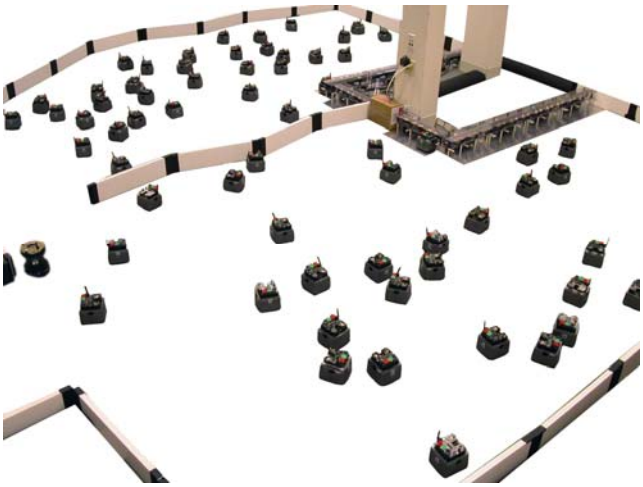

Figure 1: The iRobot Swarm is composed of 112 individual robots that work together to accomplish group goals. The robots autonomously dock with the charging stations in the top right of the figure. Long-range navigation is provided by the beacon at the middle left of the image. See Figure 2 for detailed information about the support hardware.

Section 2 describes the hardware required for hands-free operation. This includes the chargers, navigational beacons, and the power circuitry on the robots. Section 3 discusses the centralized command, control, and data collection software. Parts of this suite are inspired by video games in which the user commands a large army of individual units. Section 4 describes our approach of using lights sounds, and behaviors on individual robots as a primary output channel. This allows the user to ascertain the inner workings of a single robot, small groups of robots, or even the entire swarm, without having to look at a computer screen.

2 Hardware for Hands-Free Operation

The Swarm infrastructure components, shown in Figure 2, provide the physical resources the robots need to keep themselves running. These include chargers, navigational beacons, and a semi-automated test stand. The charging stations are the most important of these components, as they allow the robots to autonomously recharge their batteries.

The long-range navigation beacons are designed to help guide the robots to their chargers from anywhere in their workspace. In practice, we have found that it is easier to provide a multi-hop communications route, and hence a navigational path, to the chargers using the robot's local communications system (Intanagonwiwat 2000, McLurkin 2004). This eliminates the need to set up any additional hardware. The SwarmBot's bump skirts provide the robust low-level obstacle avoidance needed to allow both the navigation and docking behaviors to run successfully.

The SwarmBot's power management circuitry has four modes of operation: on, standby, off, and battery-disconnect. The standby mode allows the user to power-on the robots remotely from a "gateway robot" (described in the next section). Once on, the robots can be remotely

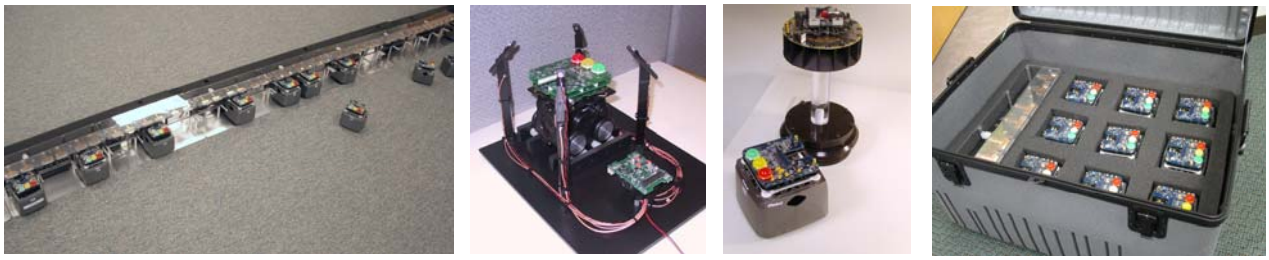

Figure 2: Working with a large swarm of robots requires them to be as self-sufficient as possible. **Left:** Chargers allow robots to dock and recharge autonomously. **Middle Left:** Semi-automated testing allows initial calibration and quick diagnosis of problems. **Middle Right:** Long-range ISIS beacons aid navigation. **Right:** Each travel case hold 18 robots and two chargers.

powered down via the same interface. This ability supports the sporadic nature of software development; the robots remain on during periods of active progress, but can be easily powered down to conserve batteries when a difficult bug slows the pace. This reduces wear on the batteries, and allows the user to maintain a particular physical arrangement of robots for testing far longer than if the robots were left on continuously. The battery-disconnect mode makes the robots safe for shipping and storage.

