COMMAND LANGUAGE FOR SINGLE-USER, MULTI-ROBOT SWARM CONTROL

Abraham M. Shultz

3 December, 2018

COMMAND LANGUAGE FOR SINGLE-USER, MULTI-ROBOT SWARM CONTROL

BY

ABRAHAM M. SHULTZ
B.S. WORCESTER POLYTECHNIC INSTITUTE (2004)
M.S. UNIVERSITY OF MASSACHUSETTS LOWELL (2014)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
COMPUTER SCIENCE
UNIVERSITY OF MASSACHUSETTS LOWELL

Author: ..... Date: 3 December, 2018

Dissertation Chair: .....
Dr. Holly A. Yanco

Committee Member: .....
Dr. Radhika Nagpal

Committee Member: .....
Dr. Jay McCarthy

Abstract

Command and control systems designed for a single operator to operate a single robot do not scale to control of swarms. Interfaces that require the user to attend to each robot overwhelm the user when the number of robots increases beyond 12 or 13 for uncrewed aerial vehicles (UAVs) and 3-9 for uncrewed ground vehicles (UGVs). As robot swarms increase beyond these bounds, the control system must shift to from controlling individual robots to controlling the swarm as a single entity while permitting easy and understandable control of the swarm.

Previous work in human-robot interaction (HRI) shows that multi-touch interfaces allow a scalable and direct mapping between the desires of the user and sequences of commands to robots. This thesis presents an interface that extends previous work on multitouch interfaces for small groups of robots to larger swarms, and automates the process of converting command gestures into programs for each robot. The use of individual control programs rather than centralized control is important to realize the potential of swarms to continue to operate despite the failure of individual swarm robots.

The contributions of this thesis are a new swarm hardware platform, software to support it, and a user interface that converts user commands into programs for each robot in the swarm. The new swarm platform initially combined an Internet of Things (IoT) platform with drivetrains from toys to allow large swarms to be built at a low cost. Ultimately, toys were not sufficiently reliable to serve as mobility platforms, so the controller was applied to 3-D printed chassis. The user interface was defined by allowing users to select the gestures that they would use to issue commands to the swarm. It was discovered that as the size of the swarm increases, the gestures that users choose vary, particularly in the case of selection gestures.

The resulting user gesture set, with some modification to remove ambiguity, can be translated into programs for individual robots, but the correctness of these programs is only provable in limited cases.

Acknowledgments

Thank you to Dr. Holly Yanco, for encouraging me to turn one of my side projects into a dissertation, for providing guidance and resources for all these years, and finally for the encouragement to get out. Your ceaseless work to run the Robotics Lab and the NERVE Center and encourage a good mixture of freedom and focus for the people working there have made it a fantastic place to work.

Thank you to Dr. Jay McCarthy for the useful pointers in programming languages and verification, and for keeping me on track when I was getting into the weeds of Turing completeness.

Thank you to Dr. Radhika Nagpal for the loan of the E-Pucks, and for an impressive body of work in swarm robotics that inspired and informed some of this work.

Thank you to Jonathan Roche for vastly speeding up the virtual laser, to James Kuczynski and Dalton Curtin for their attention to detail in coding the user responses, and everyone else in the UML robotics lab that I've asked to read papers, write ROS modules, or otherwise help out. Collaboration is one of our lab's strengths, keep it that way when I'm gone.

Thank you to my parents, for teaching me, for encouraging me to pursue my own education, and putting up with the huge piles of messed-about-with hardware that follow me around. Your unconditional love and support has been a comfort to me in times of stress, and a joy always.

Thank you to Alexander Shultz, Renee Furr, Margaret Lark, Adeline Violas, and anyone else who offered proofreading and other help along the way. If I have left anyone out, know that the blame lies with my memory, and not with your assistance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Methods for command and control that are based on issuing individual orders to individual robots do not scale to large numbers of robots [Wang, Lewis, Velagapudi, Scerri, and Sycara, 2009]. By defining a mapping from user interface gestures to individual programs loaded on each robot, an individual can control arbitrarily large, heterogeneous groups of robots. While swarm hardware is not yet at a point where very complex computation may be pushed directly to the swarm nodes themselves, that time is not far off. Indeed, some systems already have moderately powerful computers, but at fairly high cost for each the individual robots [Millard, Joyce, Hilder, Fleşeriu, Newbrook, Li, McDaid, and Halliday, 2017].

Until computational power in the individual swarm units reaches the levels required for complex computation, virtualization of computing resources can provide an adequate test environment for the development of swarm control algorithms at modest requirements in terms of space and power consumption. Centralizing the control of a single swarm of robots makes the system as a whole sensitive to the failure of the central controller. To avoid this type of failure, the

overall action of the swarm should be guided by decentralized emergent behavior, rather than centralized orchestration. Each robot receives its own program, and the sum of the execution of the programs on each robot results in the completion of the swarm's task. The various approaches to the development of swarm robot control programs show that a wide variety of approaches can still result in robust controllers for swarm robots. However, placing bounds on the sensing ability and communication ability of the robots has substantial effects on the programs that can be developed for them.

Thesis Statement

One potential method to control a swarm of robots is having a central computer dictate to individual robots how the robots should move. However, centralized control is only as robust as the central controller and its connection to the robots. Distributed control systems do not have a single point of failure as centralized models do. In order to create reliable and useful swarm robotic systems, users must be able to specify a desired end state of the system to which the swarm can converge without reliable orchestration from a central controller. Moreover, this convergence must occur in the face of unreliability on the part of the individual swarm members.

The current state of development of emergent control of swarms is guided by ad-hoc, iterative development models that are somewhat suited to software developers, but not suited for use by non-programming end users [Palmer, Kirschenbaum, Seiter, Shifflet, and Kovacina, 2005b]. The motivating examples of uses for swarms are task oriented, such as sending swarm robots into disaster zones to search for survivors. Iterative software development does not have the ability to adapt

quickly enough in the face of changing situations in a disaster area, and software development training would be out of scope for first responders. Therefore, it is desirable to automate the construction of control software for a swarm so that it can adapt to a situation, without requiring significant development time. In order to support interactive control during a developing situation, the construction of the software should occur over a similar time scale to the user interactions.

Initially, part of the intent of this work was to determine if robot control programs could be developed to function under the following assumptions, which mirror some of the difficulties found in operation of robots under difficult field conditions.

1. Robots' sensing is limited in range. Because of this limitation and dynamic environments, the information that robots can have about distant points is limited.

2. Networking between robots is unreliable, due to range, limited power, and possible interference. It is not the case that any robot can reach any other robot at any time.

3. Because of limits in sensing and networking, it may be the case that global, absolute localization is unavailable.

4. Robots can fail. Algorithms to control them should not depend on the perfect functioning of any individual robot.

Ultimately, while the resulting programs can operate under many of these conditions, the user interface (UI) design used in user experiment implies some form of localization for the robots. For the study described in this work, the UI displays, on a multi-touch screen, a top-down view of the robots and their

surroundings. Placing the robots in a map-like view relative to each other implies that the locations of the robots relative to each other can be determined. With this interface, the users were presented with a series of tasks, and could use any method they wished to command the robots to perform the task. Without metric localization, some of the tasks posed in the user experiments are possible, if certain aspects of the goals are relaxed.

Hypotheses

H1: A cheap indoor swarm can be built with commodity hardware

The Kilobots set a remarkably low price point for individual swarm robots, with the parts for each robot costing approximately $15 [Rubenstein, Ahler, Hoff, Cabrera, and Nagpal, 2014a]. However, the Kilobots move by stick-slip locomotion, and so require a smooth, level surface to operate on. Children's toys, such as radio controlled (RC) toy cars, tanks, and legged "insects" are designed to operate on slightly more difficult terrain, and so could be used to extend swarm robotic experiments into natural indoor settings. In order to enable the control of toy mobility platforms as swarm robots, a common controller with the ability to adapt to various toys is needed. As computing hardware decreases in price and size, more and more ability can be built into smaller and smaller hardware. The development and popularity of smartphones has driven the development of smaller sensors and lower power processors, as well as thinner and smaller battery technology. As Internet of Things (IoT) technology becomes increasingly popular, it becomes easier and cheaper to add smaller and lower-power devices to communications networks.

The parts that go into these consumer technologies are also made less expensive by economies of scale. Assuming a fixed set-up cost, the more finished devices are produced, the greater the amortization of the setup cost across the devices. Since IoT is expected to deliver connectivity for tens or hundreds of devices per end user, the expected economics drive down the cost of network connectivity. As a consequence, by using components intended for IoT devices, cell phones, and similar consumer electronics, the cost of building small robots will also continue to drop.

This hypothesis would be disproved by a swarm robot capable of operating in a naturalistic indoor setting requiring more than $30 in parts. The $30 target price was selected under the assumption that a base with sufficient terrain handling capability to operate in a natural indoor environment will, at worst, double the cost of the robot from the lower limit set by the Kilobots.

H2: User gestures change based on the size of the swarm

It is hypothesized that there exists a number of robots beyond which users will transition from treating robots as individuals to interacting with the robots in small groups or as a single large group. As the user interacts with the multi-touch user interface, they will choose the gestures that they feel convey their intention to the system. The collected gestures for a particular user are their gesture set. Across multiple users, the transition point will be apparent because of a change in the gesture set that the users choose to interact with the swarm. It is hypothesized that above the transition point, users will be more likely to neglect some subset of the available robots. The user will instead issue commands that control the bulk of the robots as a cloud or flock, but may leave some robots unused. For example, the user may switch from selecting robots as individuals to shaping and pushing the swarm the way a child might play with a bug, putting their hand down so the bug goes

around or avoids it, touching the back of the bug gently to make it scurry forwards, and so forth, or by shaping the group as if sculpting, with pushing and pinching to "carry" groups around. The user may also change how they indicate which robots are to be interacted with. Rather than selecting each robot by clicking on it, they may "paint" over the area containing the robots they want to use, or draw a circle around them. The size of the swarm where changes in the user gestures occur will indicate the transition point between interacting with individual robots and interacting with the swarm as a whole. This hypothesis would be invalidated by the gestures selected by the user displaying no correlation with the size of the swarm that they are controlling.

H3: Changes in display of the swarm can change user behavior

It is hypothesized that a display that obscures individual robots and displays a cloud or swarm boundary will cause the user to treat the swarm as a whole rather than individuals, which will be apparent because the user will use the same gestures they would use to control a single robot.

Once the ratio of the size of individual swarm members to the size of the area the swarm is in becomes sufficiently large, displaying the swarm members at the same scale as the map will result in the representation of the swarm members being too small to interact with. This problem will arise at smaller scales if the swarm robots are themselves quite tiny, and some of the available swarm robots are indeed small [Pelrine, Wong-Foy, McCoy, Holeman, Mahoney, Myers, Herson, and Low, 2012]. Scaling the representation of the robots up, relative to the map, will make the robot representations overlap unrealistically and obscure the map. Instead, it is proposed that for certain scales of swarms, it makes sense to represent

the swarm as the area covered, rather than the locations of the individual robots. This approach has been used successfully for navigation of a swarm of uncrewed aerial vehicles (UAVs) in three dimensions, by developing a controller that causes the individual UAVs to remain within a bounding prism, and allowing the user to control the shape and location of that prism [Ayanian, Spielberg, Arbesfeld, Strauss, and Rus, 2014].

This hypothesis would be invalidated by the gestures selected by the user in the single robot case being dissimilar from those selected in the case where the swarm is displayed as a cloud or covered region.

H4: User gestures can be converted to programs

It is hypothesized that user commands on a multitouch display can be automatically converted into programs for each robot that will converge to the desired behavior. These programs will operate using only local sensing and local communications, and without resorting to global, absolute localization.

Further, it is hypothesized that the user commands can be represented as a grammar that can be transformed into programs for each robot. These programs should result in the convergence of the swarm to the desired behavior using only local sensing and local communications, and without resorting to global, absolute localization. However, this hypothesis must be modified with a few caveats. First, under the assumption that robots can fail, it is possible that the entire behavior can fail. For example, if enough of the robots are incapacitated, it may be that not enough are left to complete the task. It's also possible that at compile time, the task is still possible, but a later change of the environment renders it impossible. Assessing whether or not a user-specified action will be completed is not possible for all of the usual reasons that prevent prediction of the future, but in some limited

cases, it may be possible to determine whether a specified action is impossible. The goal of this work is to provide a best-effort attempt to satisfy the user command, rather than prove anything about the possibility of doing so. This hypothesis would be disproven by demonstration that a desired user action, as represented by control gestures on the user interface, could not be converted to a program that operates without global localization and that requires only local sensing and communication.

Chapter 2

Related Work

Overview of Previous Swarm Hardware

Swarm robots are generally small. The reason to keep swarm robots small is related to both the cost of making them and the cost of using them. Larger robots consume more materials per unit, and so cost more money. As a result, for a given number of swarm units, larger robots will result in a higher cost swarm. Also, each robot requires some amount of space to move around in. To keep the ratio of free space to robots constant, the area of space used by the robots grows as the robots do. If the ratio is not kept constant, the robots will crowd each other: large robots will require either a very large space, or become overly crowded. Finally, larger robots are more cumbersome to deal with. They require larger storage areas, possibly teamwork to lift or repair, and so forth. All of these efforts are multiplied by the number of robots in the swarm.

In addition to budgetary constraints, interaction with an environment built for humans places an upper bound on the scale of the individual swarm

members. For example, typical indoor doorways are around 80cm wide, so a robot would have to be less than 80cm wide to fit through them. The lower bound on swarm robots is generally dictated by fabrication technology, with smaller robots becoming increasingly difficult to assemble. As detailed below, swarm robots are mostly between $1cm^3$ and $30cm^3$. This scale range divides fairly evenly into robots that can operate in large swarms on a table, and those that can operate in swarms within a room, albeit possibly a large room. The challenge of construction of swarm robot hardware is to put all of the same parts as non-swarm mobile robots into a small package. Many impressive designs for small swarm robot platforms have been proposed and constructed as part of research in swarm robotics. However, most of these platforms are no longer easily commercially available, or never were.

Tabletop Swarms

At the low end, in terms of size, the I-SWARM Project was intended to create a 2x2x1mm robot that moved by stick-slip locomotion actuated by piezo levers [Seyfried, Szymanski, Bender, Estana, Thiel, and Wörn, 2005]. Over the course of the project from 2004-2008, the hardware was developed and used in research, but was not converted to a commercial product. Other techniques have been developed to use magnetic fields to apply force to small magnetic objects, resulting in controlled motion of the objects [Floyd, Pawashe, and Sitti, 2008; Pelrine et al., 2012]. These systems are not amenable to decentralized control, because the moving components are not themselves robots. The moving parts are more accurately viewed as manipulators, with the instrumented environment, any sensors for feedback from that environment, and the manipulators themselves comprising a single robot.

Early small-scale swarm robots were based on microprocessors, and were

primarily research platforms for the groups that developed them, rather than commercially available products. Alice combined a PIC16F84 processor, motors, RF and IR networking, and enough battery power for 10 hours of autonomy into a robot measuring under 2.5cm^3 [Caprari, Balmer, Piguet, and Siegwart, 1998]. The Jasmine swarm robots were possibly the closest thing to a commercially-available successor to Alice [Kernbach, 2011]. Jasmine measured 26x26x20mm, and included an ATMega processor, IR close range communication and obstacle detection, two motor skid steering, and lithium-polymer batteries. Jasmine units cost about $111 each when they were available, and they are no longer available for purchase. InsBot was a small robot, measuring 41mm x 30mm x 19mm, that was designed to interact with cockroaches [Colot, Caprari, and Siegwart, 2004]. It used two processors, one to run higher level behaviors and one to interface with a suite of sensors that included 12 IR sensors and a linear camera. The AmIR robot measured 6.5cm in diameter and 6cm tall. It has a more modern processor than Alice [Arvin, Samsudin, and Ramli, 2009]. There is no evidence that AmIR was ever widely available, and it cost $92 per unit [Arvin, Yue, and Xiong, 2015b]. The successors to Amir, Colias, and the related Colias-Φ, are dual-microprocessor systems similar to Jasmine in functionality, with additional features for pheromone robotics applications [Arvin, Murray, Zhang, and Yue, 2014; Arvin et al., 2015b]. Colias is built out of two PCBs, with the upper PCB and processor handling IR collision avoidance and communication, and the lower processor controlling the motors, power management, and various sensors. Colias robots cost $35, in parts, but are not commercially available. The Colias-Φ model has an even more impressively low price, at $22.

Even when they are commercially available, most existing swarm robots are too expensive to build a large swarm. The E-puck from EFPL is approximately $795 per unit, so the cost of maintaining a large swarm can become daunting quickly.

The high price of the E-puck is a result of its extensive suite of sensors, including a camera and 360° IR range sensor and communication system. As of January 2018, the E-Puck 2 has been commercially available. Version 2 E-pucks have a 168MHz, VGA-resolution camera, and WiFi connectivity in addition to the bluetooth that all E-Pucks use. They cost 850 Swiss francs, or approximately $845 as of this writing.

The r-one research robot is cheaper than the E-puck, at approximately $220 per unit [McLurkin, Lynch, Rixner, Barr, Chou, Foster, and Bilstein, 2013]. The developers of the r-one position it as a more featureful and less expensive alternative to the E-puck ($795), Parallax's Scribbler ($198, minimal sensors), the iRobot Create ($220, requires additional hardware to be programmable), the K-team Khepera III ($2000), or the Pololu 3pi ($99, minimal sensors).

The Harvard Kilobots are a more recent entry to inexpensive swarms, and have been produced in large quantities [Rubenstein et al., 2014a]. Kilobots contain $15 worth of parts, while a 10-pack of assembled Kilobots sells for about $112 per robot. The Kilobots are intended for research in a highly homogeneous environment, with most or all of the robots executing the same program. As a result, they are designed to be programmed in parallel using an IR interface. For small groups, individual Kilobots can be programmed differently, but any attempt to give each of a very large collection of robots an unique program will take a long time. The Kilobots also move by stick-slip motion, and so must operate on a smooth surface, such as a whiteboard.

The GRITSBots platform is a differential-drive platform using stepper motors [Pickem, Lee, and Egerstedt, 2015]. GRITSBots use the same processor as Colias, in a similar configuration, with one processor operating sensors and the other controlling the robot's motors. The sensor board incorporates a 3D accelerometer and gyro as well as a 6-direction IR distance sensing ring. GRITSBots measure

31mm x 33mm, and cost approximately $50 for parts per unit.

The Psi Swarm robot is a 10cm diameter round robot with proximity sensing, a compass, bluetooth wireless connectivity, and autonomous charging via ground contacts [Hilder, Horsfield, Millard, and Timmis, 2016]. As with many of the platforms described in this section, it is not commercially available, but the designs to have the circuit boards fabricated and the body 3D printed are freely available.

MROBerTO is a swarm robot with a $16mm^2$ footprint. It has modular expandability via a header for daughterboards, and includes a single-point laser rangefinder, gyro, accelerometer, compass, and a VGA resolution camera [Kim, Colaco, Kashino, Nejat, and Benhabib, 2016]. The mROBerTO processor is a 32-bit ARM processor with Bluetooth and ANT+ wireless. All of this hardware is only $60 per unit in quantities of 25 or more units. In order to permit an overhead camera to localize the mROBerTO robots, a single RGB LED on top of the camera can be lit in a unique color to localize the robot, and a green LED on the front of the robot indicates its heading. The use of color information is likely much faster to process than fiducual tags, but does have the disadvantage that it is only useful in 2D unless a stereo camera is used.

The Zooids of Le Goc *et al.* are interesting in that the robots themselves are positioned as both a physical interface and a swarm [Le Goc, Kim, Parsaei, Fekete, Dragicevic, and Follmer, 2016]. They are designed to be used as physical controls, such as knobs or sliders, as well as being able to move themselves. The individual robots measure just under 3cm in diameter, and localize themselves using a projected grey coded light signal from a high-speed projector. Le Goc *et al.* estimate the individual cost of a Zooid at around $50.

Swarm robots have also been developed for aquatic and aerial robotics as

well, although the problems unique to those domains are outside of the scope of this work [Costa, Duarte, Rodrigues, Oliveira, and Christensen, 2016; Preiss, Honig, Sukhatme, and Ayanian, 2017]. It is worth noting that the Crazyflie quadcopter platform on which the Crazyswarm is based is a commercial product. In single quantities, it costs $180 per quadcopter. At least one inexpensive, legged robot platform has also been proposed as a swarm platform, but no swarm composed of them appears in the literature [Kalat, Faal, Celik, and Onal, 2015].

One way to reduce the cost of swarm robots is to use commercial, off-the-shelf (COTS) hardware in the construction of the robot. Reusing existing hardware leverages the economies of scale that reduce the price of commercial hardware, as well as eliminating the need to design or build the COTS parts. Use of COTS parts in research robotics has led to at least two platforms referred to as COTSBots [Bergbreiter and Pister, 2003; Soule and Heckendorn, 2011]. The first COTSBots used mote hardware for the communications link and sensing, plus a motor control add-on board [Bergbreiter and Pister, 2003]. The mobility platform is a modified toy, in particular, a specific brand of high-quality micro radio controlled car. At the time of this writing, the car used in COTSBots is moderately expensive for a toy car, costing a little over $100 per unit. COTSBots use TinyOS, a modular and event-driven framework for developing software for low-power wireless devices [Levis, Madden, Polastre, Szewczyk, Whitehouse, Woo, Gay, Hill, Welsh, Brewer, et al., 2005].

Room-sized Swarms

One potential problem with extremely small swarms is that while the robots may scale down, the obstacles they have to traverse do not scale with them. This sort of vulnerability prevents the smaller, tabletop swarm robots from operating well

in human-scaled environments. In order to overcome this problem, larger swarm robots can be constructed.

The MarXbot swarm platform is capable of operating in unstructured human environments. It measures 17cm across by 29cm tall, and uses a combination of tracks and wheels for mobility. MarXBots can also use their grippers to link themselves together and perform operations that an individual robot could not perform, such as bridging a gap larger than a single robot [Bonani, Longchamp, Magnenat, Rétornaz, Burnier, Roulet, Vaussard, Bleuler, and Mondada, 2010]. The size and complexity of the MarXbots, as well as their powerful computer, likely rendered the individual robots quite expensive, but their price does not appear in the literature.

Swarmanoid extends the interlinking mechanism of MarXbot to a heterogeneous swarm with three different types of robots [Dorigo, Floreano, Gambardella, Mondada, Nolfi, Baaboura, Birattari, Bonani, Brambilla, Brutschy, et al., 2013]. The "foot" robots are MarXbots, and provide ground motion for "hand" robots. "Hand" robots have grippers to manipulate objects, and can also climb. The "hand" robots have an attachment point like the MarXbots, and so can be carried by "foot" robots. Flying "eye" robots provide overviews of the work area and networking.

In order to reduce costs, another platform called COTSBots was developed [Soule and Heckendorn, 2011]. Instead of sensor motes on micro-scale RC cars, the newer COTSBots platform is composed of a laptop for processing and a modified RC car, tank, or similar toy as a mobility platform. In order to interface between the laptop and motor drivers, a second micro-controller board, such as an Arduino or Phidget interface, may be used. Due to the diversity of possible combinations of hardware that can be assembled into this configuration, it is still a very viable platform. However, the minimum size of this style of COTSBot is the size of a

laptop, which is in turn dictated largely by the minimum size of a useful keyboard. The large size of these COTSBots demands a very large space if the density of robots in a large swarm is to be kept low. Additionally, each laptop has a screen, keyboard, and so forth that are not useful while the robot is operating. All of these parts add to the overall cost of the swarm.

Pheeno is an inexpensive robot of approximately the same scale as the MarxBots [Wilson, Gameros, Sheely, Lin, Dover, Gevorkyan, Haberland, Bertozzi, and Berman, 2016]. It has an optional gripper module, and uses a Raspberry Pi miniature computer for its main processing power. The developers of Pheeno provide a comparison with other robots in the same size range, which cost from $150 for the Parallax Scribbler 2 to over $3,000 for the much more sensor-rich Khephra IV. Pheeno itself costs $270 in parts, with a $80 optional gripper.

Beyond the scale of rooms, swarm research has been done with Amigobots and Roombas, as well as larger custom platforms for outdoor multi-robot work [Guo, Hohil, and Desai, 2007; Tammet, Vain, Puusepp, Reilent, and Kuusik, 2008; Olson, Strom, Goeddel, Morton, Ranganathan, and Richardson, 2013]. In theory, swarm research could be performed using robots of any size, but financial limitations would place it out of the reach of most academic organizations.

Swarm User Interface Designs

The user interface to a swarm has two functions. The first is to allow the user to provide input to the swarm, so that the user can direct the swarm to perform tasks. For the purposes of this research, the user interface is a multitouch surface that displays representations of the area the swarm is in and of the individual swarm robots. The second function of a swarm user interface is to display information

about the swarm, or to display information gathered by the swarm to the user. By providing an overview of the activities of the swarm, the user interface can give the user feedback on the progress of the task as it proceeds, as well as allowing the user to detect problems.

Multitouch interfaces have been determined to improve on "window, icon, mouse, pointer" (WIMP) or voice interfaces for multi-robot control in a sequence of command and control tasks, including commanding the swarm to a location, performing reconnaissance, and having the swarm cross a dangerous area [Hayes, Hooten, and Adams, 2010]. The interface displayed the locations of the robots on a directly manipulatable map, and used movable or semi-transparent user interface widgets, in order to minimize occlusion of the map. Areas were selected with drawing gestures, and paths with fluid strokes, rather than, for example, selection of vertices bounding an area. The use of multi-touch interaction is desirable because one-at-a-time selection does not scale beyond a very limited number of robots. In order to interact with large groups of robots, the user must be able to perform operations on areas and groupings, rather than on the single point available with a traditional pointer-based interface.

Because of the limitations on individual sensing, especially in the case of a robot with poor localization, providing a third-person user interface rather than a first-person one is better for control of swarm robots [Kapellmann-Zafra, Salomons, Kolling, and Groß, 2016]. Kapellmann-Zafra *et al.* provided a user interface that allowed users to control swarms through the influence of teleoperated leaders. Because teleoperation places the user in the point of view of the individual robot being operated (as with many combat UAV interfaces), the user had some difficulty acquiring an idea of the overall disposition of the robots. Once the user was provided with a third-person, "eye in the sky" view, their performance increased.

Users with global information were able to aggregate 90% of the robots, as opposed to approximately 50% in situations without global information.

One approach to getting feedback from a swarm was the development of the Swarmish sound and light system [McLurkin, Smith, Frankel, Sotkowitz, Blau, and Schmidt, 2006]. Swarmish provides an ambient means of determining the overall state of the swarm, as well as some information about individual robots. The swarm that used Swarmish had autonomous charging, and so the individual robots had long runtimes, and minimal one-on-one interaction with humans. The "ambient" aspect of the interaction is that the information is continuously available, and the human user "tunes in" to it when needed. Swarmish uses a set of colored lights and sounds, produced by each robot, to provide feedback. The lights were in three colors, and had a total of 108 different combinations of colors and blink sequences, as a visual indicator of the state of each robot. In addition to the lights, each robot could produce MIDI notes over its audio system. Each note could vary in instrument, pitch, duration, and volume, as well as having tempos and rhythms as the code executes. The designers of Swarmish indicate that the sum of the audio output of the swarm could provide a overall idea of the status of the swarm, but that as a musical instrument, it is difficult to play well. Further, the use of lights as signaling mechanisms assumes that the user or operator can see the robots.

If we accept the assumption that the robots are visible to the user, the robots can carry some form of display that provides local information to the user. This information can then be displayed as an overlay in the real world, with the display of the information conterminous with its presence [Daily, Cho, Martin, and Payton, 2003]. For example, if each robot has a gas sensor, and a light that it can illuminate in response to detected gas, then the user can look at the swarm, and see which areas of the swarm are detecting gas. Local display of local information works

if the user is part of a hybrid human-robot team, and so is in the same location as the users. However, there are many situations where the robot is not in the same location as the user. A common example is urban search and rescue, where buildings may be known to be unsafe, or of unknown stability, but it is desirable to search them for trapped people. In such a situation, the human user would rather be located elsewhere, and receive information from the robots.

For situations where the user is not located in the same area as the robots, one possible approach is a "call center", where robots can request human attention when required [Chen, Barnes, and Harper-Sciarini, 2011]. The human in the call center, however, is faced with having to answer potentially multiple calls with no awareness of the robot's situation. The theoretical basis for call center UI is Supervisory Control. Supervisory control has the human act as the planner and monitor of the systems being supervised, but allowing the systems to operate on their own. Automation is frequently broken down into ten levels of automation, with level ten being a fully automatic system with no humans involved, and level one having no automation, such as a bicycle [Parasuraman et al., 2000].

It would be expected that reducing the number of times the human is required to interact with the robot will permit the user to operate more robots. With level one automation, the user has to interact constantly, and so could not be expected to operate more than one robot. By increasing the level of autonomy of the robot, the time required for the user to operate the robot decreases. Instead of continuous interaction, the user can specify actions for the robot to undertake, and then ignore the robot while it performs the actions. It is expected that the robot's effectiveness will decline over time since the last user interaction. This time that the robot operates without interaction before its effectiveness declines to a fixed minimum is called "neglect time"[Olsen and Goodrich, 2003]. With increasing

1. Robot does nothing, human makes all the decisions.
2. Robot presents complete set of possible actions.
3. Robot presents a proper subset of possible courses of action.
4. Robot presents single suggestion.
5. Robot executes single suggestion on human acceptance.
6. Robot allows human to veto actions, acts if not vetoed.
7. Robot performs automatically and then informs the human.
8. Robot informs the human if asked.
9. Robot informs the human if it decides to do so.
10. Robot decides and acts autonomously, no human input.

Figure 2.1: The 10-level model of autonomy from [Parasuraman et al., 2000]

autonomy, neglect time increases.

At the higher levels of the autonomy scale, the robot's neglect time far outweighs the time the user is expected to operate it, and so the user could reasonably be expected to operate other robots during the neglect time.

Increasing neglect time may allow the user to operate more swarm robots, but it comes at a cost. The longer a user goes without learning about the state of one of the robots, the less idea they will have of the robot's situation when they are called upon to operate that robot. The problem of automation decreasing the situational awareness (SA) of the user has been described in cockpit automation for aircraft [Wiener and Curry, 1980], and generalized well to other systems that combine automation with human control [Kaber and Endsley, 1997]. If the user takes a long time to relearn the situation, the efficiency of the system will drop. Worse, the user may make errors because of an incorrect understanding of the system when they begin operations after a long neglect time [Cummings and Mitche, 2008]. One possible approach to maintain a constant and manageable workload on the user is adapting the level of automation to the workload. When the load is low, the user is more directly engaged, but when the load is high, there is more automated assistance. By varying the level of automation, the workload for the user

is kept constant. A constant workload is desirable because the user remains engaged with the work, and so has an ongoing understanding of the situation as it develops. The user is not suddenly called into a situation after remaining disengaged for some time. However, the workload must also be manageable. If the user is overloaded, their attention will become subject to triage, and they will begin to miss elements of the task. Adaptation does not have to be based on measured load, but could instead be based on perceived load or physiological markers in the user.

However, in situations with even moderate numbers of robots, even relatively high levels of automation may overwhelm the user [Lewis, Polvichai, Sycara, and Scerri, 2006]. Level five, operation by consent, is a fairly high level of autonomy, but with a large number of robots checking in, even this level may generate too many events for the human to deal with. Increasing the autonomy to level nine, so that the robots are only checking in with the operator when an exceptional situation occurs, may still overwhelm the operator if enough robots are active. Increasing the use of automation may also create new difficulties by leaving operator out of practice, or encouraging mis-placed trust in the automation's ability [Lee and See, 2004].

In fact, any kind of multitasking may prove insufficient for large swarms. Due to their strategic potential, research in human-swarm interface for UAV swarms is a rapidly developing field [Hocraffer and Nam, 2017]. Potential issues for an interface can be cognitive limitations of the users, or actual design problems in the interface. Research in the field attempts to measure the difficulty of using the interfaces in a number of ways, including cognitive workload, task effectiveness, effective use of operator time, and situational awareness. Perhaps due to the expected mission parameters, very few studies are performed using swarms of more than tens of robots. For teleoperation, the best case is uncrewed aerial vehicles

(UAVs), which require relatively little oversight. Uncrewed ground vehicles (UGVs) require more oversight than UAVs, due to the higher complexity of the ground environment. Estimates place the limits on the number of robots under control at 12 or 13 for UAVs and 3-9 for UGVs [Wang et al., 2009]. There is some latitude, at least in UGVs, to increase multitasking by increasing automation, as shown by the relatively wide range in the interaction limits, but even 9 robots per operator is nowhere near the scale of kilo-robot swarms [Olsen and Wood, 2004]. Failure generally takes the form of task effectiveness no longer increasing as more robots are added. Instead, the amount of time the user spends interacting with the robots begins to outweigh the neglect time, and so the robots spend increasing amounts of time waiting for interactions [Cummings and Mitche, 2008].

Ecological interface design (EID) presents a possible guide to the architecture of user interfaces for swarm robotics, and has been used in interfaces with mixed human-robot teams [Vicente and Rasmussen, 1992; Gancet, Motard, Naghsh, Roast, Arancon, and Marques, 2010]. In EID, a user's abilities that enable them to interact with a system is separated into a taxonomy of skills, rules, and knowledge. The user has skills, which are rote, simple activities that form the basis of the normal operation of the system. The user also knows a set of rules about the system. Rules allow the user to handle exceptions or unusual cases that have come up before. Rules do not require the user to understand the system, just to know that when certain situations are recognized, certain actions must be performed in response. Beyond rules and skills, the user also has knowledge of the system. Knowledge gives the user an understanding of how the system works, which they can apply to react to situations that the user has not experienced or been told about before. Events are also broken into three levels: routine, which uses skills; foreseen exceptions, which use rules; and unforeseen exceptions, which use knowledge. All

levels should be supported by the interface, but the user should not be forced to operate at a higher level than is required. The abstraction of the process maps onto the hierarchy of ecological design, with the highest level being the function of the process and the lowest level being how the function is accomplished. At each level, there are constraints on the process that are used to define the normal operation of the process. Detection of exceptions requires the display of all constraints, because an exception is the breaking of constraints, and undisplayed constraints cannot be assessed to determine if they have been broken.

The user should be able to extract meaning from the information display quickly. By using the lights in Swarmish, the user can assess the state of individual robots, but by listening to the overall sound of the swarm, the user can also assess the behavior of the system as a whole. The state and status lights of an individual robot is the low level in EID, watching how an individual action of the overall process is progressing. The "tune" of the entire swarm, produced by the sum of their MIDI notes, provides the high level overview, where a user can tell if the system is progressing well or developing problems. As the system changes, the changes and predictions should be highlighted so that the user understands consequences of their actions. In Swarmish, sudden changes in the tone or tempo of the swarm tune indicate transitions in its behavior, without the user having to observe the actions of the robots closely.

EID is well-positioned to deal with emergent behavior, because the emergent behavior of the entire system is present at the functional level, but is composed of actions at the physical level. The control of swarm robots can be viewed as a hierarchy of increasing abstraction. At the least abstract, base level are the individual interactions of the swarm robots with each other and their environment, as dictated by their explicit programs. Above that level is the implicit, emergent

behavior of the swarm as a whole. Finally, the most abstract level is the user intent, as expressed in the interface through their gestures. This hierarchy corresponds well to the abstraction of process in EID, with discrete physical actions at the lowest level and the overall results of the process at the highest level. Consequently, the user is permitted to issue commands in the most abstract domain, and the system can propagate them "downwards" into the concrete actions of the robots in the world, while also propagating information from individual robots "upwards" into the global view.

Automation in EID allows the user to operate primarily with rules and knowledge, dealing with exceptions [Vicente, 2002]. The interface should allow direct manipulation of perceptual forms that map directly onto work-domain constraints and represent all of the information identified by the abstraction hierarchy. In a swarm context, this means displaying functional information in such a way that the user can move across the hierarchy from individual swarm robots to high-level swarm-wide tasks, and interact at all levels to control the swarm. More practically, this means that the information displayed must be integrated in such a way that the mapping from one unit of information to another is made apparent in the interface, rather than offloaded to the user to compute in their head [Yanco, Drury, and Scholtz, 2004]. For example, if a robot can send video and range information, the information can be projected into a 3D space around the robot, rather than being displayed in separate UI windows. Such a projection allows the user to easily relate visual and range information, and relate that information to the ongoing robot control task, which in turn increases task performance [Ricks, Nielsen, Goodrich, et al., 2004]. Previous work in multi-touch interfaces directly satisfies these requirements of EID by providing both an omniscient camera view for direct manipulation of the high-level, functional actions of the entire swarm, and the ability to move down the

hierarchy to control individual swarm members [Micire, Desai, Courtemanche, Tsui, and Yanco, 2009]. The ability to display information about individual robots along side or on top of the interface representation of the robot is an important method of providing feedback to the user [Kato, Sakamoto, Inami, and Igarashi, 2009].

UI Designs

The user interface may be able to drive the re-imagining of the robots as a unified swarm, and so alter the user's interaction with the swarm. The base case is to simply display all the units as individuals, but this may not be useful for the operator [Coppin and Legras, 2012]. Heuristic evaluation has been performed on several methods, including an amorphous shape covering the area occupied by the swarm, an amorphous shape with density shading and motion arrows, the fields of influence for leaders in the swarm, and the web generated by the flow of information within the swarm [Manning, Harriott, Hayes, Adams, and Seiffert, 2015]. The utility of different methods varies with the desired task. Considered as a whole, the swarm has properties, such as center of gravity or flock thickness, that do not exist in individual robots. Views of these properties may assist the user, for example in determining what areas have insufficient robot density for a thorough search operation.

The information available to the user through the UI also implies the availability of certain information within the system. The distinction between UI representations of the swarm that display each robot as an individual robot versus those that display a cloud or amorphous shape in the area occupied by robots is the most obvious example. A system that displays the location of each robot must actually have localization information about each robot. The presence of this information in turn implies that the localization information can be used to

plan the actions of each robot, which in turn affects the structure of the programs generated for each robot.

For example, if the task assigned to the swarm is to surround a fixed point, and localization information is available, then each robot can be given a program that instructs it to move towards a known location, based on its current known location. Even if the robots cannot determine their location, but the UI and program generator have it, the robots closest to the point can be assigned programs that cause them to act as beacons, while all the other robots are assigned programs to wander until they see a beacon and then move towards it. If, instead, neither the robots nor the program generator have information on the location of the robots, then all of the robots can be assigned programs that instruct them to wander until they detect the target point, and then act as beacons, at which point the overall behavior of the system returns to the previous example.

In the most extreme case, neither the robots nor the user interface have any information about the location of the robots. As a consequence, the system could not display the individual robots situated in relation to each other on some form of map. This extreme is outside the scope of this work, as it is more suited to an interface that permits the provisioning of the robots with a description of the target point. A method for providing such a description through multitouch gestures is likely to be more tedious than other approaches, e.g. summarizing desired sensor precepts or including an image of the target area for the robots to recognize. However, if the user task is expressible in terms of the overhead view of the area, the user interface could simply allow the user to issue commands that are situated in that view, such as rallying at a certain point or moving an object that is visible in overhead view. Without information about the robot positions, the user would not be able to watch the motion of the robots to see that the commands

were being followed.

Multitouch Gesture Discovery

Previous work in multitouch user interface gesture sets can be broadly separated into two classes: those that attempt to build a general gesture set, and those that attempt to build a gesture set to be used for a specific task.

General gesture sets would be for operations such as the cut/copy/paste editing metaphors, which show up in word processing, image editing, and other productivity applications. These operations act in differing ways, depending on their context, but are conceptually similar, e.g. paste always inserts information that was previously cut, but the information type may vary. The operations are also invoked in the same way across applications and even across operating systems.

Wobbrock, Morris, and Wilson use an interesting technique to elicit general user-interface gestures from non-technical users [Wobbrock, Morris, and Wilson, 2009]. The user is shown the *effect* of a command, and then asked to perform the gesture that *caused* that effect. The users were asked to think aloud, in order to understand their cognition about the system and gestures, in addition to their behavior. The experiment used 27 commands: move a little, move a lot, select single, rotate, shrink, delete, enlarge, pan, close, zoom in, zoom out, select group, open, duplicate, previous, next, insert, maximize, paste, minimize, cut, accept, reject, access menu, help, task switch, and undo (listed in order of complexity as ranked by Wobbrock *et al.*). Many of these commands are related to window managers, or occur in some form in multiple applications, such as the cut/copy/paste metaphors.

Task-specific gestures are for operations such as flying through a 3D rendering of an architectural space, or transposing cells in a spreadsheet. While

somewhat intuitive gestures may exist for both operations, the operation is tightly bound to the task at hand, and the gestures would likely be different.

Yao, Fernando, and Wang develop a set of task-specific gestures for urban planning from the required functionality of the interface and paper prototyping on a table [Yao, Fernando, and Wang, 2012]. The gestures were collected by placing a map on the table, and asking users to perform the gestures they would use to access the required functions. The resulting interface is modal. Depending on the selected mode, different gestures are available. The interface also permits the combination of some gestures, such as panning or rotating the map while zooming in or out at the same time.

Micire *et al.* developed a taxonomy of user gestures using a process similar to Wobbrock *et al.*, but did not show the results of the actions. Rather, the users were asked to perform tasks, and could use any gesture they chose. The gestures controlled both the actions of the robots and the behavior of the interface to control them [Micire et al., 2009]. The interface control gestures are for actions such as zooming in on the content of the screen or panning around a map. Individual users chose a variety of gestures to perform the tasks, but for almost all types of tasks, having two gestures available to perform it would be sufficient to cover 60% of the users. It was also noted that the gestures that users use are informed by their previous experience both with traditional mouse-oriented windowing interfaces and with touch-screen technology. Since smartphones are a common multitouch interaction device, it would be expected that smartphone users expectations of multitouch interaction would be informed by their phones. Micire *et al.* confirmed this, finding that iPhone owners used significantly more pinch gestures than participants with no iPhone experience.

The more general gestures sets, such as those from Wobbrock, Morris,

and Wilson, could be used in a gestural window manager or operating system interface. They are not tied to a specific application or working with a specific type or presentation of data. The gesture sets developed by Micire *et al.*; Yao, Fernando, and Wang; or in this work, are all intended to provide a more precise set of meanings for a single application and task.

Multitouch UI Design Concerns

Multitouch user interfaces do not have an agreed-upon standard for the gestures used in interaction with them, largely due to the relative novelty of the technology. Early in the development of multitouch user interfaces, it became apparent that the window, icon, menu, and pointer (WIMP) interface would not be the best way to interact with novel interface technologies [Van Dam, 1997]. WIMP interfaces are tied to a metaphor of a desktop, where a user interacts with 2D spreadsheets and documents. To extend to information with higher dimensionality, the interfaces become more complex and less direct. The somewhat prescient model proposed by van Dam is a multi-modal interface of gestures and natural language, as is found in modern smartphones. This paper also shows an early example of a two-handed manipulation of a virtual object in a 3D environment, which presages the direct interaction style of multitouch user interfaces.

The directness of user interactions can be understood to be the degree to which the interaction resembles the same interaction with physical objects, such as flipping pages of a calendar. With a physical calendar, one can turn pages by sliding a finger over them. Having a slider or buttons to switch months in a virtual calendar is thus less direct, and with a mulitouch interface, can be replaced by the same finger motion that would flip a page of a physical calendar.

This sort of emulation of design elements of an object the same when

the object is moved to a new material or created by new techniques is called "skeueomorphism", and the copied design elements are "skeueomorphs" [Gross, Bardzell, and Bardzell, 2014]. There are several reasons for skeueomorphs in design of user interactions. One reason is effectively kitsch. Design elements can be kept, despite not having a function in the new medium, simply because people find it entertaining to keep the appearance. The vestigial jug handles on bottles of maple syrup or the yellow "paper" background of Apple Notes prior to its redesign are an example of this type of skueomorphism. The jug handles are too small to use as handles, and the "paper" of Apple notes has no effect on the functioning of the application.

Another reason to keep a design element, especially in multitouch UIs, is that the design element may provide hints to the user as to how the object can be interacted with. For example, in displaying photos, if the photo has a slight border, and the border appears to curve and "lift" off the screen background slightly, it will imply that the photo is a mobile object with a small space under it, and so could have other photos under or on top of it. If the photo is displayed totally flat, without a border, it will look more like a sticker or part of the background, and not imply that it can be moved. In neither case is the image actually "above" the background, but the visual impression to the user provides a clue for interacting with it. The interface itself can be viewed as a message from the developer to the end user, which hopefully conveys enough information about the design of the interface to the end user that they are able to use it [Derboven, De Roeck, and Verstraete, 2012].

In the study of interactions with tangible interfaces, it has been proposed that the use of skueomorphic interfaces can also bridge between these two reasons for their use [Gross et al., 2014]. Rather than simple functionality, or pure amusement,

the use of skueomorphic interface designs can convey messages about abstract qualities of a product to a user. One example given by Gross, Bardzell, and Bardzell is a digital synthesizer with a user interface that copies the knobs and switches of physical analog synthesizers. While this is likely not an optimal interface for controlling the synthesizer, it has connotations of quality, performance, and reference to more traditional electronic instruments, and so appeals to the user's idea of themselves as an artist and a participant in a tradition of artists.

Human/Swarm Interaction

There does not appear to be a consensus on how the user interface for a swarm should work. Indeed, the term "Swarm UI" is sometimes used to describe the user interface to a swarm, and sometimes used to describe a user interface which is itself a swarm [Le Goc et al., 2016; Suzuki, Kato, Gross, and Yeh, 2018].

Swarm interaction is distinct from multi-robot HRI. Multi-robot HRI interfaces typically allow the user direct control of individual robots, which can be switched between, but generally do not treat the group as a whole. An example of a multi-robot HRI study is the assessment by Humphrey *et al.* of the scalability of a multiple-robot control interface that uses a "halo" around the view from the currently-controlled robot to provide awareness of the positions of the other robots [Humphrey, Henk, Sewell, Williams, and Adams, 2007]. While this interface did appear to have a positive influence on the number of robots the user could control, it is not a swarm interface, but an interface for individual control of multiple robots.

Previous work in human-swarm interaction (HSI) shows an interesting diversity of methods to convey a command to a swarm. One approach, for co-located swarms, has the user making hand signals to the robots. If the swarm is equipped with cameras, the hand signs can be recognized visually by members of the swarm

[Nagi, Giusti, Nagi, Gambardella, and Di Caro, 2014b; Giusti, Nagi, Gambardella, Bonardi, and Di Caro, 2012; Nagi, Giusti, Gambardella, and Di Caro, 2014a]. Because the recognition is performed by multiple robots at once, with different views, the robots can confer among themselves to arrive at a better understanding of the sign used. Another approach to gesture control of swarms is to have the human and gesture recognition hardware separate from the swarm [Alonso-Mora, Lohaus, Leemann, Siegwart, and Beardsley, 2015]. This approach also allows the software to use the user's gestures to select part of the swarm by natural methods, such as pointing at robots.

Another approach is the management of attractive and repulsive forces in the interface, which are conveyed to the robots. This style of interface maps reasonably directly to at least three of the programming paradigms used in swarm robotics. In physicomimetics, the attractors or repellents are points in the field that act on the robots through the same forces that the robots use in their interaction with each other. To cause robots to gather in an area, the user could place a source of virtual gravity there, which would draw robots towards it. In pheromone robotics, the effect would be much the same, but would be expressed as a pheromone source that diffused into the space and attracted robots. In vector fields, the vector field itself would be changed so that all vectors pointed towards the desired location. One implementation of a human-swarm allows the human operator to describe the desired density of robots at locations within an operating area [Diaz-Mercado, Lee, and Egerstedt, 2017]. The use of density, rather than robot count, scales with the swarm size. The desired configuration of the swarm can be drawn by the user on a tablet, and then converted to an update control law for the robots. Alternatively, the user can directly place Gaussian functions in the space and vary their parameters. The resulting desired densities are smoothed to avoid discontinuities.

The use of density over location is similar to a number of interface methods that function by allowing the user to specify attractors and repulsion points in the swarm space [Goodrich, Pendleton, Sujit, and Pinto, 2011; Brown, Kerman, and Goodrich, 2014; Vasile, Pavel, and Buiu, 2011; Kira and Potter, 2009]. Goodrich *et al.* use an attractor model under two different design metaphors, physicomimetics and a biomemetic model based on the dynamics of schooling fish, to solve an abstract information foraging problem [Goodrich et al., 2011]. The human influence is applied to a swarm system that is capable of solving some aspects of the problem without supervision, but is improved by the addition of human control. In keeping with the bio-inspired metaphor, the human can either control a leader, which the other members of the swarm are attracted to and follow, or a predator, which repels and can chase other members of the swarm. Leader-based models tend to lead to more stable, long-running interactions, as the members of the swarm move to remain near the leader, increasing the duration of the human operator's influence on the leader. In contrast, predator models result in high turnover of the swarm members that the human can influence, as the "prey" elements of the swarm act to get away from the predator's influence. This turnover is part of the two invariants proposed for swarm interaction by Brown *et al.*, which are that the collective state is the fundemental perception of the swarm, rather than the individual robot, and that the ability of a person to influence and understand the collective state is determined by the balance between span and persistence [Brown, Goodrich, Jung, and Kerman, 2015]. Persistence is the duration of each interaction with the swarm, and span is how many units the user interacts with. In a leader-based control UI, the span and persistence are both kept high, because the swarm elements are attracted towards the leader and remain under its influence for longer. In the predatory/repellent model, the span and persistence are low, because the other swarm elements flee

from the user-controlled predatory unit.

Brown, Kerman, and Goodrich demonstrate that a swarm can be controlled to form two different higher-level structures, a flock and a torus, by managing abstract attractors rather than individual behaviors. Further, the transition between modes was amenable to human control, so the overall behavior of the swarm could be configured with a relatively small set of tunable parameters. While the paper only indicates two possible modes, the torus and flock, these two behaviors are a relatively good match for the publicized behavior of the Perdix military drone swarm. Flock is a useful behavior for organized motion to a location, while the torus mode is good for loitering and overwatch of an area, matching the point-orbit, and move-to-point functions that are visible in videos of the Perdix UI. However, public materials on the Perdix system indicate that it uses a playbook interface, where the user selects from a set of available "plays", which the swarm then executes, rather than an explicit management of attractor parameters [Strategic Capabilities Office, 2015].

Parasuraman, Galster, and Miller examined the use of playbook, waypoint, and combined playbook and waypoint control on robot teams in a capture-the-flag task [Parasuraman, Galster, Squire, Furukawa, and Miller, 2005]. Playbook control provides a delegation layer to the interface, where the user does not have to specify the minutia of how a task is completed. The use of delegation decreases mission time and increases success, in this case, winning the game of capture-the-flag.