The Swarm, or subsets of it, often travel around the lab or around the country. The travel cases shown in Figure 2 make transporting the robots safe and easy. Each case contains 18 robots and 2 chargers, a ratio that supports continuous operation.

The most critical hands-free operation is remote programming. Our approach is a single-hop broadcast algorithm similar to those presented by Stathopoulos and Reijers (Stathopoulos, 2003, N. Reijers 2003), but simpler and easier to implement.

First, new software is downloaded to one robot via the serial port. This "gateway robot" computes a 32-bit CRC for each segment of memory, and then broadcasts these CRCs to the entire swarm via radio. All the other robots compare the CRC's of their memory segments to the ones they just received, and flag the appropriate segments for update. The gateway robot then queries all other robots in sequence, retransmitting code segments as necessary. Robots who are not being actively queried snoop on the channel to opportunistically receive segments they have flagged.

The resulting download is quite efficient. A 40-robot swarm can be reprogrammed with a 100-kilobyte application in under 30 seconds. This significantly lowers the barriers to testing new software, or polishing existing behaviors. For example, because the programming cycle is so painless, it is common to change only a single constant in the source code and re-download.

The downside of this simplified approach is the single-hop network requirement, as robots that are too far from the user will not be reprogrammed. However, in a typical laboratory environment, the radio's range is sufficient to reach all the robots, or the robots can be recalled to the home base for reprogramming.

3 Centralized User Interfaces

Centralized input allows the swarm to be controlled by a single user, using the gateway robot to provide connectivity between the user's computer and the Swarm. The VT100 terminal display shown in Figure 3 allows the user to send commands to an individual robot or to the entire Swarm. Simple graphical output is also possible, but limitations of the VT100 display make this interface best for input.

Commercial-off-the-shelf video game controllers are also an excellent hardware input device for some applications. In particular, the controllers designed for the Sony PlayStation are high quality, and simple to interface to the robots. This approach allows one or more users to directly control individual robots, and trough group behaviors, the entire swarm. These controllers are ideal for demonstrations and classroom lessons.

The graphical user interface shown in Figure 4 displays real-time telemetry data, detailed internal state, local neighbor positioning, and global robot positioning. Its design is inspired by the graphical user interfaces (GUI) of

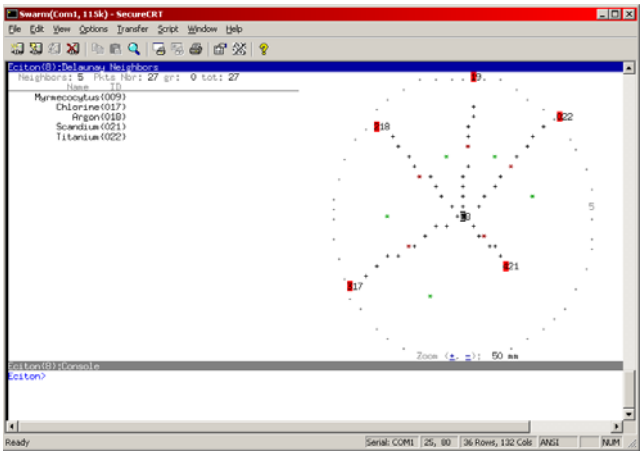

Figure 3: A VT100 console interface allows a single user to issue commands to an individual robot or the entire swarm. It can also display simple graphics output in real-time. The example above is displaying the position of the local Delaunay neighbors of the gateway robot (red squares at the end of the black "lines"), and the resulting Voronoi cell (green asterisks).