There have been previous studies of the behavior of users in control of swarms with poor localization and bandwidth constraints [Nunnally, Walker, Kolling, Chakraborty, Lewis, Sycara, and Goodrich, 2012]. These constraints are realistic as the motivating examples for swarm robotics, urban search and rescue or planetary exploration on e.g. Mars, take place in locations where high-speed and

high bandwidth networking infrastructure may be absent. The study describes a task similar to foraging, but under human control, where a 30-robot swarm is used to find a sequence of targets. The interface is mouse-based, and allows the user to issue two commands, "head towards the selected point" and "stop". The robots had Gaussian error added to their estimated location and orientation, meaning that the robots could start on a heading that was not exactly towards the selected point, when the user gave them a point to move towards. However, the robots had a consensus algorithm that would allow them to match their heading with their neighbors, which acts to counter the error in their initial headings. There were three conditions: low swarm-to-swarm bandwidth with low swarm-to-user bandwidth, high swarm-to-swarm with low swarm to user bandwidth, and high swarm-to-swarm bandwidth with high swarm-to user bandwidth. The results showed that the medium bandwidth condition, with high swarm-to-swarm bandwidth but low swarm-to-user bandwidth was sufficient to complete the task. The users did adapt their behavior to match the information they had available, issuing more commands closer together in the medium bandwidth condition to drive the swarm into a smaller region and increase intra-swarm connectivity. In the low bandwidth condition, the user did not receive updates about the swarm condition often enough to enable them to maintain a dense swarm, and as a result, had trouble finding targets. The fact that the user interface and the user's intuition about the swarm could drive adoption of different behaviors supports the use of different rendering schemes for the swarm to encourage different user control strategies or conceptualizations about the swarm.

Walker *et al.* explored the condition where a limited amount of information was available, but the time it takes for that information to become available (latency) is another factor in human-swarm interaction [Walker, Nunnally, Lewis, Kolling, Chakraborty, and Sycara, 2012]. In the study, the latency was either

absent, present both between members of the swarm and between the swarm and the user, and present, but the interface to the user predicted future swarm motion and displayed it. The task was similar to Walker *et al.*, in that the swarm was used to search a region for targets in a foraging task. The swarm could be commanded to move to a heading, stop, or flock under a model similar to boids [Reynolds, 1987]. The latency condition was significantly worse, in terms of targets found, than the no-latency condition, but the predictive display countered the latency sufficiently that the predictive and no-latency conditions were not significantly different. Interestingly, latency also drove the appearance of what the authors refer to as "neglect benevolence", where the user not interacting with the system was not only tolerated, but actually helped improve the behavior of the system. Under latency, the user would issue a command, and then wait out the latency period to see how the system responded. The resulting delay allowed operations such as flocking to converge and stabilize, thus increasing the swarm connectivity beyond what was attained in the no-latency condition.

The previously discussed work on attractor management and control under latency and bandwidth restriction is concerned with UI design for swarm robots from the point of view of controllability and influence, which is to say whether or not the user can maintain desirable aspects of the swarm under the given conditions. It is not largely concerned with whether the interface itself, as presented in the experiment, is the best way for the user to attempt to control the swarm. Instead, the interface is taken as a fixed point, and the experimental condition is variation in the quality of the link between the user interface and the swarm.

Kolling *et al.* compared two forms of influence-oriented UI, selection and beacon based control, with a variable number of robots [Kolling, Sycara, Nunnally,

and Lewis, 2013]. Selection control requires that the user select a group and then influence the selected group, or put the group into a specific autonomous behavior mode. This is arguably a form of playbook control, where some robots are selected and then given a play to execute. Beacon control has the user place beacons that attract the robots with a range, which is form of leader or attractor-based control. The paper also refers to these control methods as "intermittent," for selection, and "environmental," for beacons. The intermittent method is called that because the user issues infrequent commands, rather than continuously operating the group. This is in opposition to a persistent control method, where the user might directly operate some of the robots with a joystick or other constant control input, as in some predator/leader systems. The beacons are environmental, because rather than interacting with the robots, the user interacts with the beacons and their locations in the environment. Selection control outperformed beacon control, especially as the number of robots in the swarm increased. The authors indicate that as the swarm size increased, the per-robot precision of beacon control decreased, while selection control remained constant, allowing the user to exert a more precise influence over the swarm. This indicates that the selection and operation control scheme used in this thesis is a better method for controlling large swarms than beacon or attractor based schemes. However, humans were solidly outperformed by fully autonomous benchmarks developed to more optimally perform the foraging task in the experiment, which indicates that human control is more useful in conditions where the humans may have access to information that the robots do not have access to. Otherwise, human control may present a bottleneck, rather than an advantage.

The Interfaceless Interface

Some swarm robots are designed to be used with no control interface at all. By altering the shape of simple stick-slip locomotion toys with laser-cut foam "hats" that change their outer perimeter, and changing the outline of the laser-cut foam forms they interact with, the random motion of the toys can create stable structures [Andreen, Jenning, Napp, and Petersen, 2016]). The development of the shapes relies on human interaction, but specifying the shapes of the hats and interactive components seems amenable to solution with genetic algorithms. Because the toys used in this work were about $4-8, and everything else was composed of laser-cut foam, the platform is extremely inexpensive, but the tasks it can solve are seemingly limited to self-assembly.

The Guardians project studied the deployment of sensor-equipped robots to support firefighters by providing mapping and sensor information that would not normally be available to the firefighter [Gancet et al., 2010]. Rather than having the firefighter control the robots, the robots would deploy with the firefighters and swarm with them, attempting to maintain network connectivity and proximity to the firefighter. The robots provide a stream of information via a heads-up display (HUD) constructed of LEDs on the firefighter's visor, to which the firefighter may or may not choose to react. Instead of having to control the swarm, the firefighter acts as they feel is best, using both their own information and that from the swarm of robots that surrounds them and extends the firefighter's senses [Penders, Alboul, Witkowski, Naghsh, Saez-Pons, Herbrechtsmeier, and El-Habbal, 2011].

Figure 2.2: Typical top-down RTS view from the game Nuclear Dawn, by Interwave Studios. The view is of a courtyard and buildings, showing units (highlighted in red), a picture-in-picture map of the larger area (grey box in lower left corner), and game controls (yellow boxes in lower right corner) [InterWave Studios, 2013].

Video Game UI Design

There is a general resemblence between many of the overhead-view interfaces for swarm robot control and the interfaces used in video games for control of multiple units, especially in the genres of Real Time Strategy (RTS) games, turn-based strategy games, and Multiplayer Online Battle Arena (MOBA) games. Interestingly, most multi-unit video games UIs operate using a selection, rather than beacon, interfaces, according to the classification of interaction methods by Kolling *et al.* The influence of user interaction affordances in games with overhead views and control of multiple units may be a rich area of study for researchers in human-robot interaction, and especially human-swarm interaction.

Real time strategy games generally have the player playing against multiple other players, some or all of which may be controlled by the computer. The

goal of the game is typically to acquire some form of resources, which are used to develop military units or units that accelerate the acquisition of further resources. The military units are then used to attack the other players, with the goal of the game being to be the only remaining player. Turn-based strategy games are similar to RTS games, but instead of all players acting simultaneously and asynchronously, each player's actions are taken on their turn.

MOBA games have a smaller focus and more rapid play than RTS or turn-based strategy games. Typically, there are multiple players, but each player is on one of two sides battling over some set of objectives on the world map. Rather than controlling multiple units, each player controls a single, powerful unit, and may be able to influence weaker, semi-autonomous computer-controlled units. There may be resources as in RTS games, but typically they are far more limited in number, as the emphasis of the gameplay is on action rather than resource extraction.

MOBA games, RTS games, and turn-based strategy games all have a similar user interface. The "world" or area of play is depicted as a map, typically viewed either top-down or an axonometric perspective. The map is where the main action of the game takes place, and displays the relative position of the characters and other elements of the game. Around the map are typically displays and interactive elements such as menus or buttons to interact with the game.

The control of units in the game is done using the mouse to select units and send commands via mouse clicks, while one hand operates keys, usually on the left side of the keyboard, for other functions. The use of the left side of the keyboard is to leave the right hand free to remain on the mouse, but these controls can be customized for left-handed players. How the keyboard is mapped depends on the pace of gameplay. For extremely rapid games, such as Defense of The Ancients (DoTA, a MOBA), there are a small number of key commands that can all be

operated without removing the fingers of the left hand from the keyboard. DoTA in particular uses mouse clicking, combined with the control, shift, and alt keys in various combinations, to perform all of the actions of gameplay [Gamepedia, 2018].

Age of Empires is a RTS. It is less fast-paced than DoTA, but still takes place in realtime. Most of the keys of the keyboard are mapped to various commands, such as locating and selecting different types of units or buildings, or commanding those units and buildings. Commands to buildings typically cause the construction of various units, while consuming resources. For example, pressing 'D' would find a Dock, and then pressing 'A' would command the Dock to start work on a Fishing boat [Age of Empires Wiki, 2018]

Europa Universalis 4 uses the entire top row of letters on the keyboard to switch between different map modes, and every other key to activate some feature of the game, as well as mouse controls for interactions with individual units and locations [Europa Universalis IV Wiki, 2017]. Europa Universalis gameplay does not take place in realtime, and the player may speed or slow the game clock, as well as pausing the game.

Another common feature of MOBA, RTS, and turn-based strategy games is the "fog of war." The entire map is normally hidden from the player until they command their units to enter an area, whereupon the area is revealed. In gameplay, the intent of fog of war is to prevent players from knowing about the opposing player's location without sending sorties. This lack of information is almost identical to that experienced by a teleoperator of a robot exploring an area with a SLAM algorithm, with the map being revealed to the operator as the robot moves into the area.

Intel Drone Swarm Interface

Intel has fielded drone fleets of up to 1,200 Shooting Star UAVs [Intel Corp., 2018a]. While the control system for the drones is proprietary, some information about the user interface can be gleaned from the promotional videos for the controller. The user interface for the creation of the light show appears to be very similar to those used in 3D rendering of computer graphics, including a timeline across the bottom of the screen, a preview in the center, and toolbars along the top and sides for interaction with the show. Visible tabs on the interface include "Curves/Surfaces", "Poly Modeling", "Sculpting", "Rigging", "Animation", "FX", "FX Caching", and "HEALTH". These different tabs are likely different ways to configure the shape of the formation in 3D (Curves, Poly Modeling, and Sculpting), animating the resulting formation (Rigging, Animation, and FX), and determining the state of the swarm as a whole (HEALTH).

A screen that is likely part of the health interface is shown in one video [Intel Corp., 2018b]. The screen has a grid of circles, colored various intensities of green, yellow, and red, as well as one shade of gray. The intensity may convey a different quality about the drone than the color, allowing at least two different dimensions of data per circle. As all the circles are displayed in a grid, an overview of the swarm health can be obtained by glancing at the display and assessing the relative balance of green vs. red.

The user interface can import some form of data about the landscape where the drones will be launched, so the show can be positioned relative to hills or other obstacles in the area. The interface also supports at least some level of touch interaction, as users are shown pressing UI element buttons and rotating 3D renderings in the promotional videos.

The interface for the drone light shows allows single users to fly hundreds to over a thousand drones, but it appears to be designed to create the show in advance, and execute it under central control. Overall, the interface seems to borrow from both interface design for 3D CAD, with animation elements and somewhat natural affordances for rotating and viewing 3D objects, and EID guidelines for presenting both high-level environmental information at the swarm scale, and individual unit health at the single drone scale. The readability of the health monitoring at a glance, by assessing the amount of green and red units is similar to the ambient monitoring available in Swarmish.

Swarm Software Development Methods

Because the conversion of the specification of desired behavior for the swarm into individual programs for the swarm member robots is still an open question, it is necessary to understand the current methods used in the development of programs for swarm robots. Much of swarm robotic development follows the usual model of software development. Starting from a desired functionality, the developer writes a program that they think will provide that functionality. The program is then tested, in simulation or on real robots, and its behavior is observed. The programmer then modifies their program to account for any observed difference between the desired function and the system's behavior. This loop of coding, testing, and coding again is repeated until the system behaves as expected, or the programmer graduates [Cham, 2010].

Because the normal software development model is time-consuming, and outside of the abilities of many people who might want to use swarm robots, it is desirable to automate the development of software controllers for robots. One

approach to the conversion of the command language to programs for the robots is to define a transformation from the command language to executable code. The transformation operation can be codified as a sort of "compiler," or more accurately, a code generator that creates programs for the robots. The possibility of coding for the swarm as an entire system, rather than writing each robot's program independently, has given rise to several programming paradigms and domain-specific languages for robotic swarms.

Another possibility is the composition of preexisting behaviors that each satisfy part of the user's desired behavior. Given some library of primitives, the user could select a sequence of behaviors, or conditions for the execution of behaviors, to build a complete program. Pheromone robotics provides one possible method of controlling this composition dynamically.

Still another approach is to allow the user to specify a desired behavior and evolving controllers to match it. It would be hoped that evolving the controllers would reduce the complexity of the development task to the definition of a suitable means of determining the fitness of the resulting program. Unfortunately, even this reduction does not permit an escape from iterative development.

Amorphous Computing

Amorphous computing (AC), also called spatial computing, is computation using locally-linked and interacting, asynchronous, unreliable computing elements dispersed on a surface or throughout a volume [Abelson, Allen, Coore, Hanson, Homsy, Knight Jr, Nagpal, Rauch, Sussman, and Weiss, 2000]. The motivation for AC is that while it may be possible to produce arbitrary quantities of "smart dust," it is not possible to ensure that it all works well and is precisely located, especially in real-world applications. The goal of AC is to get useful work out of such materials,

despite uncertainty as to their reliability and location. Smart dusts are also the limit case, in terms of scale, for swarm robotics. Indeed, most of the volume of existing swarm robots is motors and batteries, with the computational components taking far less space. Unfortunately, the technology behind smart dust still does not exist, but programming methods for it have some applicability to larger swarm robots [Correll, Dutta, Han, and Pister, 2017].

There are several languages intended to program amorphous computers. "Proto" is a language for a continuous plane spatial computer that maps from behavior of regions at the global level to programs for discrete points at the level of individual devices [Beal and Bachrach, 2006]. Because devices have a size in the real world, and space between them, the devices cannot not have a one-to-one mapping with the space, but instead perform an approximation of the desired behavior. Proto also has considerable appeal as a programming language for swarm control development because of the layering in its structure. If user interface interactions can be interpreted as indications of desired behaviors displayed over spatial regions, then conversion of those behaviors into programs in Proto may be amenable to automation. Proto's layered structure also has a clear relationship to the hierarchical structure of EID, with the programming language serving as a user interface at the highest abstraction level of the interface design, but providing a smooth transition to the lower abstraction levels.

Growing Point Language (GPL) allows the specification of topological patterns in an amorphous computer, and so can also be used to specify the distribution of swarm robots, or behaviors of the swarm robots, in a space [Nagpal and Mamei, 2004]. GPL is inspired by the morphogenic controls present in biological organisms, which use gradients of chemicals called morphogens to dictate the development of cells [Turing, 1952]. The name GPL arises from one of the language's main

abstractions, the growing point. The growing point is the location of activity within the amorphous medium, at which local agents are changing their state. Growing points move through the medium, affecting the state of the computational points they pass, and emitting pheromones into the medium which control the motion of growing points.

Importantly, GPL does not make any prior assumptions on the location of the particles in the system, or robots in the swarm, aside from that they are sufficiently dense in the medium. For swarm robotics, this is an important quality, as precise localization may not be available. Initially, all agents have the same state and program, with a few exceptions that serve as seeds for the growth to begin. If the pattern is not required to be fixed at a particular location, even the seeds could be undetermined initially, and elect themselves via a method such as lateral inhibition. During the execution of the GPL program, each agent chooses its state based on the presence of pheromones, which are morphogens with limited range. Range limitation on morphogens propagating between robots is set using a TTL (Time To Live) counter that propagates with the morphogen, and is decremented with each hop in the communication network. When the TTL hits zero, the morphogen message is no longer propagated. By controlling the production or propagation of morphogens within the amorphous medium, complex patterns can be developed.

Pheromone Approaches

Pheromone robotics is a metaphor used in developing control software for swarm robots. Some social animals, especially insects, use chemical signals called pheromones to communicate with each other. For example, wasps inside their nest react to the scent of wasp venom by travelling to the outer surface of the nest and attacking nearby moving targets [Jeanne, 1981]. Ants leave trails of pheromones for other

ants to follow to food sources. Each individual ant's contribution to the trail can be modulated by the quality of the food source, which allows the reaction of the other ants to the trail to cause an emergent distribution of the foraging work force that favors higher-quality food sources [Sumpter and Beekman, 2003]. Because pheromones are chemicals with spatial locations, it would be possible to combine the use of pheromones with reaction diffusion equations to structure activity within a space or to converge to patterns of activity over time [Turing, 1952]. Assuming even diffusion of the robots in space, the global map of the pheromone concentrations can be represented over the network by the locally-computed concentrations computed by each robot.

In pheromone robotics, the pheromones are usually simulated or "virtual" pheromones, rather than real chemicals that are detected by chemical sensors (for an exception, see [Hayes, Martinoli, and Goodman, 2001]). Each pheromone can have properties such as diffusion and evaporation rates that result in the pheromone spreading in space or gradually disappearing. For example, a robot may emit a pheromone that diffuses into the environment and evaporates quickly, so distance from the robot can be determined by the strength of the pheromone, and approaching or avoiding the robot may be accomplished by moving up or down the gradient of pheromone strength. If the swarm is engaged in a search, each searching robot may emit a "search marker" pheromone that lingers in the area after the robot leaves. Other robots, on entering the area, would detect the pheromone and know that searching this area again would be fruitless. If the object of the search can move, the marker pheromone could diminish as a function of time, so areas that have not been searched for a long time become unmarked and may be searched again. Once the target is found, the robot may stop and emit a "discovery pheromone", which diffuses into the environment, attracts other robots, and causes

them to also emit a discovery pheromone. As a result, once any robot discovers the target, all of the robots quickly converge on its location.

The addition of directional communication for the messages that convey virtual pheromone information allows easy determination of the direction of pheromone gradients [Payton, Daily, Hoff, Howard, and Lee, 2001]. Rather than directly diffusing in the space as a chemical would, hop counts in the network of robots simulate diffusion. Because routes may be of different length, the message with the lowest hop count is assumed to be the truest indication of minimal distance within the network. Rather than modelling the world based on the incoming messages, the content of the pheromone messages and the network behavior as a whole serves as a model of the world, mapped 1:1 onto the real environment. While it is possible to build a set of behavioral primitives out of pheromone signaling and associated behaviors, controlling the swarm to perform a task with these primitives is still done by hand [Payton, Estkowski, and Howard, 2003].

In all of these examples, the sensing of the pheromones is assumed to be local to the robot, at least metaphorically. To actually maintain pheromones in the environment without robots being present to transmit them requires, again, a global representation of the task space that the robots can refer to when needed. However, if some degradation in performance is acceptable, the robots can maintain a shared tuple space that lists information such as pheromone gradients and share it through local connections [Pinciroli, Lee-Brown, and Beltrame, 2016].

Use of pheromones to guide swarm robots for simulated search and patrol tasks has been demonstrated, with the assumption that there is a central controller maintaining the concentration of pheromones on the map, and informing the swarm members [Coppin and Legras, 2012]. In a real implementation, some robots could remain stationary and only act as transponders, computing and transmitting the

local pheromone information for a given area [Hoff, Sagoff, Wood, and Nagpal, 2010]. Another possible approach to enable pheromones in space is to operate the robots in an area that is illuminated by a projector. The projector can then color the area with light that the robots can sense, and modulate the intensity of the light to indicate the intensity of the pheromone [Arvin, Krajník, Turgut, and Yue, 2015a; Diaz-Mercado et al., 2017]. However, even if the robots are limited to only the pheromones they can directly perceive and emit at the present instant, some emergent behaviors are still possible.

It has been demonstrated that a swarm can perform construction tasks using only local sensing and no communication [Wawerla, Sukhatme, and Mataric, 2002; Bowyer, 2000]. However, the addition of communication between systems and memory of the state of the world will improve the efficiency of the system. Pheromone approaches can guide the construction of objects, even if the individual swarm members have no memory and only local perception [Mason, 2003]. The agents engaged in the construction move at random, and take actions governed by their individual perception of environment at present time. The agents can release and react to pheromones in the environment, and so there is an implicit communication via stigmurgy, but no explicit agent-to-agent communication. The set of rules that govern the mapping of sensor precepts to actions must be such that no point in the construction of the building can be mistaken for another, as that could result in loops or skipping parts of the building sequence. The TERMES project created a compiler that translates desired final structures into rules to guide the construction of those structures by cooperating agents [Werfel, Petersen, and Nagpal, 2014].

Vector Fields

Both global vector fields and the global and local blending of vector fields in co-fields can be viewed as subsets of pheromone robotics that use a global spatial representation. The vector field represents the space the robots are operating in as a continuous vector field with the magnitude and direction of the vector at each point in the field controlled by some system of equations. By altering the specification of the field, the user can change the actions of the robots within that field. One approach to a control UI for a remotely-located swarm is a multi-touch interface for specifying a vector field [Kato et al., 2009]. Because the user interface design focuses on the vector field rather than individual robots, the same control interface can scale to an arbitrarily large collection of robots. Vector field paths can have loops, which do not exist in waypoint-based paths. Waypoint paths have explicit ends, unless an additional command is added to join beginning and ending points.

Vector field paths have substantial limitations. Because the vectors are bound to a 2-D plane, the paths they create cannot cross each other. Instead, they flow together. A 2-D vector field is also not a useful metaphor for controlling UAVs. The vector field could be possibly extended into three dimensions, to control UAVs as well as ground vehicles, but there would have to be some form of discontinuity in the field to prevent assignment of UAVs to ground vehicle paths and vice versa. The vector field can be viewed as an abstraction of pheromone control, or even implemented in terms of the presence or absence of virtual pheromones, but it has some limitations that pure pheromone control does not have. For instance, pheromones permit the presence of multiple pheromones at one point, with multiple meanings, but the vector field has one value for each point. Attempting to solve this problem by proposing multiple fields raises the question of how to combine the

influences of each field, and if a universal combination function applies at all points in the field, it raises the question of why they were not combined in a single field under that combination equation.

The vector field is also not very intuitive to users. Kato *et al.* indicates that in order to use the vector field well, the users had to anticipate and project the future motions of the robots. Interface changes, such as showing particles on the vector field, could improve usability, but these approaches would have the same scaling problems that the robot representation does. When the view is zoomed out very far, individual whirls and eddies in the field may not be visible to the user.

Using a vector field for the interface does not directly map to programs on the robots. Instead, the vector field describes how a robot located at a specific point should move. The central computer maintains the vector field representation and commands the individual robots. Because the vector field describes motions, rather than tasks, conversion from a task-based user command to a vector field, the task would have to be converted into a series of changes to the field at specific locations. Since the robots may not have accurate localization within the task space, it may not be possible to guide the robots by relating their position to a global vector field.

The use of co-fields may provide a way to move the vector field representation from the central computer to the swarm, or allow the swarm to act for some time without constant updates from a central controller [Mamei, Zambonelli, and Leonardi, 2003]. Co-fields distribute the data within a space, which may be physical or may be abstract. Agents react to gradients in field, and spread their own fields over local communication networks. The overall vector space created by the user (the UI vector space) could be propagated to the robots periodically, and combined with their own internal vector fields to generate movement based on

both the user's desires and the local rules operating on each robot. As with general vector fields, knowing which areas of the UI vector space are relevant to each robot may require global localization, and so only be available for swarms operating in conditions that permit global localization.

Compositional Approaches

Rather than developing a novel control program for each robot automatically, it may be possible to compose programs from behavioral primitives, such that some combination of the primitives results in the emergence of the desired behavior. A compositional approach to program generation requires the definition of primitives out of which programs can be composed, and some degree of assurance that these primitives can cover the space of possible tasks required from the robot. One possible list of primitives is disperse (no other nodes within distance d), general disperse (no more than n nodes within distance d), clump/cluster, attract to location, swarm in a direction, and scan area [Evans, 2000]. Another proposed catalog of behaviors for swarm control bases the simple behaviors on pheromones or chemical sensing in single cells [Nagpal, 2004]. The proposed behaviors are the use of gradient sensing for position and direction information, local inhibition and competition, lateral inhibition for spatial information, local monitoring, quorum sensing for timing and counting, checkpoint and consensus sensing, and random exploration. While these behaviors are themselves expressed in terms of pheromones, the composition of the primitives into complete programs is not dictated by a pheromone-based system. Furthermore, compositional approaches have been proposed in control-theoretic terms as well as pheromone-based terms, so the process of composition of primitives can be viewed as a metastrategy for the creation of programs, rather than a process specific to pheromone robotics [Belta, Bicchi, Egerstedt, Frazzoli, Klavins, and

Pappas, 2007].

Another compositional method for programming robots proposes that the behaviors can be separated into classes, such as motion, orientation, and so forth [McLurkin, 2004]. Among these behaviors are "primitives" such as several forms of clustering, which other, later works have treated as an emergent behavior itself, arising from more atomic primitives. The variable granularity of the primitives available to compose swarm control programs seems to point to a hierarchy of control elements, with perhaps single motor operations at the bottom, and an increasing composition of elements to create more and more complex behaviors. Swarm control programs would then call multiple primitive behaviors, providing them with parameters such as degrees of bearing and centimeters of proximity. The behaviors ideally run concurrently, and some of them respond to sensor inputs. The output of behaviors is whether they are running, translational and rotational velocity for the robot, and LED configuration. Because multiple behaviors might specify differing outputs, subsumption and summation are used to arbitrate between behaviors of differing priorities.

These emergent approaches do not have the robots perform all of their available actions all of the time. Instead, it is assumed that the behavior of each robot is controlled by its reaction to the environment around it, and possibly to signals from other robots, so that actions are only performed when they are required. As a result, user programs compiled from a higher-level representation could be a table consisting of possible values for the sensors, and the actions to undertake when those values are met. Guarded Command Programming with Rates (GCPR) provides a formal framework for the analysis of this type of compositional program [Napp and Klavins, 2011]. Robots are assumed to only have local sensing. The guards of GCPR are conditions on the environment. When a condition is met, the

robot performs actions at a given rate. In the concurrent case, this is modeled as each action happening one at a time, but in random order. On a real swarm, the actions would take place in parallel, but the concurrent model is more amenable to analysis. To determine if a set of actions will be successful, it is required to ensure that for all orderings of all actions, the final state space of the swarm is the desired final state. Correct programs are those that reach the target state with probability 1.0, even when composed with bounded failures. Once the target state is reached, the program is assumed to halt, so while the final state may be reached very slowly, once it is reached, it is not left. In the GCPR models, the time to execution of an action is stochastic, but in the real-world case of noisy or imperfect sensors, the variable time to execution of a guarded behavior would be caused by the imperfection of the robot's ability to detect that the guard was satisfied.

Evolutionary Composition

Determining how the behaviors should be composed for an individual swarm robot's controller is difficult. Unfortunately, much of the process of composing programs for swarm robots consists of iterating between composing sets of primitives and observing the behavior of the system in an ad hoc process, just as with the creation of control programs by coding [Palmer et al., 2005b]. Rather than removing iterative software design from the process, it has simply been moved up one layer of abstraction, from writing code for the robots to composing that code from behavior primitives.

One possibility is to permit composition to behave like the programming environment Tierra [Ray, 1991]. In Tierra, there is no such thing as an invalid program. All sequences of the existent symbols are regarded as executable programs, although some are more useful than others. The possibility that programs can be

ranked by some function creates the possibility that genetic algorithms (GA) can direct the automated composition of behavioral primitives into programs [Palmer, Kirschenbaum, and Seiter, 2005a]. A GA expresses the robot program as a genome, which is translated into the actual program and run on the robot. The result of running each program is assessed using a fitness function and then the genomes for the best programs are combined to produce a new generation of genomes. This cycle of combining and assessing genomes continues until a certain level of quality is reached, as judged by the fitness function.

Unfortunately, given the time required to iterate over multiple generations of controllers, genetic approaches are unlikely to be fast enough for interactive control of robots by a human user. However, it is still useful to examine the possibility of developing a fitness function as a way of investigating methods for automatically assessing the behavior of a swarm. Without some way of determining the "goodness" of a swarm's behavior, it is difficult to say that one algorithm or design paradigm is better or worse than another.

In order to determine the quality of the behavior of the swarm, its behavior must be measured. Harriott *et al.* propose that metrics for measuring the interaction of humans and swarms differs significantly from the interaction of humans and individual robots, and can be broken down into nine classes, summarized below [Harriott, Seiffert, Hayes, and Adams, 2014].

1. Human attributes - Interaction, trust, intervention frequency
2. Task performance - Ability to accomplish task, speed, accuracy, cost
3. Timing - Command diffusion lag, behavior convergence
4. Status - Battery life, number of functioning members, stragglers
5. Leadership - Interaction between special members of the swarm and others
6. Decisions - Action selection, likelihood that the correct action is chosen

7. Communication - Speed, range, network efficiency
8. Micromovements - Relative motion of individual swarm elements
9. Macromovement - Overall swarm motion, flocking, elongation, shape

Task performance is an especially interesting metric, but it is difficult to automatically extract from the observed behavior of the system an overall understanding of the progress it is making on the task, and so a value for the output of the fitness function. Worse, without a time bound on solving a problem or a way to calculate progress, it is impossible to tell if a program has failed, or has merely not yet succeeded. For example, assume a program's intended purpose is to gather all of the units of a resource at a goal. If the program merely moves the units stochastically, sometimes they will enter the goal, creating an appearance of progress. However, it may be vanishingly unlikely that all the units will randomly happen to be in the goal at once. Counting the units moved to the goal, then, cannot distinguish between a program that cannot find the goal, and so will never put any units in it, and a perfect resource-gathering program that just has not moved any units to the goal yet.

Ideally, it would be possible to recognize and evaluate performance on sub-problems. It has been proposed that the interactions and emergent behavior of the system are observable, while the reactions of the agents in the system are programmable, and so by observing the interactions and emergent behavior, the developer can receive feedback on how the system is progressing [Palmer et al., 2005b]. However, all of the proposed observation and hierarchy of reaction and emergence is intended as a design process, not an automation process. In other words, while the observation and hierarchical structure may guide the development of an emergent system, the system is still developed by programmers writing code and then running it on the robots.

In the limit, the swarm could be treated as a gas, and for tasks such as diffusion over an area, the performance of the swarm would be compared to the behavior of an ideal gas [Jantz, Doty, Bagnell, and Zapata, 1997]. The addition of sensors and computation would then allow the robots to outperform a gas at tasks, and so achieve higher scores on a task-oriented metric than a gas could attain. Unfortunately, this metric is constrained to tasks that an ideal gas could perform, which are largely restricted to diffusion and alteration of density in response to temperature.

Controllers have been evolved to allow robots to move into formation from random starting positions [Quinn, Smith, Mayley, and Husbands, 2003]. These controllers use local interactions and minimal sensing to achieve their goals. One point the authors make, which is not frequently mentioned in other work, is that while flocking or shoaling behavior is a relatively simple behavior to have emerge from robots who can detect the distance, position, and velocity of the other nearby robots, implementing that perception on real robots is quite difficult. Because a specific behavior was desired, the fitness function used to evolve it was specified in terms of metrics related to the behavior. Task-specific fitness functions are also found in later work on evolution of swarm robot behavior, which seems to indicate that evolution of behaviors in swarm robots may only be a time-complexity trade-off.

Interestingly, some of the work in evolvable controllers leads to inter-robot communication as one of the emergent properties of the evolved controller [Quinn, 2001b]. In order to move as a formation, one of the robots must be the leader, but there is nothing in the fitness function or any of the other code that designates roles for the robots. Instead, the selection of the leader arises from the evolutionary development of the controllers, and is present in the controller as a response to a

particular series of stimuli. Genomes that did not encode such a symmetry-breaking reaction never developed a leader-follower distinction, and so failed to move in formation, and so received low fitness scores. For the follow-the-leader task, genetic variation among the robots increased fitness more readily than having all robots share the same genome [Quinn, 2001a]. The condition where all robots shared the same genes was called "clonal", while each robot having its own genome was "aclonal". Oddly, while one would expect that the aclonal condition would result in a specialization, with each robot developing a genome that performed either the leader or follower role well, the aclonal condition developed robots that could perform both roles. It was hypothesized that while the clonal condition had to evolve roles and an allocation mechanism simultaneously, the aclonal condition could specialize the roles during early evolution, and then develop an arbitration mechanism to select roles.

Genetic algorithims have also been used to develop aggregation behavior in swarm robots [Bahgeçi and Sahin, 2005; Dorigo, Trianni, Şahin, Groß, Labella, Baldassarre, Nolfi, Deneubourg, Mondada, Floreano, et al., 2004]. Aggregation was chosen because it is a preliminary behavior primitive, which the swarm might engage in prior to doing some other task, such as moving an object or attacking together. The resulting controllers only control aggregation behavior, so each behavioral primitive would require its own evolutionary development. Solutions discovered by genetic algorithms are also prone to overfitting. The swarms described in Dorigo *et al.* decreased in performance when the number of robots involved in the swarm was changed from the values used to evolve the solutions, and when a more accurate physical model was used in the simulations.

An interesting recent application of genetic algorithms (as well as simple exhaustive parameter searches) is the development of swarm robots that operate

without computation. These robots are generally theoretical, rather than actually implemented in hardware. They typically have one sensor, which is binary and detects the presence or absence of an obstacle, and a program consisting of an if statement that branches according to the output of the sensor. The output of the program is typically the speed of two wheels in a differential drive arrangement. Despite the minimalism of the model, by tuning the output parameters of the robot, they can be configured to form stable circles, and with a few more sensor states, to aggregate, shepherd "sheep" robots, forage, gather objects, follow walls, and disperse [Gauci, Chen, Li, Dodd, and Groß, 2014; Johnson and Brown, 2016; Özdemir, Gauci, and Gross, 2017; Brown, Turner, Hennigh, and Loscalzo, 2018; St-Onge, Pinciroli, and Beltrame, 2018].

While these abilities of computationless robots actually do cover the desired tasks from the user experiment in Chapter 4, this approach has some problems with transition to deployment on real robots, rather than point robots in simulation. The simulations used in computationless swarming frequently assume a perfect sensor of infinite range, which is unlikely to exist on a real robot. However, at least one study was performed with sensor noise that could change the sensor reading, causing it to give an incorrect value with probability p, and found that the controller outperformed doing nothing, so long as p was less than 0.5 [Gauci et al., 2014]. While extensions which admit a sensor that can detect obstacles, and so avoid them, are no doubt possible, there are more troubling results that indicate that the general design problem for an computation-free swarm in an arbitrary environment is NP-complete [Wareham and Vardy, 2018] Further, the use of evolutionary programs is still prone to long run times for the evolutionary algorithm itself, and to exploiting errors in the statement of the fitness function. As a consequence, it is ill-suited to user interaction with users who are not experienced

in genetic algorithms.

The common hope of users of genetic algorithms is that they can reduce the complexity of directly specifying the task to the (hopefully lower) complexity of describing the results in the fitness function. Rather than describing how to solve a problem, one simply describes what the solution would look like. Unfortunately, the reduction in hands-on time spent programming frequently comes at the expense of time spent waiting for the system to converge, or determining why it converged on a problematic solution. One attempt to evolve aggregation controllers had a fitness function that allowed the evolved motion strategy to acquire a high fitness by spinning in place. The ad hoc iterative process of creating emergent behaviors is replaced by an ad hoc iterative process of creating fitness functions. As a result, developing novel behavior in the field by converting user specifications of the behavior of the swarm into a fitness function for a genetic algorithm is unlikely to yield results in a timely manner. However, individually evolved primitives could be saved in a library of primitives for use by a higher-level compositional approach. Such a library could take advantage of the possible overfitting of GA evolution by storing primitives intentionally overfitted to specific situations and robots, and using the best matches. For example, a controller that aggregates small-scale UAVs outdoors is likely quite different from one that aggregates medium-scale wheeled robots indoors, even though they are both aggregation controllers.

Domain-Specific Languages for Swarms

Proto and other programming languages for amorphous computers provide abstractions for computation performed on homogeneous spatially-distributed computing nodes. Other versions of tuple-space based amorphous computation include motion of the agents, but do not explicitly support heterogeneity [Viroli, Pianini, and

Beal, 2012]. Several programming languages have been developed that are designed specifically for programming swarm robots.

The Voltron programming language provides what its authors describe as "team-level programming" for autonomous drones [Mottola, Moretta, Whitehouse, and Ghezzi, 2014]. This level of programming is distinct from drone-level programming, where specific instructions are provided to each drone, and swarm programming, where each drone has the same instructions and operates without communication with other drones. Voltron programs consist of sensing tasks that are subject to space and time constraints, so the language does not permit the user to specify direct interactions between drones. This leaves out activities beyond sensing that may be useful for swarm robots, such as patrolling an area or collaboratively moving an object. Additionally, Voltron is based on the assumptions that drones have global localization, synchronous clocks, and reliable inter-unit communication.

Karma provides a programming framework for micro-aerial vehicles with minimal localization and no communication in the field [Dantu, Kate, Waterman, Bailis, and Welsh, 2011]. The framework allows the composition of behaviors described at the level of individual robots. Rather than each robot performing a set of behaviors in response to input from its sensors, each robot is tasked by a central "hive" to perform a single behavior, such as performing a sensor sweep of an area. The hive collects data from robots as they return to the hive, and updates a central data store, which includes both the sensor information from the individual robots and spatial information about the sensor information. The hive then assigns activities to robots based on rules that use the central data store to determine which activities should be performed. As a result, while the individual robots are autonomous and not in communication with the hive while operating, the hive maintains a central data store that is used to guide the future behaviors of

the swarm. This model does permit a form of interaction between swarm members, in that information from one swarm member can inform the behavior of another member, but it does not permit dynamic collaboration between swarm members while they are operating away from the hive.

Meld is a programming language for robot ensembles, which are composed of individual modular robots [Ashley-Rollman, Goldstein, Lee, Mowry, and Pillai, 2007]. However, many of the problems facing an ensemble of modules are the same as those facing a swarm, such as determining the overall goal, moving to proper positions, and detecting when the goal has been acheived. Meld is written in terms of facts and rules. Facts describe things such as adjacency between robots and location of robots. Rules are applied to facts, resulting in the generation of new facts. By including rules that generate facts that describe motor actions, the application of rules to the known state of the system can create a "to-do list" of actions for individual members of the system to perform. Rules are said to "prove" facts, and when no further facts can be proven, the system has arrived in a final state. Since facts that alter the state of the world can make previous facts false, the set of facts available must be periodically purged of facts that no longer hold. The authors of Meld point out that logic programming, of which Meld is an example, is poor at representing state beyond what can be computed as a consequence of the base facts. However, it is also claimed that the use of aggregates, which compute results based on the provable facts, can be used to store state about the system, avoiding this restriction.

Buzz is a programming language and a virtual machine (VM) to run it on that is designed for programming swarm robots [Pinciroli, Lee-Brown, and Beltrame, 2015]. Each robot is assumed to be running the same bytecode on the Buzz VM (BVM). Buzz is also based on the assumption that robots exchange information in

a situated manner, with any robot that receives a communication also being able to estimate the relative location of the source of that communication. In order to support programming for swarms, Buzz treats the swarm and virtual stigmergy as first-class objects in the language. Swarms in Buzz provide an abstraction for a group of robots that allows the programmer to have every robot in the group execute a function, as well as dynamically create and disband groups. Virtual stigmergy provides a global, distributed data store for the swarm. The implementation of the virtual stigmergy structure allows the robots to maintain relatively up-to-date versions of the values stored in the data store, and to refresh them after recovering from failures of network connectivity. Buzz also provides a convenient abstraction of the neighbors of the robot executing the program, which usually consists of all robots within communication range. Using the neighbors abstraction, the robot can, for instance, query all its neighbors about the value of a sensor precept at their location, in order to build a local map of the intensity of the sensed quantity.

The design of a programming language can be considered a user interface, after a fashion, as it provides an abstraction of an underlying system that a user can use to control the system in a manner that is ideally easier than less abstract interfaces. However, both domain specific programming languages and amorphous computing languages share the same problem: for end users, learning a programming language in order to command a swarm is an unacceptably high barrier to adoption.

Program Generation from Formal Specification

Controllers for modular robots that allow them to form shapes under stochastic mixing have been automatically generated from descriptions of the desired shape [Klavins, 2002]. These controllers are focused on structures and formations that can be expressed as a tree. They do not relate the structure to any other point in

space, but are purely in terms of connectivity between modules.

Other work with modular robots has used sensing to characterize the environment and decide on the reconfiguration of the robot that will best allow it to complete the assigned task [Daudelin, Jing, Tosun, Yim, Kress-Gazit, and Campbell, 2017; Jing, Tosun, Yim, and Kress-Gazit, 2016]. The high-level planner uses linear temporal logic in a tool called LTLMoP (Linear Temporal Logic MissiOn Planning) to synthesize a controller for the robot task, and from the behaviors required from the controller, a sequence of transformations of the robot into forms that provide those behaviors.

Generation of programs from e.g. linear temporal logic (LTL) descriptions of the task and environment are promising because they can generate provably correct controllers. Unfortunately, these approaches to program generation are prone to extreme conservatism in the face of an unknown environment. Assume that a robot is placed in a simply-connected maze, with instructions to traverse the maze. Selecting one wall, either the left or the right, and following it, will result in the robot finding the exit, or returning to the entrance if there is none. However, the *a priori* generation of a motion plan to reach the exit from the entrance is impossible, as the actual path taken is unknown. LTL and similar logics can only work on what is known about their environment, and so cannot operate in unknown environments.

A potential method to extend LTL or other logics to an expanding environment is to develop what is known about the environment as the robot explores, and rebuild the controller as new facts are added. When the robot has explored enough, it will become possible to build a controller that achieves the desired result, or prove that it is not possible because the entire environment has been explored without gaining enough information to build the controller. While

effective in theory, this approach can run aground on combinatorial problems, particularly in dynamic environments, as the facts may vary often, and require reworking of the controller.

One approach to correct-by-construction controller synthesis for dynamic environments combines an offline LTL-based planner with an online motion planner for collision avoidance [Alonso-Mora, DeCastro, Raman, Rus, and Kress-Gazit, 2018]. Rather than replanning as the environment changes, the system generates alternative plans, or explains to the user why no alternative plan satisfies the requirements. Unfortunately, this controller synthesizer still relies on a known map of the entire environment (but not of the obstacles in it). For real-world applications in disaster recovery, the map may not be available, or may be undergoing constant revision by the disaster. The synthesizer also creates revisions to the assumptions about the environment, and the number of these revisions grows combinatorially. Each revision is an assumption about the environment that must hold for the controller to be correct, such as "There will never be a deadlock if Robot 1 is not in the hallway when Robot 2 enters the hallway". The revision can be viewed as a constraint on the motion of the robots, preventing Robot 1 from entering the hallway, or Robot 2 from entering the hallway if Robot 1 is in it. With one robot in a fairly simple map, the paper indicates that 16 revisions are needed. Adding a second robot increases this to 1306 revisions.

Combinatorial explosions are a frequent problem in synthesis of provably correct controllers. Mehta *et al.* presented work that synthesizes not just the controller, but the entire robot [Mehta, DelPreto, Wong, Hamill, Kress-Gazit, and Rus, 2018]. In the controller, the sensors are represented by binary propositions of the controller logic, one for each possible sensed object. For small cases, this is sufficient, but the number of such propositions is $2^{|objects|}$, and so becomes unwieldy

for more complex specifications and robots. However, it does pose an interesting argument for heterogeneity of robots. Rather than changing how a swarm of robots with a fixed morphology is used to perform a task, robots can be generated as needed for tasks and added to the swarm. One would expect that once a diverse enough pool of robots exists, the work of Mehta *et al.* could be extended to select combinations of existing robots to perform a task, rather than creating new ones.

It has been suggested that LTL is not the best formalism for the description of desired behavior for robot controllers [Belta et al., 2007]. Belta *et al.* also raises some other problems with symbolic control and planning as a paradigm for robot control. Among others, they point out that the invariance of the dynamics of the robots to certain transformations, such as rotation around certain axes, permits symbolic control. However, as pointed out by Rodney Brooks in "Artificial life and Real Robots," environment conditions such as carpet nap can change dynamics of the robots in ways that break the symmetry of the dynamics about some axes [Brooks, 1992]. In addition, the symbolic methods still have problems with incomplete knowledge, and dynamic worlds (which imply incomplete knowledge, as things may change without being observed). While valuable for the strength of the assertions they can make about the generated controllers, symbolic and formal methods still require some level of reactive or heuristic buffering from the real world, to present them with the atomic actions and perfect sensors on which their assertions rely.

Another potential approach to the generation of robot programs for swarms is the use of Supervisory Control Theory (SCT) [Lopes, Trenkwalder, Leal, Dodd, and Groß, 2016]. SCT provides a method to formally synthesize controllers called "supervisors". Both the possible state transitions of the robot and the control specification are represented as state transition diagrams, and realized as generators.

The generators are then composed into a synchronous automaton, which is the supervisor. Changes in state are caused by events. Uncontrollable events are produced by feedback, such as from sensors or timers, and controllable events are produced by the supervisor to control the robot. The resulting controllers can be demonstrated to not deadlock, and with probabalistic extension into pSCT, also not to livelock, while safety can be formally determined by ensuring that there exists no path in the supervisor leading to a forbidden state [Lopes, Trenkwalder, Leal, Dodd, and Groß, 2017].

However, SCT alone is not sufficient to automate the generation of robot programs. The expression of the abilities of the robot and the desired behavior, as state transitions in some form, is left as an exercise for the swarm developer. SCT supplies a way of ensuring that the resulting program does what the specification requires. As noted in [Lopes et al., 2017], a specification that is logically incorrect, such as including a possiblity to livelock, will generate a program that can be formally demonstrated to match that specification, and so includes the specification's errors.

EvoStick and AutoMoDe-Vanilla both provide automatic design of swarm robot software using a test-and-refine method [Francesca, Brambilla, Brutschy, Garattoni, Miletitch, Podevijn, Reina, Soleymani, Salvaro, Pinciroli, et al., 2014a]. AutoMoDe creates controllers by combining parameterized modules in a PFSM. AutoMoDe-Vanilla outperformed EvoStick, which is a typical neural network control evolution algorithm. It was also able to outperform unconstrained human programmers, but not humans using the same set of modules available to AutoMoDe. An enhanced "flavor" of AutoMoDe, AutoMoDe-Chocolate, outperformed even the constrained humans [Francesca, Brambilla, Brutschy, Garattoni, Miletitch, Podevijn, Reina, Soleymani, Salvaro, Pinciroli, et al., 2015]. The goal of using pre-defined modules is actually to simplify the design space, under the theory that approaches

such as evolutionary design of neural network controllers have, essentially, sufficient representational power to overfit to their test environment [Birattari, Delhaisse, Francesca, and Kerdoncuff, 2016].

Group Perception of Humans

Because it is desirable to have the user operations remain consistent despite variation in the size of the swarm, and because such a consistency implies that the user treats the swarm as a whole, discovery of the factors that result in perception of groups as a single entity rather than a collection of individuals is of interest. One potential guide to when users will stop treating robots as individuals and begin treating them as a group is the numerical point that humans come to regard a collection of other humans as a collective, instead of individuals. Unfortunately, that point is quite complex to determine, because humans have vastly more complex social interaction with each other than they do with robots, and elements of that social interaction color how humans think of each other.

In perceptual psychology, gestalt perception provides a number of heuristics or principles that can influence whether a set of objects is perceived as a group.

Proximity principle Objects near each other are perceived as a group.

Common fate principle Objects that move together are perceived as a group.

Similarity principle Objects that are visually similar are grouped together.

There are various other principles, and the order in which they are assessed or their relative strengths are unclear, so it is difficult to determine how a complex scene will be parsed. However, these general guidelines do appear to

influence how people perceive groups of other people. For example, all of the people in a line or a parade move together, and so are perceived as a group, separate from people around them moving in other directions. It has been suggested that these principles may enhance the intelligibility of the behavior of robotic swarms [Nagavalli, 2018].

Humans also make some distinction between aggregates and groups [Wilder, 1978]. Groups have a boundary, such as a common belief or goal, and sharing that element makes someone a member of the group or not. Aggregates are random sets of people, who may have nothing in common. The people inside a church at a given time are likely a group, the people on the sidewalk outside are more likely an aggregate. This factor, combined with the gestalt common fate principle, may be useful for encouraging users to think of a swarm as a group. Since it can be made explicit to the user that the swarm shares common goals, and the swarm can move together, the user would come to regard it as a group, rather than a simple aggregate.

Whether a set of people is perceived as a group is referred to as "entitativity." Groups with high entitativity are expected to have a higher degree of unity and consistency within the group. The smaller a group is, the more similar its members are perceived to be, which seems to indicate that smaller groups would have higher entitativity [Stewart, 2003].

Entitativity is not a binary classification, with groups being either entities or not. Rather, it is a spectrum, with groups such as families or organizations at the high entitativity end, and groups such as people in line at the bank on the low entitativity end. By surveys, some clusters of groups have been elucidated, such as intimacy groups (family and friends), task groups (co-workers, juries), social groups (gender, race, class), and loose associations (parade crowds, fans of a music

genre) [Lickel, Hamilton, and Sherman, 2001]. In a situation familiar to computer scientists, the analysis of perception of group entitativity is prone to the curse of dimensionality: what counts as a group is highly variable along multiple axes [Lickel, Hamilton, Wieczorkowska, Lewis, Sherman, and Uhles, 2000]. Among the factors influencing it are differences in the perceiver (level of need to perceive a group, individualism or collectivism bias), contextual factors (group membership or opposition), and properties of the group itself (visual or behavioral similarity). Minorities become higher in entitativity due possibly to the perception of them as figures distinct from a "ground," as in the gestalt figure/ground principle, or because stereotyping increases the perception of similarity within the group. The perception of similarity within the group may be in turn be related to the gestalt perception of similarity, and so visually-similar minorities would be expected to have a higher entitativity than visually dis-similar groups of people.

Research in the influence of group size on effectiveness of persuasion reveals that the gestalt principle of similarity works on the level of information as well as visual similarity [Wilder, 1977]. When a person is confronted by a group attempting to influence them, increase in the size of the opposition beyond about 3-4 people has little effect. Rather than treating each person in the opposing group as contributing a bit of information that might sway the subject, and so a large group would contribute many bits, the entire group is lumped together and contributes one bit of information. The first person from the group who expresses their opposition may be viewed as a leader, but the rest can then be dismissed as followers, expressing an opinion no different from the leader's opinion.