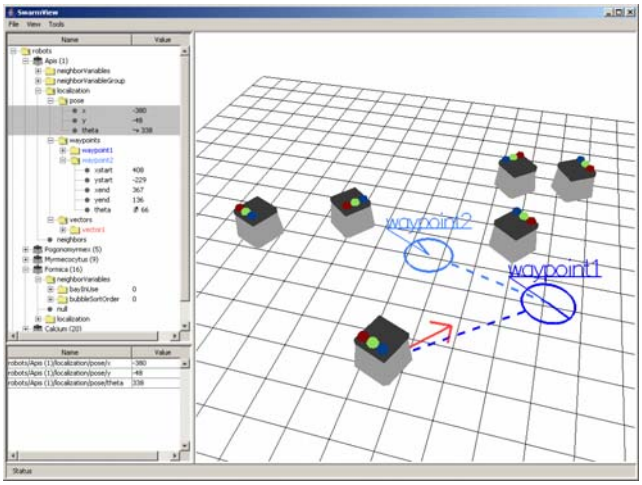

Figure 4: The SwarmCraft centralized GUI is inspired by real-time strategy video games and graphical software debuggers. The tree view on the right displays the available telemetry for all the robots in the swarm. The view on the left displays global positioning, annotated with debugging information. This view shows a test with motion waypoints and a velocity vector. Individual robot windows can be opened to display telemetry and neighbors of a single robot. A "quick graph" function allows the selected datum to be graphed in real-time. The GUI only draws what the robots tell it to, allowing it to remain unchanged as the Swarm's software develops.

real-time strategy video games such as StarCraft and WarCraft (Blizzard, 1998). Games like these challenge the user to direct an army of individual units to victory. Although it is common to have over 100 units on each team, elegant user interfaces make it simple for the user to control individual units, groups, or the entire army.

Attempts to implement this style of centralized graphical user interface (GUI) on a physical swarm have met with mixed success, as it is often easier to type commands into the console or to manually move the robots to the desired arrangement. However, these user interface metaphors are highly effective for centralized data collection and display. The gateway robot collects telemetry data and application-specific data from individual robots; this data is then passed to the GUI, and displayed in many different formats. The image in Figure 4 shows the swarm view with a single robot moving towards two waypoints. This GUI is currently under active development, and we plan to add many features and annotations to enhance the visualization of swarm data.

The problem of keeping the GUI and robot code in sync is a development challenge. Our solution is to design the GUI to function like a specialized terminal, receiving most of its commands from the robots. Then, as the robot's code changes, the GUI can support different software with little or no modifications.

4 Global User Output

When developing and debugging group behaviors, it is critical to monitor both the actions and the internal program state of the robots simultaneously. Displaying the internal state on monitors requires the user to continuously alternate between watching the robots and the screen. To solve this problem, we make the internal state readily observable by looking at the robot itself. Consider the SwarmBot shown in Figure 5. Each robot is a 12 cm square, so small text displays are useless when viewed from a distance. So we install large LEDs (lights) and an audio system on each robot, allowing them to communicate with users in a language we have affectionately dubbed "Swarmish".

4.1 Swarmish LEDs

Each robot has a red, green, and blue LED on top that can be programmed to blink in several patterns. This is our primary behavior-level debugging interface on the robots. Patterns that use one LED alone or all the LEDs together seem to work best, as patterns involving multiple lights communicating independent information can be difficult to read quickly.

We have settled on two intensities, bright and dim, and two wave patterns, a square wave and a semi-sinusoidal wave. The two patterns are only distinguishable at lower frequencies, which gives us 12 distinct single-LED patterns that can be read by an experienced user. The absolute minimum time to read square patterns is ½ the period of the second-lowest frequency, which is 533 milliseconds. The semi-sinusoidal patterns can be read slightly faster

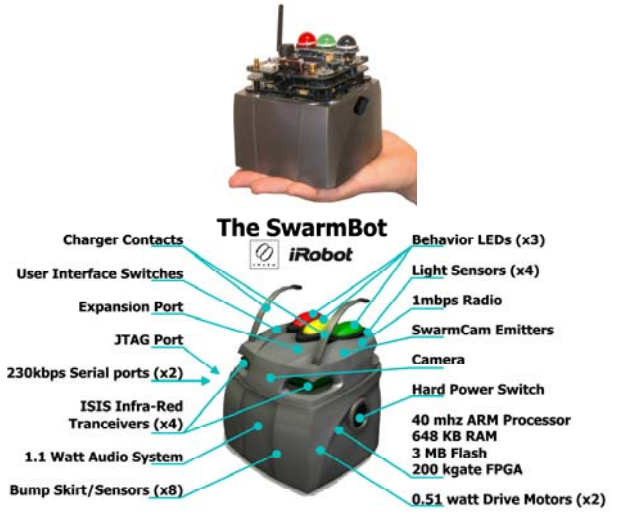

Figure 5: The iRobot SwarmBot is designed from the ground up for distributed algorithm development. The robots use an infra-red inter robot communications system (ISIS) for local communications and positioning. About one third of the robot's hardware (charger contacts, user interface switches, serial ports, audio system, behavior LEDs, and radio) is used exclusively for human-robot interaction and hands-free operation.

because the user can infer the frequency from the slope. Besides the square and semi-sinusoidal patterns from above, the four all-LED patterns also include two patterns that cycle back and forth through all the lights, either with smooth or sharp transitions.