Unfortunately, the study of the entitativity of groups makes it clear that the factors that humans consider when deciding how strongly an association of other humans count as a group extend well beyond how many of people are in the group.

Human perception of groups has less to do with exact count of people, and far more to do with common cause, visible similarity, tasks, interpersonal relations, and so forth. One potential upper bound that is simply numeric is Dunbar's Number, a constraint on the understanding of a social network placed by neocortex size [Dunbar, 1992]. For humans, Dunbar's number is estimated to be around 150 individuals, and beyond that, people would be expected to begin chunking the people in their social network into hierarchies or groups in order to keep track of a reduced number of entities (e.g. one group called "co-workers" instead of every individual on the team).

Human-Robot Teaming

One area that may have interesting insights into control interfaces for heterogenous swarms is work on human-robot teams. Typically, human-robot teams involve robots working along side humans in a team, as peers or subordinates. As a result, the interface design is primarily for control of the robots, and the return of data to the humans, rather than in control of the team as a whole. One attempt to unify the command and control of a human-robot team [Lopez, Kuczynski, and Yanco, 2017] did so by expanding previous work, which had only commanded robots [Micire et al., 2009]. The resulting interface allowed the end user to command robots and humans using the same command modalities, rather than specializing the command set based on the type of unit being commanded. In user tests, commanders of human teams, robot teams, and mixed teams had similar performance, and used similar control strategies. The use of similar control strategies is surprising, because this includes direct, joystick-like control of human team members, which the researchers originally thought would be an unusual way to instruct a person to move. The users

also mentioned that having an interface that reflects the generally superior mobility and perception of humans would be an improvement, so while the unspecialized interface did not appear to hinder users, it may have not taken full advantage of the team ability. By similar arguments, it could be that an interface for a heterogeneous swarm should support special operations for robots that can support them, rather than attempting to treat all robots the same in the interest of interface consistency.

The multi-robot interface is also distinct from a swarm interface, although there is some overlap. A purely swarm-oriented interface would issue commands to groups of robots, and not have the ability to command single robots, under the assumption that tasks assigned through the interface were not possible for a single robot to complete. A pure multi-robot interface is designed to command multiple robots as individuals, with their own tasks. The interface described by Lopez, Kuczynski, and Yanco is somewhat in the overlap area, as it allows group selection and task assignment as well as individual command, both as "groups" of one, using similar command interactions to the group commands, and direct, joystick-like interaction. The interface described by Humphrey *et al.* is an interface at the multi-robot end of the spectrum, as it does not appear to provide any faculty for assigning a task to multiple robots at once. It does, however, encourage the user to be aware of the functioning of all of the robots.

In this work, the description of the user interface does not position the robotic swarm as a teammate. Robots can be team members with humans, but they can also be treated as remotely-operated tools. Teleoperation is a case of treating the robot as a remotely-operated tool. Rather than taking any initiative itself, the robot does as commanded, acting as the remote user's eyes and sometimes hands in the environment. Indeed, a teleoperated system should be invisible to the user, with the UI acting as the user's interface to the remote environment, rather than

to the remote robot [Larochelle, Kruijff, Smets, Mioch, and Groenewegen, 2011]. Examination of the work on robotic teammates lists some points about teammates that the swarm described in this work does not share.

Teammates make decisions during task execution, while considering the impact of those decisions on their teammates [Shah, Wiken, Williams, and Breazeal, 2011]. The system described in this paper does not have any mechanism for the system to consider how its decisions will affect the human user. Some of the design decisions were made by with the impact on the human user in mind, but these decisions were made by the author, not the system.

Teammates communicate progress and provide support to each other. This communication can be explicit, or implicit, by doing things like placing completed components of the task where the other teammates can see them. The system described in this work can only be viewed as communicating implicitly, by having the swarm behavior attempt to match the user expectation.

Being a team or not can be viewed as a spectrum, with robots taking the place of equal peers at one end and remotely-operated tools at the other end. Safeguards, such as safe-mode teleoperation, where the robot can act to prevent collisions, are on the spectrum, but more towards the remote tool end. Safeguards do not permit the robot to assume responsibility for a role or task, which is a component of team membership [Bruemmer, Few, Boring, Marble, Walton, and Nielsen, 2005]. Taking initiative combined with assuming responsibility is the basis of leadership. In a team of equal peers, initiative and responsibility, and so leadership, can be distributed rather than centralized. The system in this paper has a central authority, the human user, who commands, but is not commanded by the system. Humans find it easier to delegate responsibility to more human-like robots than to non-human-like robots [Hinds, Roberts, and Jones, 2004]. This is in keeping

with the spectrum of remote tools versus equal peers. A non-humanoid robot is easier to perceive as a tool, and a tool is not blamed if its user makes an error. The difference in perception of humanoid robots can also serve as a design guideline, indicating that non-human robots are to be preferred if the robots are likely to be unreliable, as the robot's form will encourage people to take more responsibility themselves instead of delegating it to the robot.

On the spectrum between teammates and remotely operated tools, the swarm described in this work is solidly at the tool end of the spectrum. The user controls the swarm in a task-based manner, assigning it a job they want done and then allowing the system to operate. The robots used are non-humanoid, both in their depiction on the user interface and their physical construction.

Chapter 3

Swarm Robot System Development

Swarm robot platforms tend to fall into one of two groups, from a hardware perspective. The first group uses microcontrollers and very limited onboard computation, but is small and relatively cheap. This group includes Alice, Jasmine, AmIR, and the other tabletop systems. Due to their limited computation, these systems do not generally support complex algorithms such as vision processing. The second group use more powerful computers, but at a significant cost in weight, power consumption, and financial outlay.

The robot described in this work is intended to occupy a theoretical "sweet spot" at the high end of the tabletop swarms or the low end of the room-scale swarms, depending on how large of a mobility platform is used. As a result, if it is configured for tabletop operation, the system can be used with a minimum of available space. If, on the other hand, it is configured for room-scale operations, the system can be tested in natural or naturalistic human environments.

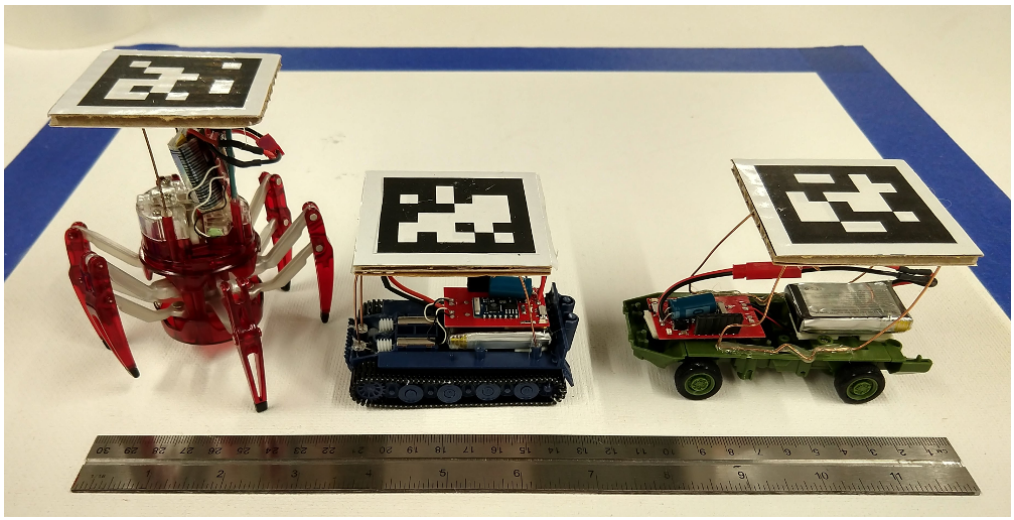


Figure 3.1: Toys with controller boards and batteries mounted. The spider has a two-motor holonomic drive, the tank uses differential drive, and the car is Ackerman drive.

Hardware Platform

The robot swarm developed for this work consists of a hardware module for controlling two motors of a toy, such as a small RC car, for mobility. The reasons for choosing this hardware design are explained in more detail below, but the overall intent is to have an inexpensive platform available for swarm research, without having to rely on any particular group of swarm robotics researchers starting and maintaining a side business supporting and selling robots. Duplication of software and other digital artifacts is trivial, so constructing a duplicate of the hardware becomes the primary difficulty. The use of toys for the mechanical components of the robots was intended to reduce the difficulty of constructing the hardware. If researchers are not to be expected to become entrepreneurs, they should also not be expected to become expert machine tool operators. The hardware resulting from this work is designed so that it can be duplicated by a researcher using common tools, and possessed of no more than hobby-level familiarity with electronic hardware.

In order to be both heterogeneous and inexpensive, the robots used for this work were initially designed to be constructed by developing a modular control hardware platform that can be attached to children's toys. The controller module was designed to be used as a replacement for the control electronics of children's toys, similar to the Spider-Bots developed by Laird, Price, and Raptis, or Bergbreiter's COTSBots [Laird, Price, and Raptis, 2014; Bergbreiter and Pister, 2003]. However, unlike the Spider-Bots and COTSBots, this platform did not specify a particular toy chassis to use for mobility. Most children's toys use either one motor with a mechanical linkage to cause the toy to turn when the motor is reversed, or two motors. Two-motor toys frequently use either differential steering or have one motor provide drive power and the other provide steering. All of these toys can be controlled by the hardware described in this work.

The robots are intended to be heterogeneous, partly because of the advantages of heterogeneity in a swarm, and partly because toy supplies are unreliable. While toys in the general case are expected to remain available, a particular line of toys might be discontinued or a modified version released.

The mobility platforms used for the existing TinyRobo swarm cost 12-20 dollars in single quantities, putting the total cost for a single robot at $35-45. Where bulk ordering is available, the cost of 100 mobility platforms is $8 per unit, reducing the per-unit cost of a 100-member TinyRobo swarm to $20. This is roughly in line with the parts cost of the Kilobot, which is $15 [Rubenstein et al., 2014a].

However, this should not be taken to mean that the TinyRobo platform is a competitor with the Kilobots. Kilobots were designed to have scalable interactions, which is to say that programming, charging, and even turning on the Kilobots does not take more time as the number of robots increases. To have programming take constant time, the Kilobots are programmed in parallel using an infrared broadcast

Name	Value	Cost (1)	Cost (100)	Count	Subtotal (1)	Subtotal (100)
Battery	3.7V	3.41	1.50	1	3.41	1.50
C1, C2	4.7uF	0.50	0.20	2	1.00	0.40
C3	0.1uF	0.10	0.02	1	0.10	0.02
C4	10uF	0.19	0.06	1	0.19	0.06
Charge IC	MCP73831	0.59	0.44	1	0.59	0.44
Charge LED	Green	0.54	0.30	1	0.54	0.30
Diode	GF1A	0.51	0.23	1	0.51	0.23
Motor driver	DRV8830	2.27	1.64	2	4.54	3.28
Header	6-pin	0.52	0.37	1	0.52	0.37
PCB		3.30	0.79	1	3.30	0.79
R1	470 ohm	0.10	0.02	1	0.10	0.02
R2	2k	0.10	0.02	1	0.10	0.02
R3, R4	0.22 ohm	0.46	0.13	2	0.92	0.26
R5, R8-12	10k	0.10	0.01	6	0.60	0.06
R6, R7	1k	0.10	0.01	2	0.20	0.02
Switch	410-2016	0.91	0.72	1	0.91	0.72
Thermal Fuse	0ZB0050FF2G1	0.13	0.10	1	0.13	0.10
V Regulator	MIC5265	1.40	1.06	1	1.40	1.06
Wifi	ESP-8266-03	4.32	2.25	1	4.32	2.25
Total Cost for One Robot					23.38	11.9

Table 3.1: Prices in US Dollars for TinyRobo components.

device that illuminates the entire swarm at once. To charge together, the Kilobots have one charging contact on their legs, and the other on a leaf spring on their tops. By sandwiching the robots between two conductive plates, the entire swarm can be charged at once. Finally, to all be turned on quickly, the Kilobots never turn off. Instead, they enter a low power sleep mode, and wake occasionally to check for an infrared signal to become fully active. This last attribute makes it very difficult to have a robot that has both wifi and a very low power sleep mode. The wifi module used in the TinyRobo platform consumes 15mA in its highest-power sleep mode, and 20μA in its lowest power mode. Unfortunately, only the highest power sleep mode can remain connected to an access point and receive a wake-up command, and so it will deplete the 750mAH battery used in the TinyRobos in just over two days. In contrast, a Kilobot can sleep for 3 months.

The processor of the controller is an ESP-8266 wifi module. The ESP-8266 costs approximately $3-5, and contains both a wireless interface and a micro controller that can be programmed from a variety of programming environments and languages, including Lua and the Arduino variant of C/C++. The ESP-8266 module is based on the ESP-8266 IC, made by Expressif Systems. The IC itself has an 80Mhz Tensilica Xtensa L106 processor with 64kB of instruction memory and 96kB of data RAM. The modules come equipped with 512kB to 16MB of flash memory for program storage, and some combination of the 16 GPIO lines of the IC available for use. The ESP-8266 is available in several form factors, each designated by a different suffix. The version selected is the ESP-8266-03, which offers more GPIO pins than most other versions, and includes an internal antenna.

In addition to 802.11 b/g/n WiFi, the ESP-8266 supports a variety of serial protocols, including a UART, I^2C, SPI. The I^2C interface is used on the board to connect to two DRV8830 motor driver ICs by Texas Instruments. The

DRV8830 provides 1A of drive current. Experimental tests with 8 different toys indicate that small toys draw well under 1A while moving freely, and peak around 2A when the motors are stalled. The tested toys include 3 insect-styled walkers, 3 wheeled vehicles (2 differential drive, 1 Ackerman steering), 1 toy helicopter, and 1 toy quadcopter. The DRV8830 provides overcurrent limiting, so a stall condition or short circuit of the motor leads will disable the motor drive, but not damage the DRV8830.

The control module also provides connections for a 3.7V lithium-ion battery pack, as well as charge control circuitry for the battery. The charge controller allows the robot to be charged from the same USB connection that is used to change the programming of the ESP-8266. Reset and entry into programming mode is controlled by a separate USB-to-serial adapter board, the Sparkfun BOB-11736. Moving this functionality to the adapter board reduces the size and cost of the control module.

Toy Compatibility

Children's toys normally use inexpensive brushed DC motors in their construction. These motors have not been the subject of extensive study, as they are commodity parts. However, it is useful to quantify their behavior to some extent, to determine which kinds of toys can be used with the controller.

Two common types of motors found in children's toys are the RE and FA series of motors produced by Mabuchi Motor, or imitations of these motors produced by other companies. These motors use simple metal brushes and are constructed to be inexpensive, rather than precise. The intended voltage range of the motors varies with different winding types, but according to datasheets available from Mabuchi Motor, the voltage ranges and current draws for motors in this range

Model	Voltage	No Load Current	Max Efficiency	Stall Current
RE-140RA-2270	1.5-3	0.21A	0.66	2.1A
RE-140RA-18100	1.5-3	0.13A	0.37	1.07A
RE-140RA-12240	3-6	0.05A	0.14	0.39A
FA-130RA-2270	1.5-3	0.2A	0.66	2.2A
FA-130RA-18100	1.5-3	0.15A	0.56	2.1A
FA-130RA-14150	1.5-4.5	0.11A	0.31	0.9A

Table 3.2: Current draw for Mabuchi-branded motors.

Motor number	No-Load Current	Stall Current (measured)
Hexbug brand mini spider	0.03A	0.13A (see caption)
Hexbug brand 6-legged insect	0.06A	0.25A
Miniature toy RC car	0.21A	0.8A
Miniature toy RC insect	0.19A	1.13A
Miniature toy RC vehicle	0.37A	0.8A
Miniature toy RC vehicle	0.06A	0.74A
Toy helicopter	0.07A	1.12A
Toy quadcopter	0.74A	1.99A

Table 3.3: No-load and stall current for coreless DC micromotors. Measurements were performed at 3V supply voltage. The Hexbug mini spider includes a slip clutch, so attempting to stall the motor by holding the toy does not prevent the motor from turning.

are as shown in Table 3.2.

These motors have a volume of around $2cm^3$. For smaller toys, coreless motors are more common. The values in Table 3.3 were measured from six of the toys used in constructing the swarm. The measurements from the toy helicopter and toy quadcopter are included for comparison. While the board can supply sufficient current to control all of these toys, it has not been tested in flying platforms.

Potential for Expansion

The current design for the robots does not include sensors as a cost-saving decision. However, the communication between the ESP-8266 and the motor drivers uses the industry standard I2C bus serial interface. Due to the non-proprietary nature of this

interface standard, it has been widely adopted, and many sensors are available to connect to an I2C bus. For example, Vishay Semiconductor makes the VCNL3020, an infrared proximity sensor with a 200mm range. If greater range is required, The ST Microelectronics VL53L0X Time-of-flight (ToF) laser ranger and gesture sensor provides a 2M range and 1D gesture sensing in a 4.4mm x 2.4mm package. As of this writing, the VCNL3020 is $3.44 and the VL53L0X costs $6.28 in single quantities. These prices are reduced significantly when buying components in bulk, but because they increase the cost, size, and power draw of the hardware, they have not yet been integrated with this platform. Numerous multichannel ADC ICs with I2C interfaces are also available, which permits the addition of analog sensors to the platform.

As a thought experiment, the cost of adding a 6-direction IR transmission and reception board to the TinyRobo platform was explored. This is not a finished design, but as much of the cost in a device of this type is in the semiconductors and PCB, it gives an estimate of the cost. The IR receiver selected would have been the TSOP5700TR, as used in the E-Puck Range and Bearing sensor board, but it is listed as obsolete by the manufacturer, Vishay Semiconductors. The modern replacement appears to be the TSMP6000. The IR LED was also selected to match the E-Puck Range and Bearing board. The AND gate is intended for use as in the Colias swarm robots [Arvin et al., 2009]. By providing a clock signal gated through the AND gate by the data signal, the IR LEDs can transmit coded data to other robots. If coding is not required, as in rangefinding, the clock and data signals can be varied to turn the LED on and off. Colias implements rangefinding by detecting the intensity of received IR light from either reflections of IR emitted by the robot, or the strength of IR messages arriving from other roobts.

Adding a 6-direction range and bearing board to the TinyRobo system

Part number	Cost (1)	Cost(100)	Count	Subtotal(1)	Subtotal (100)
ATMega168	1.40	1.12	1	1.40	1.12
TSMP6000	1.81	1.03	6	10.86	6.18
SFH4255	1.06	0.59	6	6.36	3.54
SN74ACT08DR	0.51	0.32	2	1.02	0.64
PCB	3.30	0.79	1	3.30	0.79
Totals				22.94	12.27

Table 3.4: Semiconductors for a simple IR communication ring, and their prices, in US Dollars. The PCB is the same size and type as the TinyRobo controller, and so has the same cost.

could be expected to cost in the neighborhood of $12-22 USD, depending on the quantity of boards produced.

Firmware

The current version of the robots' firmware is developed in the open-source Arduino development environment. Arduino programs are written in a dialect of C++.

Every robot runs the same firmware. The firmware listens for connections on port 4321 for TCP/IP packets containing one of two types of messages. Messages starting with a 0x51 byte (ASCII 'Q') cause the firmware to respond with a message containing the ASCII string "TinyRobo". This function allows automatic detection of robots on a network by querying all connected systems to determine if they respond in this way.

Messages starting with a 0x4D byte (ASCII 'M') followed by four bytes are motor speed commands. The firmware interprets the first two bytes as the speed and direction for the first motor, and the second two bytes as speed and direction for the second motor. The control bytes are converted to a single byte command for the DRV8830 motor driver and transmitted over the I2C bus to set the motor speed.

Bit 1	Bit 0	Out 1	Out 2	Function
0	0	Z	Z	Coast
0	1	L	H	Reverse
1	0	H	L	Forward
1	1	H	H	Brake

Table 3.5: Truth table for DRV8830 drive direction bits. Coast allows the motor to turn freely. Brake connects the motor leads, resulting in braking using the motor's back-EMF. Z indicates the output is in a high-impedance state

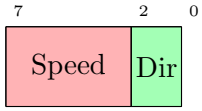


Figure 3.2: Layout of bits in motor command byte for DRV8830

The DRV8830 driver is a voltage-controlled motor driver. It accepts a single-byte command for each motor. Bits 7-2 of the byte define the output voltage to be applied to a motor, and the driver attempts to maintain that output voltage. The valid range of motor voltage commands for the DRV8830 driver is 0x06 to 0x3F, which corresponds to a range of 0.48V to 5.06V in 0.08V increments. Because the robot battery is nominally 3.7V, the motor command 0x30 is the highest output available. Bits 1 and 0 of the command byte control the polarity of the output voltage, and so the direction of the motor, as per Table 3.5.

Once the motor speed is set, the firmware reads the fault bytes from the DRV8830, and sends the motor command and the fault bytes for each motor back to the client over WiFi. The client uses the fault bytes to detect overcurrent conditions in the motor drivers and reduce output power.

The decision to have all of the robots have the same firmware and control the speed of the motors from ROS was made because different toys have different control schemes. Toy tanks use differential drive, toy cars have Ackerman steering, and so forth. By moving the control to the main computer, the firmware can be kept simple while still allowing researchers to adapt the system to the available

toys by modifying the software.

Why Heterogeneity?

Heterogeneity is a good model of many real-world systems where members of a group have different capabilities. Family groups of pack animals have young and old members, sometimes ill members, and sometimes infant members that cannot participate fully in pack activities. In human groups, work is divided according to ability, so a contractor may hire an electrician, a framing carpenter, and a plumber to build a house, to much better effect than attempting to do it with a team consisting entirely of plumbers.

Another use of heterogeneity in swarms is to prevent individual robots from becoming overly complex by sparing them from having to be capable of doing everything. The presence of multiple robots with a given ability in a swarm strikes a compromise between all robots having that ability (and being complex and expensive) and only one robot having that ability (and so providing a single point of failure). Perhaps the most impressive recent demonstration of a highly heterogeneous swarm is the Swarmanoid project's video "Swarmanoid: The Movie", in which three different kinds of medium-sized robots cooperate to retrieve a book from a shelf [OGrady, Birattari, and Dorigo, 2011]. The movie explicitly mentions that one sub-team of robots (two mobile robots and one gripper robot) is positioned as back-up, in case the first sub-team fails.

Beyond the possible utility of robots with multiple abilities, the swarm design presented in this work was intended to be heterogeneous as a matter of convenience of implementation. As toys go out of production and are replaced by others, it may not be possible to continue to operate the swarm on an entirely homogeneous mobility platform. Because of this possibility, the software infrastructure

tries to keep platform-specific calculations in a single module, and allow the rest of the system to operate using standard ROS messages. At present, these calculations only consist of conversion of ROS twist messages, which contain rotational and angular velocities in 3 axes each, into motor speed commands for the robot, which consist of a speed and direction for up to two motors.

Handling the motion of the robots in this way means that the heterogeneity of the mobility platforms has a minimal impact on the conversion of user gestures into programs. However, as a direction for future work, it will become increasingly important to consider mobility as an aspect of program generation. For example, if the system is extended to include UAVs, and the user directs the robots to the center of a lake as part of a task, only the UAVs can be reasonably expected to reach the location undamaged. The system could be extended with some capacity for reasoning about the task environment, to determine how the capabilities of the robots interact with that environment. Such an augmented system could then refuse to direct ground robots into water, and report if there are not a sufficient number of aerial robots to perform the task over the lake. It could also permute task assignments based on robot capabilities in order to meet other goals, such as minimizing the number of robots used or maximizing available robot battery life.

Swarm Robot Software Framework

The individual robots being developed for this research have minimal sensing capacity and relatively weak processors. The majority of the processing is performed on a host computer running the ROS software framework. Each robot's processor is mostly concerned with controlling the motors of the robot. The structure of the software framework is such that as available processing power on each individual

Figure 3.3: The image on the left shows the swarm arena. The top-down camera is mounted on the crossbar at the top. The image on the right shows the camera view before ROS image rectification removes barrel distortion.

robot increases, more of the processing can be handled locally, without changing the overall design of the system.

The central computer has a top-down camera over the "arena" the robots are active in. Each robot has an AprilTag on top of it, so that the central computer can localize them within the arena [Olson, 2011]. The central computer uses the location information to create "virtual sensors" for each robot. Since the central computer knows the location of each robot, the relevant information can be sent to each robot's control process as if it were coming from a sensor on the robot. For example, a range and bearing sensor that allows each robot to detect the distance

and angle of the nearby robots is simple to implement in software. Range and bearing sensor functionality is available in hardware on E-pucks and Marxbots, but since each robot must be equipped with it, the cost scales linearly with the number of robots to equip. It is possible to calculate the odometry for individual robots by watching the change in position in their tags over time. The calculated odometry could then be published as a ROS topic, just like odometry collected from e.g. wheel encoders. The virtual sensors can also be configured to emulate error conditions such as noisy sensors, failed sensors, degraded localization, and so forth. Virtual parameter tweaking allows fine-grained testing of the behavior of algorithms under imperfect conditions, and the response of human users to unreliability in the swarm.

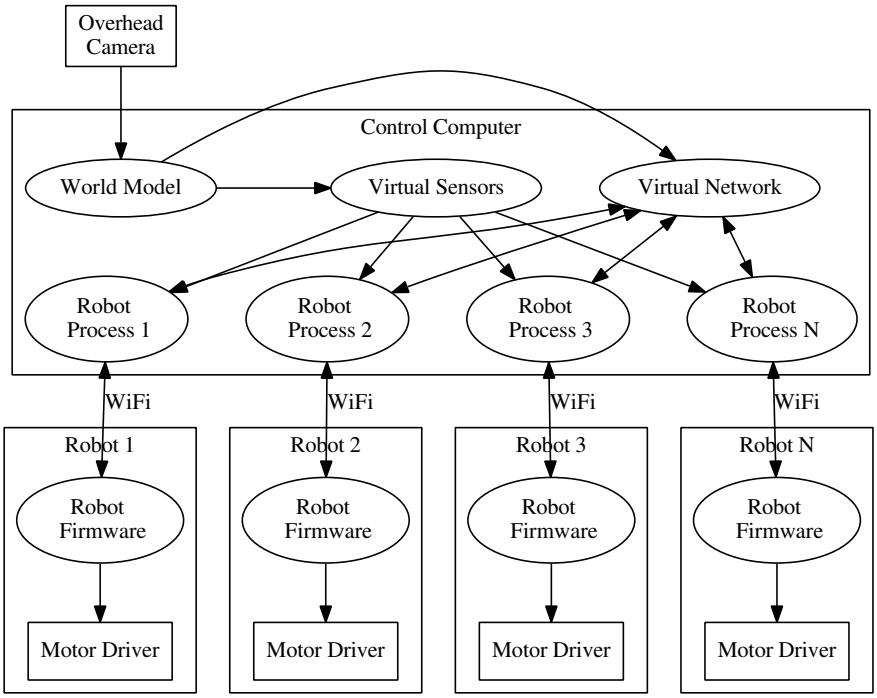


Figure 3.4: Overview of the software framework. Rectangular nodes are hardware, oval nodes are software.

Since the robots are reporting to a central server, and the central server also receives the video from the overhead camera, it may appear that this is a

highly centralized system. However, the central computer provides a framework for implementing a decentralized control scheme on the individual robots. Rather than controlling each robot, the central computer maintains a separate process for each robot in the swarm. Each of these robot processes only has access to the information that would be available to that robot, based on its physical location, and so acts as a local control program for the robot, but with the full processing resources of the host computer. As a result, the individual robots can be small, lightweight, and consume relatively little electrical power, but the system as a whole gives them significant computing power. When more powerful and lower power consumption processors become available, more of the processing can be moved from the virtualized robot processors and onto the actual robots, enabling a smooth transition from a simulated decentralized system to a real decentralized system.

Virtual Localization

The AprilTag tracking of the robots provides localization of the robots within a common coordinate frame. It should be stressed that while the central computer can localize the robots, both relative to each other and by absolute position within the arena, this information may be withheld from the individual robots, or given to them if required. The code virtually operating on the robot may be neither aware of its own position in the world, nor the location of other robots, if the experiment calls for such a lack of information.

Currently, the AprilTag-based localization is used to implement virtual laser scanners similar to the Sick or Hokyuo brand laser scanners used on larger robots. It is also used to limit the range of messages sent between the robots through a virtual network, and to implement range and bearing sensing between robots.

Range and bearing sensing to other robots is a very important sense for swarm robots, as indicated by its presence in algorithms and frequent implementation in swarm robot hardware. As discussed above, Colias and E-Pucks both include IR-based range and bearing sensing. The parts cost of a Colias-style six-direction range and bearing sensor is around $12, while the more sophisticated E-Puck range and bearing extension board is $397. Virtualizing this hardware results in an immense cost savings.

Virtual Laser Scanners

The AprilTag localizations and the image of the arena are used to provide virtual laser rangers for each robot. The virtual laser ranger consists of two ROS nodes, a service and clients for the service. The service is called "laser_oracle_server". It subscribes to the AprilTag detections and the images from the arena overhead camera.

When a client requests a laser scan, the virtual laser service masks the modified arena image with a circle with a radius of the laser range, centered on the robot requesting the scan. This masking removes all of the objects that are out of range of the laser, and so reduces the time spent calculating the laser scan points.

Each sample of the laser scan is represented as a line segment, located based on the requested start, stop and inter-measurement angles for the virtual laser scanner. Each line segment is checked for intersection with the lines defining the contours of the blue objects in the image. As the virtual laser service receives images, it draws a blue dot over the location of every robot. This dot provides the outer edge of each robot in the virtual laser scan. The approach of using blue objects as obstacles was chosen because if the laser scanner service treats anything blue as an obstacle, then "walls" can be created in the arena by making lines of

blue masking tape on the arena floor. If multiple intersections are found for a line segment, the intersection closest to the robot is used, as the laser would stop after reflecting off an object. The service then formats the distances to the intersection points as a ROS sensor_msgs/LaserScan and returns it as the service response to the requesting client.

The virtual laser clients take the place of the laser driver ROS nodes that would be used to control a real linear laser scanner. The laser client is initialized with some parameters, such as the sweep angle and angular resolution of the virtual laser, and polls the laser service regularly. As it receives laser scans from the service, it publishes them to a ROS topic in the same manner as a ROS node for a hardware laser.

The apriltags_ros node publishes the detected locations of the tags in meters, but the computer vision detection of blue objects in the arena camera image operates in pixels. In order to convert from pixels to real-world distances, the apriltags_ros node was forked and a modified version was created that provides the locations of the tags in pixel as well as real-world coordinates. The modified version is available at https://github.com/ab3nd/apriltags_ros.

Virtual Networking

If the robots are required to communicate directly with each other, the communication passes through a virtual network. From the point of view of the robots, messages sent into the virtual network are delivered to other robots as if the messages were sent directly from one robot to another. However, all the communication is taking place between processes running on the central computer. By changing how the messages are delivered by the central system, the virtual network can provide full connectivity, range-limited mesh networking, directional beacons, or

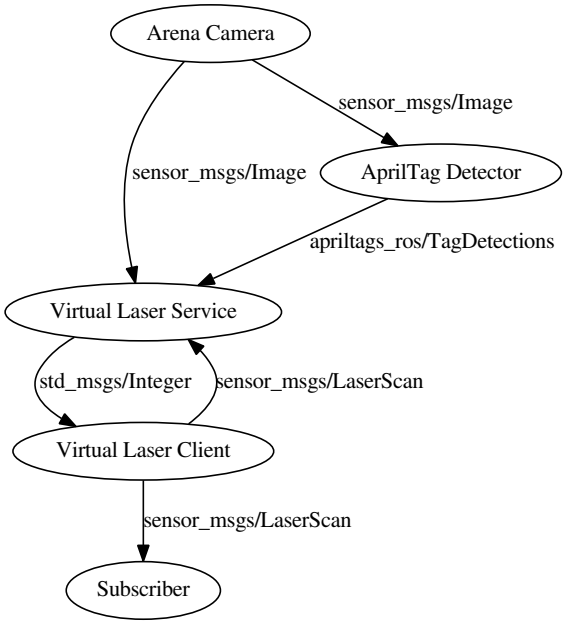


Figure 3.5: Data flow in the virtual laser service

other forms of networking. The reliability of the network can also be varied, by dropping some messages or otherwise changing them based on information about the robots. For example, the likelihood that a message arrives at the robot that it was transmitted to may depend on the distance between the sender and receiver. Signals that pass through a virtual wall may have a reduced virtual signal strength and range.

Swarm Hardware Results

Due to mechanical flaws in the toys used as motion platforms in TinyRobo, the robots would sometimes not move as commanded (see Figure 3.7). In a single-robot system, transient failures can sometimes be accommodated by repeated effort or replanning. However, in a multi-robot system the scale of the system works to offset the reliability of each individual robot. If robots have an individual mean time between failures (MTBF), the expected mean time to any failure is the MBTF

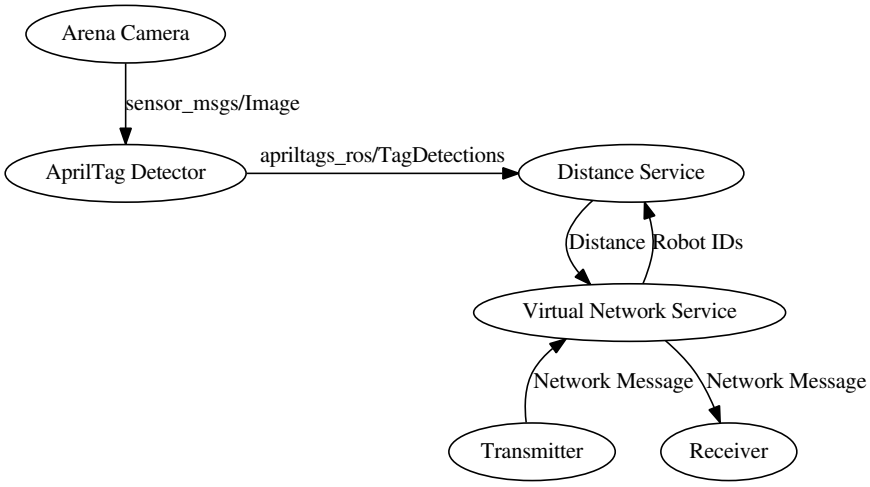


Figure 3.6: Data flow in the virtual network. The virtual network service can take the distance between the transmitting robot and the receiving robot into account when determining if the message is delivered.

divided by the number of robots. For example, if an individual robot can work for 100 hours between failures, it would be reasonable to expect it to work for at least a day. However, if the swarm consists of 1000 such robots, a failure of at least one robot can be expected within 6 minutes. Tracking the failure of existing platforms in the field placed the average MBTF of commercial mobile platforms at 24 hours, with indoor robots fairing much better than outdoor ones [Carlson, Murphy, and Nelson, 2004].

This problem was highlighted by the use of inexpensive toys. It is desirable to have a swarm platform be able to run for extended periods, in order to acquire data for experiments. Toys, especially cheap ones, are designed for low cost and an MBTF more compatible with the attention span of children than that of researchers. The amount of effort devoted to locating and eliminating mechanical problems relative to the runtime of the system was not acceptable.

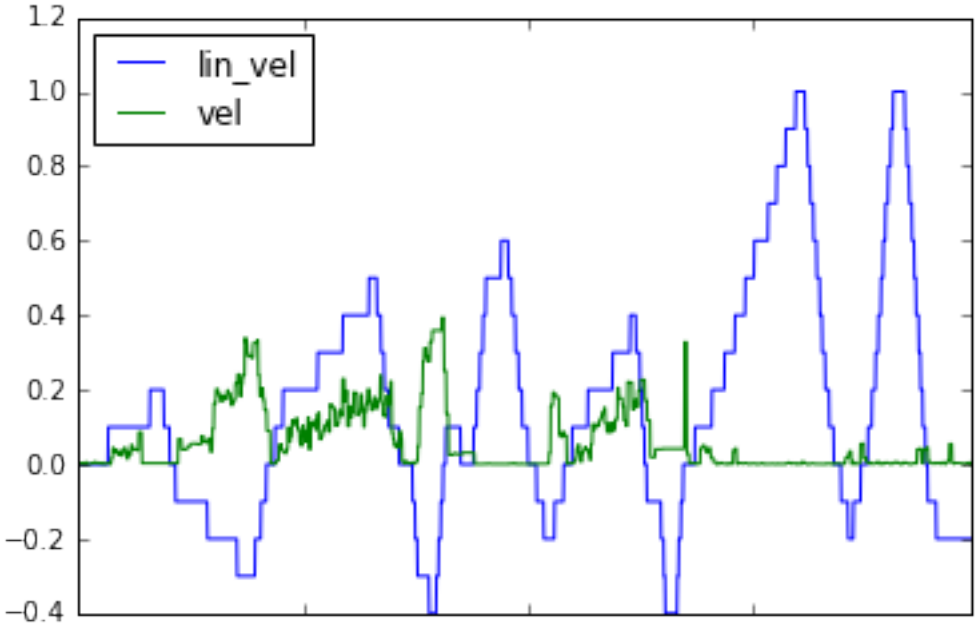


Figure 3.7: Commanded velocity (lin_vel) as opposed to recorded motion (vel). Vel is always positive because it is measured in terms of euclidian distance moved by the center of the AprilTag between successive updates of the tag tracking. Note that while the magnitude of the motion is proportional to the commanded motion, sometimes the robot did not move at all, and when it did move, the recorded velocity is quite noisy. Noise may be removed in software, but mechanical failure cannot.

Calibration

Children's toys are prone to failure and inaccuracy. In toys, particularly remote controlled toys, the user acts as a the feedback element of a control system, observing the behavior of the toy and changing their actions as a result. If an individual toy, for example, has a bias to turn to the left, the user will learn this and apply a correcting bias. To extend this to a computerized system requires some form of calibration. These calibration steps, combined with appropriate control, such as PID loops, can account for systemic inaccuracies. However, the use of toys also introduces some failures that are not consistent or linear. For example, the tanks used in some instances of the TinyRobo platform use motors with high speed,

but relatively low torque. As a result, dirt in the drivetrain near the motor can cause the motor to become difficult to start, but can be removed by the user, or by operation once the motor does start. Calibration when the dirt is present means the robot will start very abruptly when the dirt is removed, while calibration when the dirt is absent means the robot may not start if dirt gets into it later. It is worth noting, in light of this example, that GritsBots, Kilobots, and mROBerTO all use nearly-sealed drivetrains, either direct motor drive of the wheels or sealed vibration motors [Rubenstein et al., 2014a; Pickem et al., 2015; Kim et al., 2016].

An early intent of the author was to have the system learn the control law for each robot through observation of the relationship between the commanded motion of the robot and the resulting motion. Due to the overarching concern with human control of a swarm, such online calibration was decided to be out of scope for this work. However, a computer-vision-guided calibration technique was used in the mROBerTO swarm robots to compensate for manufacturing differences between robots [Kim et al., 2016]. This approach results in learning bad controls if the system observes the robot during a temporary failure. In a system with minimal failures, this problem can be minimized, but as discussed earlier, inexpensive children's toys are not such a system.

GRITSBots have a calibration step, but the calibration is automated, and is only performed once for each robot, after which it is assumed that the calibration variables are constant. Calibration can be automated in a homogenous platform with reasonably reliable hardware. Non-homogenous platforms require different calibration for different platforms, which reduces the benefit of automation. Further, requiring calibration works against transitioning to a new platform by adding an additional hurdle in the form of developing a new calibration method.

The lack of an automated calibration and control method drove the

TinyRobo swarm to use more differential drive vehicles, as they struck a balance between the fully holonomic drive used in the SpiderBots, which is expensive but easy to control, and the Ackerman drive used in inexpensive RC cars, which is less expensive, but has more complex control math [Laird et al., 2014]. It also increases the difficulty of using a heterogenous system, and so operates against the advantages of heterogeneity as discussed in Section 3.1.4

Drive Testing

In order to characterize the behavior of the various toy hardware that the TinyRobo platform could be used on, seven TinyRobos were commanded by a program that instructed them to move forward until the AprilTag tracking determined that they had moved a fixed distance, stop, and move again, in a loop. This program was run until the robot failed to move or ran into the walls of the robot arena.

Each robot was run 5 times, starting from the same location each time. The robot was power-cycled between runs. Each run was recorded using rosbag, and the data from the recordings were used to generate visualizations of the robot's commanded trajectory from the motion of the center of the robot's AprilTag.

The robots consisted of three toy tanks from the same manufacturer, a differential-drive Hexbug-brand robot, a differential-drive toy car with large wheels, a hexapod bug-like walker, and an Ackerman-drive toy car. The bug-like walker and the Ackerman-drive car both have different turning kinematics from the differential drive tanks, but this test only consists of forward drive. It would have been preferable to have the test include turning, but control of the yaw rotation of a robot via feedback from the AprilTag system was found to be problematic. As the number of pixels that an AprilTag takes up on a camera image decreases, the accuracy of the subpixel estimation used to localize the corners of the tag decreases.

Each estimation of the tag location from the ROS AprilTag detection node can then differ from previous estimations, and so even when the robot remains still, there is some noise in its detected position. Because it was assumed early on that the AprilTag fiducial tracking would be sufficiently accurate, the tradeoffs between camera resolution, tag size, and tracking speed have not been fully explored. In some configurations, AprilTags can have very solid position and orientation tracking, but are computationally intensive to detect and localize in typical webcam images. More tags also leads to longer computation time, increasing the latency of the robot control loop. Parallelizing the implementation of the AprilTag library could improve it significantly, but is out of scope for this work. The latency and accuracy problems with AprilTags could be mitigated in a number of ways. First, the resolution of the camera can be increased while keeping the tags the same size and the camera the same distance away. Increasing the image size requires more computational power to process the larger image, in order to keep the tag detection framerate high enough to be useful for realtime control. Decreasing the size of the image being checked for tags increases the speed of tag detection, at the cost of requiring larger tags in order to have them legible in the lower-resolution image. The update rate places an upper bound on the ability of the system to respond to the motion of the robots. The tag size could be increased, either by moving the tags closer to the camera and increasing their visual size, or by actually making the tags larger. Moving the camera closer reduces the useful area of the robot arena, and making the tags larger increases the size and weight of the robots, potentially overbalancing some of the less nimble robots.

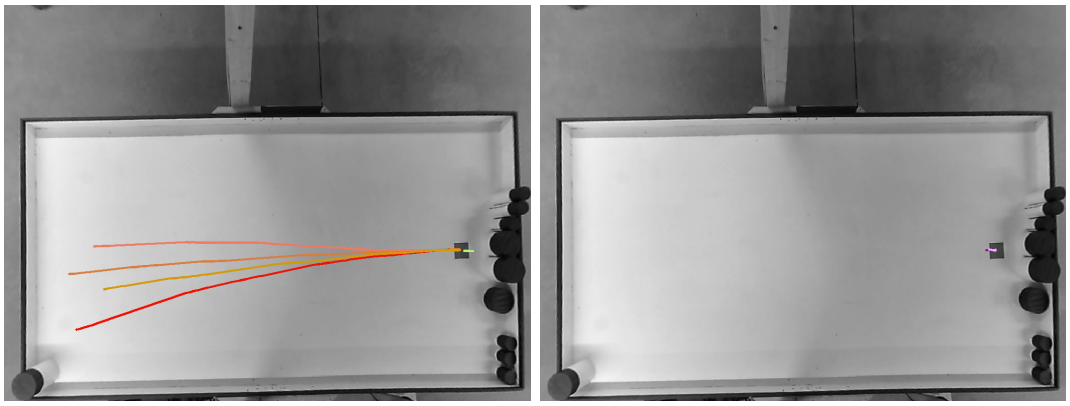
The effect of the noise in subpixel position on the perceived rotation of the tags is larger than its effect on the perceived rotation of the tags, since rotation around the center of the tag does not change the position of its center at all. As

a consequence, the tags could be detected to be in lateral motion by comparing the displacement of the center of the tag and checking that it was greater than the expected noise, but detection of rotational motion and velocity calculation was sometimes swamped by noise.

The noise was also determined to be unevenly distributed over the robot arena. Because the arena is wide, a 140° wide-angle lens is used to ensure that the entire arena is visible. ROS provides tools for removing the distortion inherent in the use of a wide-angle lens, but at the cost of decreased effective resolution at the edges of the image. This decrease in resolution results in reduction of subpixel location accuracy, and so increased noise in localization of the tag at the edges of the arena closest to the edges of the camera image.

Because the desired figure-8 motion could not be performed with the rotational localization noise present in the system, the robots were instead commanded to move forward 0.25m, stop, and repeat that sequence of actions until the program was stopped. The program was stopped if the robot was about to run into the wall of the arena, or had entered a state in which it could not move forward anymore. Acceleration during the movement phases was managed by increasing the commanded velocity of the robot until the AprilTag tracking detected that the robot had begun moving. Once the robot began moving, it was not commanded to change velocity until it had moved at least 0.25m. In the following descriptions of the motions of the robots, the directions left and right are relative to the direction of travel of the robot.

The Ackerman-steering toy car (Figure 3.8a) displayed very consistent mobility. In four out of its five runs, it crossed the arena without incident. In one run, it started and went a small distance, but then did not start again. However, the Ackerman-steering toy car has a gearbox that permits backdriving, so when



(a) Motion of toy car based robot, showing long tracks across arena

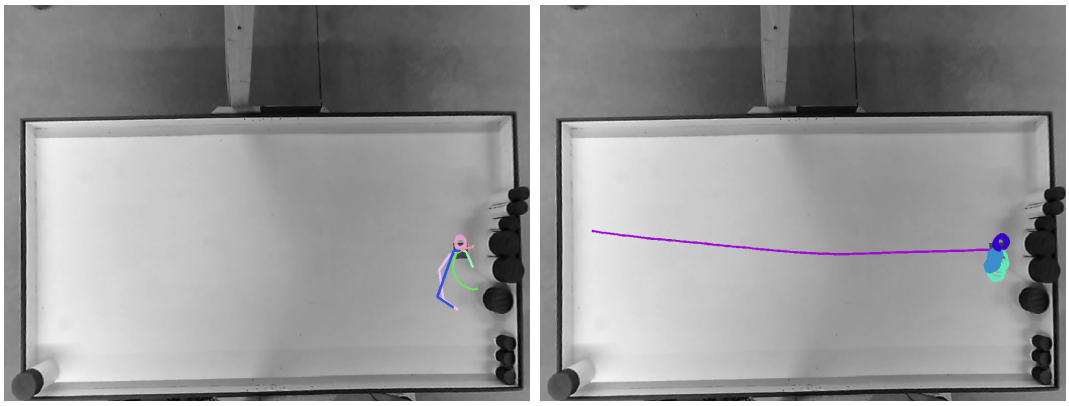
(b) Motion of big wheel robot, showing no track due to the loss of tracking as the robot either did not move, or flipped over

Figure 3.8

commanded to stop, it coasts for a few centimeters. The toy tanks use a worm gear in their drivetrains, and so do not permit backdriving of the motors by the vehicle's inertia. Instead of coasting, they immediately stop when the motor is commanded to 0 velocity.

The big wheel robot (Figure 3.8b) displayed an unfortunately small range of commanded velocities between those that caused it to begin moving, and those that caused it to flip over on its side. In three of its five runs, the big wheel accelerated quickly and then flipped during the first motion period. In the remaining two runs, it did not move at all, possibly due to gears jamming.

The Hexbug-branded 6-wheeled bug (Figure 3.9a) displayed an asymmetry in its motor drive speeds. For three of its five runs, both motors ran, and the wheels on both sides rotated, but the right side was driven more quickly, and so the robot made a wide arc to the right and hit the wall of the arena behind the starting location. In one run, one side did not begin moving at all, causing the robot to rotate rapidly around that side. In another run, the robot twitched briefly, remained still, and then accelerated backwards quickly. This was likely caused



(a) Motion of 6-wheel bug, showing tight arcs and spirals caused by different motor speeds

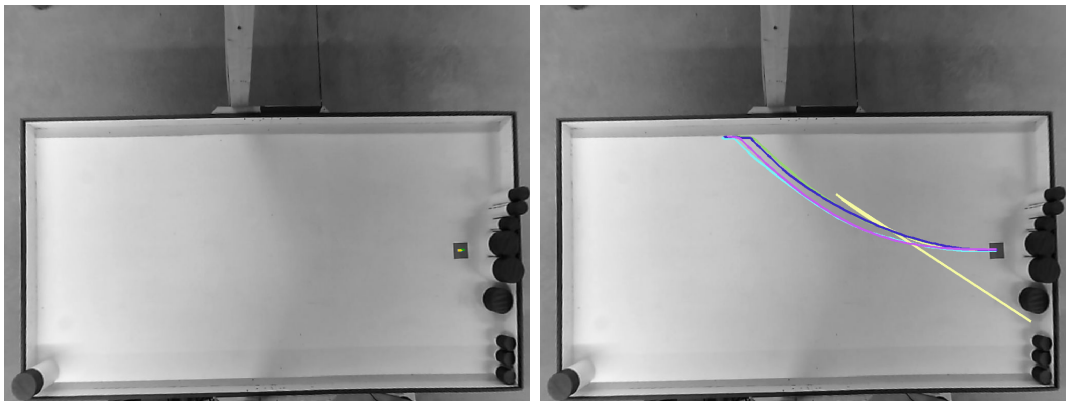
(b) Motion of green tank, showing one successful run and four tight spirals due to a stopped track on one side

Figure 3.9

by the gradual incrementing of the forward velocity eventually causing an integer overflow, resulting in a large forward velocity command being interpreted as a large negative velocity.

The green tank, carrying the number 0 AprilTag (Figure 3.9b), experienced problems with one side of its drive train in three of its five runs. One tread drive did not move, while the other did, resulting in tight turns to the left in two runs, and to the right in one run. In one of the two remaining runs, the green tank did not move at all. In the second, the tank alternated movement and stopped periods until it reached the other side of the arena, which constituted success on this test.

There are two blue tanks, carrying AprilTags numbered 8 and 18. Tank number 8 (Figure 3.10a) moved slightly on four of its 5 runs, and did not move on one of them. At no point did it move the full 0.25m. Tank number 18 (Figure 3.10b) moved much more consistently, but displayed an arc to the right in four of its 5 runs. In one run, the tank failed to restart after one of the stop phases, and eventually accelerated quickly in reverse. As with the 6-wheeled bug, this was likely



(a) Motion of blue tank #8, showing lack of motion

(b) Motion of blue tank #18, showing arc to the right on most runs, and overflow leading to sudden reverse (light yellow track)

Figure 3.10

the failure of the robot to move leading to a long enough delay that incrementing the commanded acceleration resulted in an integer overflow.