All these variations produce 108 common patterns, each of which can be read in about ½ second. In an actual application, similar behaviors and states are grouped into single colors, leaving many patterns unused. Even with this limitation, the level of expressiveness is sufficient for debugging a new behavior or displaying the top-level state of an application.

For example, an LED assignment for an application that arranges the robots in order (a physical bubble sort) is given in Table 1 below:

Application Behavior	color, frequency, pattern, dimmer
lowest robot	blue low-freq sine bright
intermediate robot, unsorted, moving towards lower robots	blue hi-freq square bright
intermediate robot, sorted, in position	green low-freq sine bright
intermediate robot, sorted, not in position	green hi-freq square bright
intermediate robot, unsorted, moving towards higher robots	red hi-freq square bright
highest robot	red low-freq sine bright
gateway robot	blue hi-freq square dim

Table 1. Swarmish LED assignments for a bubble sort program.

In this example, “low-freq sine” translates to “complete”, “hi-freq square” means “in motion”, blue lights are related to the lowest robot, and red to the highest. Careful grouping of colors and patterns allow fairly complicated software to be readily understandable. We do not program lights to be all off or steady on, as either of these patterns is impossible to distinguish from a software crash.

A current limitation of this approach is that the light patterns are programmed into the software – to change the information displayed requires a swarm download. Future work will allow the user to select the information to be displayed from the command line or GUI.

4.2 Swarmish Audio

The LEDs offer detailed information about an individual robot. In contrast, the audio system can give the user an overview of the activities of the entire swarm, and can be monitored while looking elsewhere. This is a somewhat nostalgic approach: many veteran software engineers reminisce fondly of a time when they could debug programs by listening to the internal operations of their computers on nearby radios. Once the user learned what normal execution sounded like, deviations were quickly noticed, focusing attention on the offending part of the program. This resurrected approach to debugging has proven to be very effective on the Swarm, and our modern interpretation is discussed in this section.

Each robot has a 1.1 watt audio system than can produce a subset of the general MIDI instruments. This allows the Swarm to play any MIDI file, but we have found that single notes work best for debugging. There are four parameters to vary per note: instrument, pitch, duration, and volume. Care must be taken to blend these selections into a group composition that is intelligible. Good note selections for correctly operating programs produce chords, tempos, and rhythms. Once a user has become attuned to variations in these elements (especially tempo and rhythm), he or she can spot bugs from across the room in seconds that would only be apparent after careful analysis of the combined execution traces from all the robots.

5 Limitations and Summary

There are many factors to consider in multi-robot HRI design: ease of use, user workload, information flow, software maintenance, and robot maintenance. The GUI display allows visualization of internal state, but it can become cluttered if not designed properly. The Swarm contains a great deal of data, the task of organizing it for the user requires much future work. The Swarmish lights and sounds are thought-intensive to design, and can only convey limited data. Moving towards augmented-reality visualizations (M. Daily 2003), will allow telemetry to be superimposed on the robots, but this approach requires the user to wear specialized hardware. Our current approaches are practical and usable, but much work to be done on Human-Swarm Interaction remains.

Acknowledgements

The iRobot Swarm Project was funded by DARPA IPTO under contracts SPAWAR N66001-99-C-8513 and SMDC DASG60-02-C-0028.

McLurkin is supported by a grant from Boeing Corporation.

References

M. Daily, Y. Cho, K. Martin, D. Payton, “World Embedded Interfaces for Human-Robot Interaction”. Proceedings of the 36th IEEE Hawaii International Conference on System Sciences, 2003

C. Intanagonwiwat, R. Govindan and D. Estrin. “Directed diffusion: A scalable and robust communication paradigm for sensor networks”. In Proc. Sixth Annual International Conference on Mobile Computing and Networks, 2000.

J. McLurkin. “Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots”. S.M Thesis, Massachusetts Institute of Technology. 2004.

N. Reijers, K. Langendoen. “Efficient Code Distribution in Wireless Sensor Networks”. ACM international conference on Wireless Sensor Networks and Applications, 2003

T. Stathopoulos, T. McHenry, J. Heidemann, D. Estrin. “A Remote Code Update Mechanism for Wireless Sensor Networks” CENS Technical Report # 30. Center for Embedded Networked Sensing, 2003