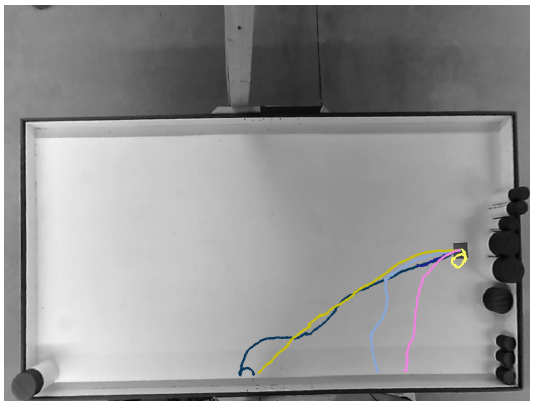


Figure 3.11: Motion of bug robot, showing tendency towards the left and more erratic path than tracked or wheeled robots

The single-motor Hexbug-branded blue bug (Figure 3.11) moved consistently, but with a heavy bias towards turning to the left. In every run, it started and stopped, but the curvature of the path to the left caused it to run into the left side of the arena. During one run, it fell over while moving, due to the somewhat "bouncy" nature of the toy's gait.

Some of these problems, such as the coasting of the Ackerman-steering car and the biases towards the left or right of some of the robots when commanded to move straight, can be overcome by software. For example, assuming the tags could be used to accurately measure the rotation of the robots, error between the detected rotational velocity and perceived rotational velocity could be accounted for in subsequent motion commands. The Ackerman-steering car drift could be reduced by sending a "brake" command to the motor driver IC to activate its back-EMF braking mode, rather than simply stopping by reducing the power output to zero. However, some of the problems are more difficult to alleviate. The tendency of the big wheel robot to flip over could be mitigated by very gradually increasing its speed, at the cost of reducing its control responsiveness, or by increasing the frequency of the feedback loop that checks its speed. Perhaps the blue bug would not fall over if its height were reduced, or the battery were placed lower on the body of the robot. However, these mechanical problems could be more easily avoided by simply not using toys that have them.

3D Printed Robots

Because the toys were demonstrated to be unreliable as motion platforms, a new design for the motion platform was developed using a 3D printer. For this version of the TinyRobos, the motor selected was a commercially available miniature gearmotor. The gearmotor was selected because it was cheap, easily available from Amazon.com, and used the same voltage that the TinyRobo drive boards used. The motors used were listed on Amazon as "NW 3pcs 3V Micro Planetary Reducer Motor High Torque DC Motor DIY Robot Gearbox Motor". Unfortunately, these particular motors are from an unknown supplier, and as such, may become unavailable. However, a number of sellers are offering the same motors, or motors of

similar size that could be used as replacements if the 3D print design were altered. The motors cost $9.69 for a package of three, so $3.23 each or $6.46 for the pair needed for a robot. Given that the toys being used as mobility platforms cost $8-20, depending on the volume purchased, even purchasing these motors at retail cost was a savings over using toys. The motor has a planetary gearbox, and a plastic output shaft.

The 3D printer used is the Monoprice Maker Select Mini (MPMSM), which costs approximately $200 at the time of this writing. The MPMSM has a 120mm^3 build volume, and can print two robot chassis at the same time, as well as wheels for them. Wheels were designed to be 3D printed along with the robot, and glued to the output shaft of the motor. 3D printing provides the flexibility to change the motor mounts of the robot to accommodate various motors. The 3D printed components of the robots do not add significantly to the cost of the robots. Material for 3D printers is generally sold in spools containing 1kg of filament for $15-35, although some types of material are much more expensive, depending on the qualities of the material. The robots described were printed in Hatchbox brand polylactic acid (PLA) filament, which costs $20/kilogram. The resulting prints are approximately 25g of material, and so contain $0.50 worth of material.

On top of the 3D printed TinyRobo, posts can hold up an LED ring for color blob tracking, or any kind of trackable code. The LED rings have 12 RGB LEDs, which can be lit to indicate the robot's heading to the camera, but can also convey information to the user. For example, robots could illuminate LEDs based on the value of a sensor, resulting in a visible map over the swarm of the sensed quantity at the location of each robot. They could also illuminate a single LED pointing to the next robot in a chain, and so indicate a path that the user could follow.

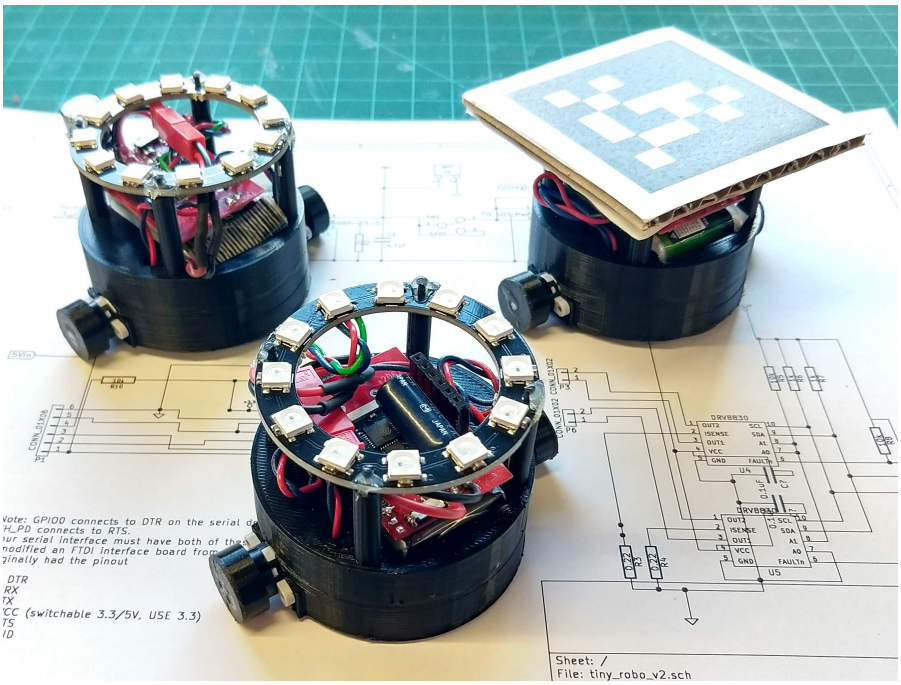


Figure 3.12: 3D printed robots, two with LED rings and one with an AprilTag for tracking. The LED rings are intended to display computer-trackable constellations and human-readable information.

Because the 3D printed robots all used the same motors, there was no need to have them use an adaptive velocity control for drive testing as in the drive test for the toys. Instead, they could be commanded to move at a fixed velocity, and observed from the camera. However, the low output speed of the geared motors meant that the thresholds for detecting that the robot had stopped and started had to be changed. Once the program was adjusted so that it could detect the robot's motion or lack thereof correctly, the robots were commanded to move 0.25m, stop, move 0.25m and so forth until they ran into a wall of the arena. Once the robot hit a wall, it was reset and run again for 5 attempts. Each 3D printed robot was tagged with an April tag, numbered 2, 3, and 5.

Robot 2 missed the first stop command on two of its runs, but worked normally for subsequent stop commands. It stopped and started regularly on the

other three runs. Robot 2 made a long arcing path to the left, indicating that one motor was running slightly faster than the other, or acquiring more traction. This sort of error can be compensated for by feedback, assuming the robot can detect its own trajectory.

Robot 3 missed all the stops on its first run and the first stop on its second run. On its last three runs, it started and stopped as expected. Rather than arcing left as with Robot 2, Robot 3 arced to the right.

Robot 5 also had an arc to the right. For its first run, it missed the last stop, but otherwise functioned correctly. On its second run, Robot 5 started and stopped correctly, but hit something on the floor of the arena that caused it to turn more sharply right. On its third run, the system incorrectly detected that robot 5 had started after coming to a stop, and so began waiting for it to move 0.25m. The robot was not actually moving, so this condition would not have resolved. For run four, Robot 5 stopped during its first move and did not restart. The cause of this error is unknown, but it is possible that it was erroneously detected as having started before it did, and so was not sent motion commands to start. On the last run, Robot 5 started and stopped as commanded until it was nearly at the wall, and then one wheel got stuck on a bit of wood on the arena floor, causing the robot to spin around that wheel.

Overall, the 3D printed robots were more reliable than the majority of the toy bases, in terms of runs completed. However, a direct comparison cannot be easily made due to the changes that had to be made to accommodate the new bases. The use of a fixed velocity command for the forward motion is particularly problematic, as the gear ratio on the 3D printed robot motors is much higher than the gear ratio on most of the toys. As a result, a command that moves the 3D printed robots at 3-4cm/sec will set the toy tank bases moving too quickly for the

camera to track them. The detection of robot motion also had problems due to the noise in the April tag detection and the low speed of the robots. The missed stop commands in the runs of the 3D printed robots were caused by the movement of the robot between successive updates of the tag tracking being sufficiently small that it was less than the expected noise of the AprilTag tracking, and so was considered noise and ignored. As a result, the robot was never detected to be moving, and so was never commanded to stop. Similarly, when the robot was moving, if its detected motion was sufficiently slow that it would fall below the noise floor, the robot was detected as having stopped, and so would not be sent a command to stop, as the system regarded it as having already stopped. Given more accurate tracking, the 3D printed robots could be expected to have fewer false starts and erroneously-detected stops.

Currently, the 3D printed TinyRobo base suffers from a problem also seen in E-Pucks: the edge of the robot can catch on small obstacles, and so there is still at least a little work that could be done to improve the shape of the 3D base. The flexibility of the 3D printed design also opens up possible extension to hardware with e.g. whegs or other unusual forms of mobility. These directions have not been explored in any depth. It is unlikely that any single design is best for all users, but development of various designs using the same control hardware could produce a library of designs with different features to suit different tasks.

Conclusion and Discussion

One goal of the personal project that this work developed from was to provide an inexpensive swarm platform that could be used by hobbyists and other non-traditional researchers to work with swarms. The required reduction in price was

intended to be accomplished by combining the diminishing cost and increasing power of Internet of Things (IoT) networking modules with the ready availability of toys to create a system that lowers the barrier to development of multi-robot systems.

The use of toys as a source of drivetrains for swarm robots proved to be unsatisfactory due to the lack of reliable motion provided by inexpensive toys. Sensors and networking can be virtualized, as in TinyRobo and GRITSBots, but no amount of clever programming will compensate for balky motion. The use of direct drive, as in GRITSBots or mROBerTO, provides a more reliable method of locomotion, because the resulting drive train will be sealed against foreign matter. Further, the use of stepper motors in GRITSBots provides some degree of precision in motion control by directing the motor in steps of known resolution, rather than commanding a particular speed. If the system requires additional torque, sealed micro gearmotors can provide increased torque (although with reduction in speed), and will be more reliable than adapting a drivetrain from a toy.

Adapting drivetrains from toys also adds the expense of purchasing the toy to the total robot cost. This cost is frequently unnecessary, as the chassis of the robot can be constructed from the same printed circuit board (PCB) that the electronics are supported on. Over the scales of forces present in tabletop swarms, PCB can be considered completely rigid, and electronics solder provides sufficient mechanical strength for motor mounts. The use of custom mechanical assemblies (e.g. in Jasmine micro robots) adds complexity to the build process. Where possible, small robots should be designed to use the PCB instead. Using children's toys in TinyRobo was intended to avoid the use of custom parts, but brought with it additional problems that were outside of the scope of the work to solve, and could have been avoided with a simpler drivetrain. The use of hobby-level or toy-like

platforms for swarms is not certain to be a dead-end, as the UB Swarm and the Spider-bots demonstrate, but the cost of sufficiently high-quality toys to produce a sufficiently reliable swarm will drive the overall cost up [Patil, Abukhalil, Patel, and Sobh, 2016; Price, Laird, and Raptis, 2014].

Ultimately, the resulting swarm hardware platform does demonstrate that Hypothesis 1, that commodity hardware can enable the construction of an indoor swarm for under $30 per robot, is correct, assuming parts are purchased in sufficient quantities. As seen in Table 3.1, the cost of an individual TinyRobo control board is just under $12, if parts are purchased in quantities for 100 robots, which is on par with the cost of Kilobots, which cost $14 for parts, if the parts are purchased in sufficient quantity to produce 1000 robots. The 3D printed chassis consumes approximately $0.50 worth of material, and the motors add just under $4 in 100-robot quantities. The total price per robot is then $16.50, well under the $30 target. Unfortunately, in single quantities, the cost of the TinyRobo controller climbs to $23, and the cost of the motors to $6.96, putting the total cost including 3D printer material at $30.44. This is only slightly over the target cost, and could likely be reduced by changing the selection of semiconductors in the power regulation section of the TinyRobo board, or by finding slightly cheaper motors.

Chapter 4

User Gesture Collection

Multitouch gestural interfaces, like those found on tablets and smartphones, offer the possibility of a very direct user experience, especially compared to the Windows, Icons, Mouse, Pointer (WIMP) interface design. Rather than, for example, using an arrow key to scroll in a document, the user can drag the document directly, as though they were sliding a piece of paper on a table.

This directness is a hallmark of what has come to be called Natural User Interfaces, or NUI. A natural user interface is one that allows the user to re-use existing skills and natural motions to interact directly with content [Blake, 2010]. In practice, this means that the elements of the interaction are actions such as pointing and other gestures, drawing with a pen, speech, gaze, and so forth, rather than computer-specific interface devices. By way of contrast, the command line interface is defined in terms of typing with some form of keyboard, and the graphic user interface is (in most cases), defined in terms of mouse actions.

However, as the number of operations the user wishes to perform increases, the limitations of multitouch screens become more apparent. Screens are flat, and

so only afford 2-dimensional gestures, such as dragging, poking, and tapping. Even if the screen depicts a 2-dimensional projection of a 3-dimensional world, operations that would make sense in the 3D world, such as grasping, have to be mapped to the 2-dimensional space of operations to be performed. Typically, the gestures used to perform the available operations are chosen by the interface developer, and the user is trained to perform them, possibly by a short tutorial program [Wobbrock et al., 2009; Vanacken, Demeure, Luyten, and Coninx, 2008; Freeman, Benko, Morris, and Wigdor, 2009].

Unfortunately, the use of screens with interactions inspired by the affordances of physical objects leads to the user having to decide between "natural" skills and motions, which are used on physical objects, and the skills and actions used with screens: single-point dragging and clicking. Most users understand that they are looking at pictures of things on a screen, and so default to single-point interactions [Vanacken et al., 2008]. Further, tangible UIs, a subset of UIs that include real, physical objects for the user to interact with, raise a set of possible interactions that the designer may not forsee, or that the system may not be able to support [Hornecker, 2012]. Attempting to leverage the affordances of a physical object by depicting it on a screen adds another gap, where, in addition to failing to account for all of the affordances of the physical object (e.g. hitting it with another object), the system also fails to account for the affordances of a picture of an object (e.g. shrinking or enlarging it, which a real object cannot easily do). Additionally, even if these affordances are handled, they may not have clear meanings. What does it mean to use a photograph to hit a text file?

The behavior of users interacting with an NUI device is not solely informed by their intuitions about the physical objects represented on the screen. Smartphones, tablets, and other multitouch user interface devices, as well as specific

types of computer programs, such as CAD and realtime strategy game programs, also inform the user's expectations about interactions with new interfaces. Rather than claiming that the users are untrained, and the gestures are "intuitive," it would be more accurate to claim that the users already have a form of training, from the devices that they use in their daily lives. This work attempts to discover the gestures that users would choose themselves, based on their own thinking about the interface and their own past experience with computer technology such as tablets, smartphones, and video games.

User-defined gestures have advantages in memorability and user preference over gestures designed by the researcher or interface programmer [Nacenta, Kamber, Qiang, and Kristensson, 2013]. However, the study that indicated this preference was for gestures that each user explicitly defined for themselves, rather than a gesture set that was constructed by eliciting gestures from the users during task execution [Micire et al., 2009]. By exending the task-oriented gesture collection from single or small groups of robots to swarms, this work attempts to discover if the gestures that users select vary with the number of robots available.

Experiment Setup

The multitouch user interface device used in this experiment is a 3M M2265PW touchscreen. This screen can track up to 20 simultaneous points, but reports only points, rather than shapes or areas of contact. While the user interacted with the touch screen, their touches and the positions of their hands were recorded by the computer connected to the screen and by the video cameras. One video camera was placed high, looking down at the screen, to track where the user's hand position over the screen. The other video camera was placed in front of the screen at a low

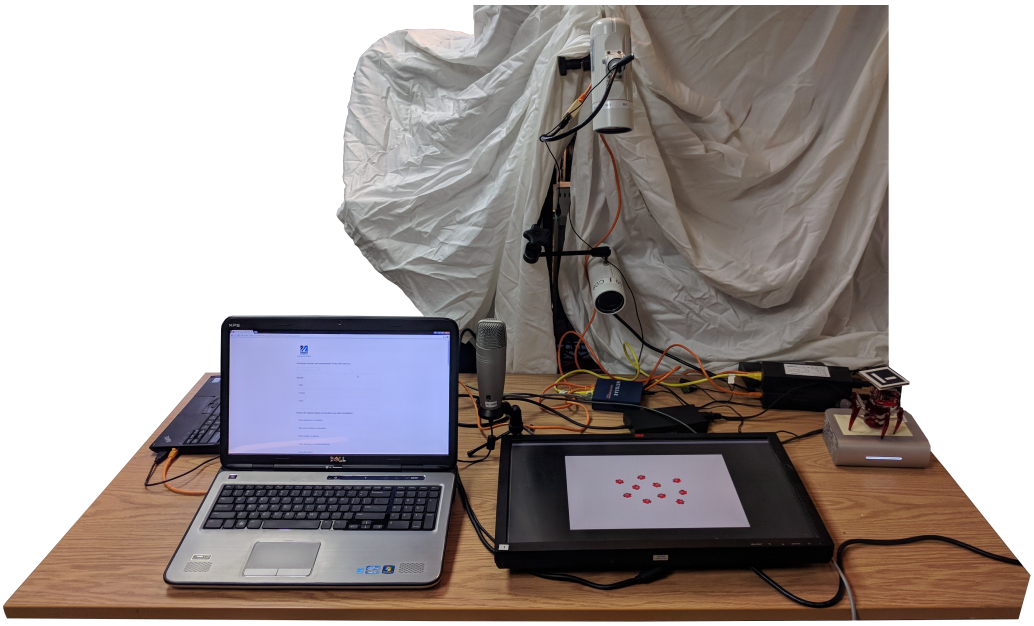


Figure 4.1: Experiment setup, showing, L to R, the survey computer, microphone, cameras and multitouch interface device, and an example robot.

angle, in order to observe whether the user's hands were touching the screen, or moving above it. In addition to screen contacts and video, users were asked to think aloud about their actions. A microphone placed near the screen was used to record everything that the user said.

The software used to record all of this information is ROS, the Robot Operating System [Quigley, Conley, Gerkey, Faust, Foote, Leibs, Wheeler, and Ng, 2009]. ROS was developed as a message-passing framework and hardware abstraction layer for robots. Software using ROS is implemented as "nodes," which communicate by passing messages, generally in a publisher/subscriber pattern. The format of the messages is formally defined, and the generation of the code for generating messages and handling routing of messages is provided by ROS.

It may seem unusual to use a framework intended for operating robots as a recording program for collecting experiment data, but ROS provides a utility called rosbag that records some or all of the messages emitted by ROS nodes in a

"bag" file. In this case, the cameras, microphone, and UI application are all recorded by rosbag. A ROS launch file starts multiple ROS nodes to record image data from the cameras, audio from the microphone, and touch events and screen updates from the UI. ROS also provides tools for manipulating bag files, and playing them back. All of the data in the file is timestamped, so it plays back with the audio, video, and UI interactions all accurately synchronized. Because all of the data is treated as standard ROS message types, it is relatively easy to write custom processors for the recorded data. For example, a node was written that accepts the replayed UI screen changes and touch events, and renders them as a stream of ROS image messages showing the contact points overlaid on the UI screen. Ultimately, the entire data stream was rendered to video, as the rosbag playback does not support rewinding the playback, and being able to re-view a section conveniently was useful for coding.

Users were seated in front of the interface and read a script describing the system and the experiment. The user interface displayed alternating slides of instructions to the user, such as "Move the robots to area A", and interface screens for them to interact with. The interface did not visibly respond to user contact or move the robots depicted on it. In this regard, it more closely resembles the paper prototypes of the User-Centered Design process than a fully functional interface [Ehn and Kyng, 1992].

Some users were initially confused by the interface not responding. It may be that running this experiment on a computer, rather than a paper prototype, contributed to the user expectation that the system could react. Paper prototypes, obviously, do not change in response to the user performing an action, although the experimenter may show the user new sheets of paper depicting the next state of the interface. However, most users' experience with touch screens is that when

they touch them, something visible happens nearly immediately. A system that does not visibly react, as in this experiment, is usually assumed to be broken, or waiting for further input.

For future work, it may be desirable to structure attempts to elicit user gestures as in Wobbrock *et al.* [Wobbrock et al., 2009]. The experiment described in this section showed the initial situation, and asked the users how they would make a specific change. Wobbrock *et al.* showed the change occurring, and then asked the users what command they would issue to cause that result. Showing the response before asking for the gesture removes the expectation that the system will react.

Unfortunately, showing the response of the system may also act as a cue to the user that suggests a specific solution for gesture selection. For example, if the system shows a square formation being formed by multiple robots moving directly to the closest point on the square to their starting location, the paths shown are multiple direct motions. If instead, the system shows the robots forming a chain that snakes around the perimeter of the square before coming to a stop, the path shown is forming the snake, and then traversing the perimeter. The motion in the first case may suggest that the user make individual gestures to position each robot, while the gesture suggested in the second case might be more like dragging a lasso around the group and then a line from the group around the perimeter of the box. Even for simple cases like moving to a target area, if the robots are shown moving along a path, it might discourage the user from using waypoints instead of dragging a path.

	1	10	100	1000	Unknown
Move to area A	x	x	x	x	x
Move to area A with a wall	x	x	x	x	x
Stop the robots	x	x	x	x	x
Divide around an obstacle		x	x	x	x
Orange to B, red to A	x	x	x	x	x
Orange to A, red to B	x	x	x	x	x
Orange to A, red to B (mixed)	x	x	x	x	x
Divide group	x	x	x	x	x
Merge groups		x	x	x	x
Form a line		x	x	x	x
Form a square		x	x	x	x
Move the crate to area A	x	x	x	x	x
Move the crate to area A (dispersed)	x	x	x	x	x
Mark defective robot	x	x	x	x	
Remove defective robot	x	x	x	x	
Patrol the screen border	x	x	x	x	x
Patrol area A	x	x	x	x	x
Disperse over screen	x	x	x	x	x

Table 4.1: User tasks per condition.

Experiment Conditions

Each user was assigned to one of five conditions, varying by how many robots were in each condition. The conditions consisted of 1-2 robots, 10 robots, 100 robots, 1000 robots, or an unknown number of robots. In the unknown number condition, the area the robots were present in was represented by a cloud. For each condition, the user was requested to perform a sequence of tasks. The exact number of tasks varied between conditions due to some tasks not making sense with the number of robots involved, as shown in Table 4.1.

The individual robot case is lacking the tasks that do not make sense for a single robot. A single robot cannot, for example, divide around an obstacle or form a square. The “Merge groups” task was left out of the single robot case because of the potential for confusion when referring to a single robot as a group.

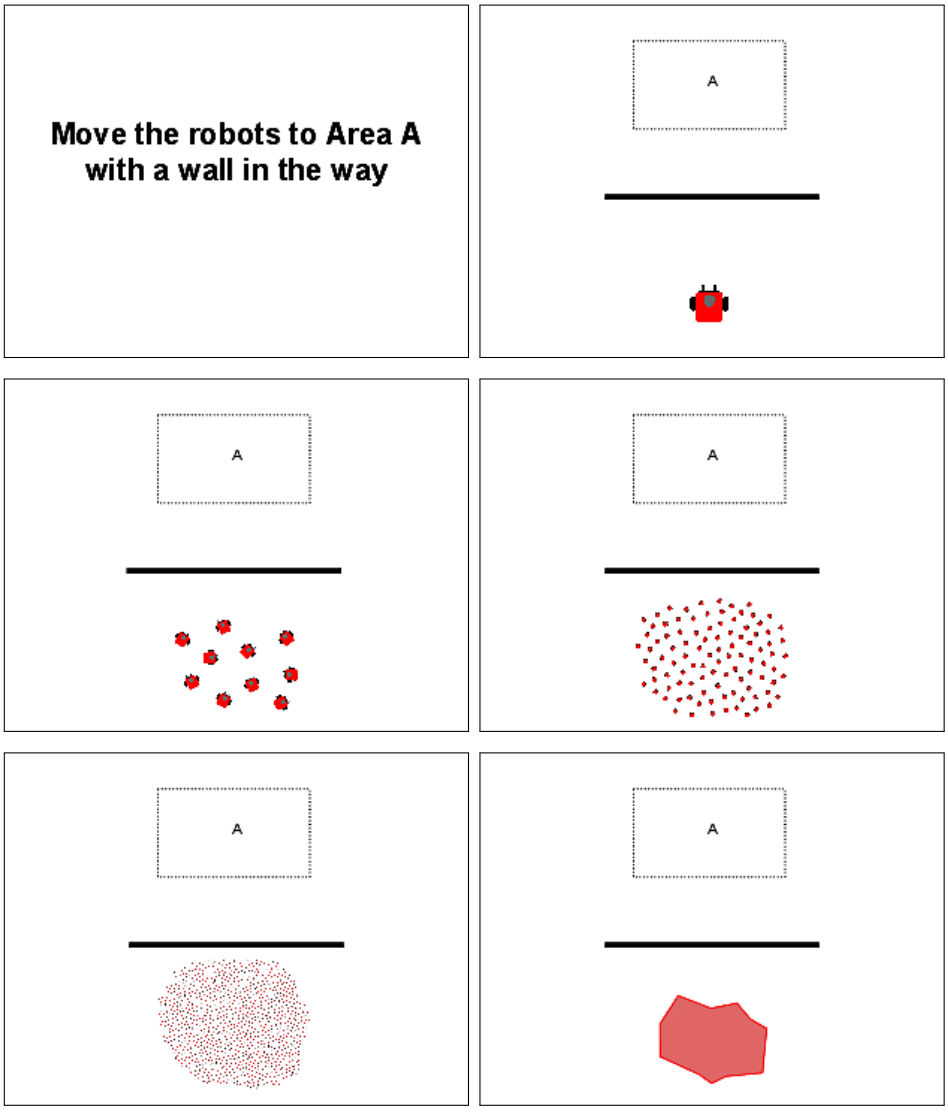


Figure 4.2: Instructional slide and situations for moving around the wall to area A, in each condition.

The unknown number of robots condition has the same tasks as the 10, 100, and 1000 robot cases, except for the "Mark defective robot" and "Remove defective robot" task. Without UI elements that represent individual robots, the user cannot take any actions that refer to a specific robot.

Participant Demographics

Participants were recruited through fliers distributed on campus and in the surrounding city. The recruitment process and experiment were approved by the UMass Lowell institutional review board. The experiment had 50 participants, 10 in each condition. 28 of the participants were male, 22 were female. The average age of participants was 22.1 years, with a standard deviation of 3.16.

These demographics are representative of the location that the study was performed, the campus of an American college. It has been suggested that research in psychology focuses too much on a population that is WEIRD (Western, Educated, Industrialized, Rich, and Democratic), and that the results of such studies may not generalize beyond that population [Arnett, 2008]. However, for the purposes of this study, particularly assessing the influence of smartphone use on expectations of user interface gestures, it is useful to have a population with significant experience using smartphones, which are a product of both rich and industrialized societies. It is not proposed that the results of this work generalize to humanity as a whole.

Analysis

User gestures were coded using a methodology adopted from the social sciences, Grounded Theory [Glaser and Strauss, 2017]. Grounded Theory is an iterative process, where the data are first coded at a very fine-grained level, and then the

resulting coded elements are compared to each other to try to determine their qualities, similarities, and differences. Codes can be consolidated or divided until repeated passes of coding and comparison no longer alter the emerging structure of the coding scheme. During each iteration of coding and comparison, the coder makes memos as well, describing the links they see between related coded elements and higher-level abstractions that relate the elements. These memos are eventually written up as a social scientific theory that is believed to be grounded in the data because it arises from the coding process.

Initial Coding Pass

The inital pass used open coding, where the "codes" were essentially free-form text entry. Rough counts of the open codes for the first 10 participants indicated that ten of the codes covered 81% of the 580 total coded events. The ten most heavily used codes are, in order of occurrence: drag, tap, voice command, box select, 2 finger drag, double-tap, lasso, tap and hold, 2 handed drag, reverse pinch, and parallel hands. This coding pass indicated that a majority of the user actions could be coded as some form of drag, some form of tap, box or lasso selection, pinch, and parallel hands.

The most common code was "drag", which accounts for 37.58% of the rough coding, or 42.07% if all forms of drag in the top ten codes are considered. "Drag" is when the user places one finger down, moves it to another location, and raises it again. Two finger drag is the same, only with two fingers on the same hand placed on the screen rather than one. Two-handed drag is single-finger drag, but executed with both hands at the same time.

The second most common code was "tap", with 20.34% of the rough coding, or 25.34% if tap, double-tap, and tap and hold are all considered. Tap is

when the user places a finger down and then very quickly raises it again. Double-tap is two taps in the same location in quick succession. Tap and hold is when the user places their finger on the screen and leaves it in one place for more than a second before raising it.

Box select consists of a diagonal (relative to the screen edges) drag gesture over the robots or another object on screen, with the intent to select everything within the box whose diagonal is represented by the drag. Lasso select is a drag that ends near where it began, forming a loop, with the intent to select everything inside the loop.

Pinch and reverse pinch are essentially two-hand drag or two-fingered drag but with the hands or fingers moving towards (pinch) or away (reverse pinch) from each other. This gesture is common for zooming in multitouch user interfaces on smartphones.

Voice command was used to code when a user spoke a command out loud, rather than using gestures. The high incidence of voice commands (7.07% of all codes) in the first ten users can be attributed to a single user who issued commands almost exclusively through voice. The final gesture, "parallel hands" is placement of the hands, palms facing each other, over some area of the screen. This gesture was used many times by the same user who issued voice commands, to indicate where the robots should form a line. Because parallel hands only accounted for 0.69% of the gestures, it was left out of the development of the coding application for the second stage of coding.

Second Coding Pass

To facilitate coding in the second pass, an application was developed to record codes. The application has coding functions for the six most common gestures:

drag, tap, voice command, box select, lasso select, and pinch. It also includes coding functions for user interface elements described by the user, such as buttons or menus, a function to code user gestures not covered by the six most common gestures, and a function for coders to enter free-form text memos.

A second pass of coding of the first ten user recordings was performed by two coders. Because the coders were responsible for deciding which user actions to code, as well as how to code them, it is possible for one coder to miss a gesture that another coder codes, or to split a single gesture into two coded units instead of one. For example, one coder initially split spoken commands at conjunctions such as "and", resulting in two units both coded as voice commands, while the other coder coded the entire sentence as a single unit. This leads to the possibility that for a single task, the coders will produce different length lists of coded units.

Cohen's κ is a measure of inter-coder reliability for categorical items, but it assumes that both coders are coding the same number of codable units [Cohen, 1960]. In order to calculate inter-coder reliability in the presence of potentially missing data, the shorter of the two lists of codes for each task was padded with a code for "no data". The codes were then aligned chronologically, with pairs consisting of an item from each list such that the total time error within the task was minimized. As a result, the alignment process created pairs of a valid code and the "no data" code for the codes in the longer list that did not have a good chronological match in the shorter list. This was based on the assumption that the source of the error is one coder missing an event that did occur, rather than the other coder coding an event that did not occur.

The initial pass of coding got poor inter-coder reliability, with the first 10 participants having an average Cohen's κ of 0.422. Cohen's κ of over 0.75 is excellent agreement, 0.4-0.75 is fair, and below 0.4 is poor, so the average κ was

barely above the cutoff for fair agreement [Fleiss, Levin, Paik, Shewart, and Wilks, 2003].

Analysis of the data, particularly with confusion matrices, showed several problems. The simplest was training error in the training of the coders. One coder had not been instructed that the "UI" code existed, and so had coded user interface widgets as "tap" events. This was immediately apparent in the confusion matrix as a very high confusion between UI and tap events. The other main source of confusion was lasso and box selection actions being coded as drag events, and vice versa. The actual action performed in a lasso is a drag, in that the user puts their finger down and drags it in a circle around something before lifting it again, but it is intended as a selection of the things inside the circle rather than e.g. drawing a circle shape.

In order to reduce these errors, and ensure consistency in training the coders, the descriptions for each code were written up in discussion with the coder, so that questions about interpretations could be answered in advance and recorded. The codes were box selection, lasso selection, drag, tap, pinch, ui widgets, voice command, and other. The coding definitions, as presented to the coders, are documented in Appendix A. Another coder was trained with the code description document, and coded the first 5 participants. This coder obtained an average Cohen's κ of 0.794 with one of the original coders. These values indicate a very high level of agreement, so the remaining videos were coded by these two coders, using the description of the codes from the code description document.

The resulting data set had 3,256 individual gestures coded. For analysis, drags that were used to draw something on the screen were separated from non-drawing drags. Taps were separated into single, double, and triple taps, plus tap and hold. Pinch was separated into pinch (where the contact points move towards

Gesture	Count	Percent
Drag	1084	33.29
Draw	761	23.37
Tap	514	15.78
Other	189	5.80
Lasso	184	5.65
Box Select	113	3.47
UI	112	3.43
Double tap	95	2.91
Hold	66	2.02
Reverse Pinch	49	1.50
Voice	44	1.35
Pinch	26	0.79
Triple Tap	19	0.58

Table 4.2: Gestures used by experiment participants, by count and as a percentage of the total gestures used. This table includes example gestures in the counts, as defined in appendix A.

each other) and reverse pinch (where the contact points move apart). All of these divisions were based on modification flags recorded by the coders during the coding process.

Selection Gestures

One area in which the gestures were expected to change between conditions is the use of selection gestures. The intuition behind this expectation is that for small numbers of robots, there is no need for a gesture that selects groups, as the user can interact directly with each robot. Similarly, in the case where no individual robots are displayed, the use of group selection gestures would be minimal, as the group is presented as a single cloud.

Users performed selection of robots in groups by box select, lasso, and UI interactions. These selections were counted by condition by counting lasso or box events that had a robot or robots as their targets. Single selections of robots

Condition	Box Select	Lasso	Tap	Total (condition)
Unknown	0	1	43	44
One	0	3	78	81
Ten	26	105	140	271
Hundred	68	53	35	156
Thousand	18	16	22	56
TOTAL	112	178	318	608

Table 4.3: Per-condition total use of selections

were performed by tapping on the robot. Similarly to the count of group selections, tap events were counted where the object of the tap event was a robot or robots.

The ten robot case has the most selection gestures for either group or single selection. As was expected, the unknown number and single robot cases have very low counts of group selections. Interestingly, the thousand robot case also has relatively low group and single selection use, especially compared to the ten robot case.

To determine if these differences were statistically significant, the count of each user's gestures per task were normalized by dividing by the total number of gestures that user used to perform the task. Normalizing in this manner converts raw gesture counts to a proportion of the total gestures used on that task, preventing more verbose users from dominating the analysis.

The tasks checked for statistically significant differences between conditions were those that all conditions had in common. These nine common tasks are 'move crate', 'divide by color (cross)', 'divide by color', 'move to a', 'move to a (wall)', 'patrol a', 'patrol screen', 'split', and 'stop'. Across all the common tasks the proportions of each gesture were collected per gesture, resulting in, for each gesture, 5 lists, one for each condition. Each list consists of 90 entries, 10 users times 9 common tasks. Each list entry consists of the proportion of that gesture the user used to complete the task. ANOVAs were performed for each pair of conditions.

	Unknown	One	Ten	Hundred	Thousand
Unknown		0.5622	0.1772	**0.0006**	**0.0193**
One			**0.0389**	**0.0030**	**0.0015**
Ten				0.1089	0.2719
Hundred					0.5457
Thousand					

Table 4.4: P-values for the use of tap as select between conditions

For uses of tap as selection across the common tasks, the unknown number condition differs from the hundred (F=7.6964, p=0.0061) and thousand (F=5.5772, p=0.0193) robot conditions. The one robot condition differs from the ten (F=4.3299, p=0.0389), hundred (F=13.5889, p=0.0003), and thousand (F=10.4274, p=0.0015) robot conditions. These differences are summarized in Table 4.4.

For uses of group selection across the common tasks, the unknown number condition differs from the ten (F=47.635915, p <0.0001), hundred (F=60.435124, p <0.0001) and thousand (F=10.2346959, p=0.0016) robot conditions. The one robot condition differs from the ten (F=47.6359, p <0.0001), hundred (F=60.4351, p <0.0001), and thousand (F=10.2347, p=0.0016) robot conditions. The ten robot condition differs from the thousand robot condition (F=19.0110, p <0.0001). The hundred robot condition differs from the thousand robot condition (F=30.2373, p <0.0001). The identical F and p values for the one and unknown robot conditions and the various conditions that they differ from are because for the one and unknown robot conditions, no group selection gestures were used in the common tasks. These differences are summarized in Table 4.5.

Examining the non-normalized counts of taps as selections and group selections across both the common tasks and all tasks indicates that heaviest use of tap selection is in the ten robot case. Heaviest use of group selection is in the hundred robot case, but the ten robot case only lags it by two uses. The second heaviest for group selection is the hundred robot case, while the second heaviest for

	Unknown	One	Ten	Hundred	Thousand
Unknown		*	**0.0000**	**0.0000**	**0.0016**
One			**0.0000**	**0.0000**	**0.0016**
Ten				0.1808	**0.0000**
Hundred					**0.0000**
Thousand					

Table 4.5: P-values for the use of group selections between conditions. The ANOVA between the unknown case and the single robot case was not computable, as no group selections were used for the unknown case or the single robot case for the common tasks.

	Common Tasks	All Tasks
Unknown	0	1
One	0	3
Ten	55	107
Hundred	60	109
Thousand	15	28

Table 4.6: Counts of group selection gestures in the common tasks and all tasks.

	Common Tasks	All Tasks
Unknown	15	29
One	38	53
Ten	45	108
Hundred	4	29
Thousand	7	18

Table 4.7: Counts of tap selection gestures in the common tasks and all tasks.

	Sequence Count	Mean Length	Standard Deviation
Unknown	28	1.0357	0.1856
One	42	1.2619	0.5798
Ten	72	1.5000	1.6499
Hundred	23	1.2609	0.4391
Thousand	17	1.0588	0.2353

Table 4.8: Lengths of sequences of taps within conditions.

tap selection is the single robot case. This seems to indicate that with single robots, users prefer tap selection, with one hundred robots they prefer group selection, but with ten robots, either selection method could be viewed as appropriate.

It was surmised that users may have been performing multiple tap selections in a row to select robots in the ten robot case, but that a 10-robot tap selection would be condensed by the normalization into the same proportion of the overall gestures as a single-robot tap selection. To check if users were performing multiple taps in a row as selections, the lengths of sequences of tap actions were checked across all cases. As seen in Table 4.8, while the ten robot case does have a much higher count of sequences of taps, the mean is higher than the other cases, and the standard deviation is much higher. On examining the data, there was exactly one case of a 10-tap sequence in the ten robot case. While it is possible that a user in the ten-robot case might perform a selection by tapping each of the ten robots, it is not a common occurrence.

Multi-hand Gestures

Multi-hand gestures fall into two different groups. The first group is gestures where each hand is performing part of a single gesture. Generally, these were pinches and reverse pinches with one finger on each hand. The other group of multi-hand gestures consists of a simultaneous pair of different gestures, one performed with

Gesture Pair	Count
Drag and Drag (Different Objects)	26
Pinch	22
Reverse Pinch	7
Tap and Tap	6
Drag and Ttap	6
Drag (Same Objects)	5
Other and Other	5
Box Select and Box Select	3
Box Select and Tap	1
Box Select and UI	1
TOTAL	82

Table 4.9: Two handed gesture pairs. Note that the total is lower than the actual count of total gestures, since it counts e.g. two simultaneous drag actions as a single two-handed drag action.

each hand. For purposes of coding, dragging one object to one place with two fingers was coded as a single drag with two fingers, while dragging two objects to two different places was coded as two single-fingered drags at the same time.

There were 82 instances of two-handed gesturing, which makes up 2.519% of the gestures used. Fifty percent of the users performed at least one two-handed gesture. This is fewer than in some previous research, but more than might be expected if users were generalizing from single-point mouse interaction [Micire, 2010; Epps, Lichman, and Wu, 2006]. Epps *et al.* required users to use a single hand for over half of the gestures in their study, and they point out that their use of a Windows desktop as the working environment of the study may have influenced people towards a single-point interaction style.

Most of the two-handed gestures were simultaneous drags of two different objects. Simultaneous drags and two handed pinches or reverse pinches account for 58.5% of the two-handed gestures.

Due to the prevalence of smartphone adoption, it is no longer simple to determine if smartphone use contributes to the use of pinch and reverse pinch

Task	2-Handed Gesture Count
Merge	14
Divide	12
Square	12
Divide Color 2	9
Patrol Screen	9
Split	6
Divide Color 1	5
Divide Color Mix	4
Move Wall	4
Line	2
Move a	2
Crate	1
Disperse	1
Stop	1

Table 4.10: Use of two-handed gestures by task.

gestures, because there are almost no smartphone non-users in the population of this study. Out of 50 subjects, 46 (92%) reported using an Android or iPhone smartphone daily. Only one user did not report having used an Android or iPhone at least moderately, and stated that they did not own any touchscreen devices. Despite not having any touchscreens, even this user made two-handed gestures.

Influence of Video Games

Previous research had indicated that users of Real Time Strategy (RTS) games would be predisposed to the use of box selection gestures, because many RTS games use box selection to select areas of the screen or units to control [Micire, 2010]. Nine users reported playing RTS games, including Age of Empires, League of Legends, Supreme Commander 2, Civilization, and Europa Universalis. The count of per-task box selection gestures was collected, and divided between users who had played RTS games and those who had not. RTS users made 51 box select gestures, while non-RTS users made 49. While these two totals are quite close, bear in mind that

RTS players are less than one fifth of the total population, but used over one half of the box selection gestures. The mean per-task use of box selection among RTS users is 0.3828 (std. dev. 0.7408), and the mean use of box selection among non-RTS users is 0.0748 (std. dev. 0.3392). ANOVA results indicate that the difference is statistically significant ($F=53.8527$, $p<0.0001$). These results persist even if all games are considered, rather than only RTS games. Thirty-six (72%) of the users reported playing video games. Non-gamers made 10 box selection gestures, while gamers made 90. The mean per-task use of box selection is 0.0418 for non-gamers, and 0.1576 for gamers ($F=11.3061$, $p=0.0008$).

The use of user interface elements, such as menus or buttons on the screen, was also much higher in RTS gamers than non-RTS users. RTS-playing users made 74 UI gestures, while non-RTS users made 20. The average per-task use of UI gestures was 0.5781 among RTS users (std. dev. 1.8608) and 0.02932 (std. dev. 0.1853) among non-RTS users. ANOVA results indicate that the difference is significant ($F=56.2048$, $p<0.0001$). Use of UI elements extends more broadly to gamers versus non-gamers as well. If all users who reported playing games are compared to users who reported not playing games, 90 of the user interface gestures were performed by gamers, and 4 were performed by non-gamers. The per-task average use of UI elements by gamers was 0.1576, and the per-task average use of UI elements by non-gamers was 0.0167 ($F=5.4502$, $p=0.0167$).

Lasso select does not display a relationship between use of the lasso select gesture and playing video games. The per-task mean use of lasso selection for gamers and non-gamers are 0.1821 and 0.1841, respectively, and the difference is not statistically significant ($F = 0.3885$, $p = 0.5332$). Non-gamers made 44 lasso selections, while gamers made 104. Non-gamers composed 28% of the population, and made 29.73% of the lasso selection gestures, further indicating that the use of

lasso selection was equally likely between gamers and non-gamers.

While previous work was able to determine that pinch gestures were used by smartphone users more than smartphone non-users, this study is unable to make such a distinction due to the absence of smartphone non-users in the population. The same work also determined that RTS gamers did not use pinch gestures. It was surmised that since almost the entire population of this study used smartphones, the use of pinch gestures may have become widespread, and would extend to RTS players. However, this is not the case. Out of 51 non-example pinch or reverse pinch gestures, all of them were made by users who do not play RTS games. RTS players made no pinch gestures. This difference is statistically significant ($F = 14.5905$ $p = 0.0001$).

Because of the differences in use of box selection, pinch gestures, and UI widget interactions, it appears that previous user interface experience not only biases users to expect specific types of interaction, but also to expect the absence of other types. In the case of RTS gamers, they use an RTS-like interaction style, with box selection and UI widgets, but they also exclude interactions absent from RTS games, such as pinch. If an interface were designed to query new users as to their previous UI experiences and customize the available interaction methods to anticipate the user's biases, it may be useful to disable some functionality, in addition to adding likely expected functions. As a result, the user will not be confused or surprised by accidentally triggering interaction methods that they do not expect to exist.

It is also likely that the top-down view and multi-unit control in this experiment is similar enough to the interfaces presented by RTS games that RTS users recognized the similarity and so treated it as an RTS game. This idea was confirmed by a few users, who referred to the interface as being "like Starcraft".

Exploiting these similarities for robot user interfaces may make interface design easier, because intentionally emulating a specific style of game will cue the user to use certain interaction styles. For example, a single robot user interface that displays a full-screen first-person view from the robot would cue the user that the use of WASD keys to move the robot and mouse motion to pan/tilt the view is the control scheme for the system, because that is a common control scheme for First Person Shooter (FPS) video games. Extending this metaphor, the use of mouse clicks would tell the robot to interact with things in the environment, since that is the usual "shoot/interact" command in FPS games. Unfortunately, this sort of emulation and leveraging of previous game experience does not provide any design heuristics for interfaces that would give similar cues to non-gamers.

This cueing effect is also important to recognize when considering the results of this study. While the study does allow the development of a set of user gestures that matches the majority of the user-chosen gestures, those gestures were selected to operate with a graphic presentation of the swarm robots, in a top-down view, on a multitouch surface. It is possible that some other interaction modality would ultimately be more effective, or some other set of gestures would be more easily discovered and learned by naive users. This set of gestures should not be understood to be the best or most general set, but the one that users felt best allowed them to control the given presentation.

Influence of Operating Systems

For the disperse task, 12 users made a gesture that was similar across multiple users, consisting of placing four or five fingertips on the same hand together on the screen, and spreading them away from each other. Because a single robot cannot disperse, this task was only presented to users in the unknown, 10, 100, and 1000 robot

cases, so 40 users saw this task. Several users compared the multi-finger scatter to a gesture to show the desktop or all the windows on a Macintosh computer, and Apple's multitouch help indicates that spreading the thumb away from three fingers on the touchpad of a Macintosh laptop will have this effect [Apple Corp., 2017].

Unfortunately, the possibility that a desktop OS might predispose users to a certain style of multitouch interaction was not considered, as desktops have only recently begun to integrate multitouch interaction. As a result, the post-test survey did not ask users what OS they were familiar with. Of the 12 scatter gesture users, 8 reported familiarity with other Apple iOS devices, but this should not be taken to imply that they were familiar with the Mac OS multitouch gestures.

Use of Voice Commands

Twenty percent of the users (10 users) used voice commands. Only one user used voice commands for all of their interactions. Over all of the tasks, voice commands were used for the formation tasks, 'line' (5 commands) and 'square' (4 commands), more than any other task. This is likely due to a bias inadvertently created by colloquial use of the verb "tell" in the instructional slides for the formation tasks. The instructional slides read "Tell the robots to form a line/square", and some users read this to mean that they should speak the words "Form a line" or "Form a square", addressed to the robots.

The 'stop' task also was performed with a voice command by three users. Users indicated that even if they hadn't otherwise been using voice commands, they would use voice commands for the task of stopping the robots. Users stated that because this task assumes the robots are already moving towards a goal, attempts to interact with all of the robots by using gestures while the robots are moving could be difficult.

The 'divide color mix' task was also performed by voice command by three users. This task requires some form of selection by color to separate robots that are in a mixed group, so users would assume that robots knew their color, and address them with commands such as "red robots, unite".

Overall, the 20% use of voice commands is much higher than the 1.3% observed in a previous study [Micire, 2010]. While that study took place as smartphones were becoming popular (and observed effects related to smartphone experience), this study was performed after the introduction of Google Now/Assistant (2012/2016), Microsoft Cortana (2014), Amazon Alexa (2014), and Apple Siri (2011). Aside from Alexa, these voice assistants are all accessed through smartphones. All of the users of voice commands reported high familiarity with Android or IPhone smartphones, but the survey did not ask if they used the voice assistant feature of their phones. As a result, no conclusion can be drawn about the influence of voice assistant technology on user interface expectations from this study, but it may be a topic of interest for further research.

Use of User Interface Widgets

"UI Widgets", in this case, are interactions with the experiment system where the user referred to or expressed a desire for user interface elements such as buttons, menus, or similar controls. Not all users expressed a desire for UI widgets. Of 50 users, 13 used UI widgets. As discussed previously, the desire for UI widgets was higher among gamers, who likely had previous experience with them in games with a similar interface to the experiment tasks.

UI widgets were not used in all tasks. However, the only tasks that did not have at least one user suggest a UI element are the "move" and "move with a

Task	Interactions	UI users
Mark	9	7
Patrol A	10	6
Patrol Screen	12	6
Remove	8	6
Divide Color Mix	7	4
Split	4	4
Square	23	4
Crate	8	3
Crate Dispersed	3	3
Disperse	5	3
Line	3	3
Stop	3	2
Merge	13	1
Divide	2	1
Divide Color	1	1
Divide Color 2	1	1

Table 4.11: Counts of UI widget interactions and of users requesting them, per task.

wall in the way" tasks. Table 4.11 is sorted by how many users expressed a desire to use UI widgets, as this may indicate which of the gestures seemed most difficult to do with a gesture and instead should be done via a more conventional UI.

User Strategies

The term "gestures" in this work generally refers to a single interaction with the multitouch interface device, from when the user's hands contact the screen to when they leave the screen. Some tasks could be performed with a single gesture. For example, many users in the single robot case would perform the "move to A" task by placing a finger down on the robot, dragging the finger to area A, and then lifting their finger. However, many tasks were not performed with a single gesture. For example, to move one group of robots to area A and one group to area B, users would frequently select the first group with a gesture, move it with a second gesture,

select the other group with another gesture, and move it with a fourth gesture. This sequence of selection, move, selection, move is a strategy for performing the task.

User Strategies for Formations

It was observed during the "form a line" and "form a square" tasks that there were some gestures used that were ambiguous with gestures that had been previously used. In the line formation task, many users selected robots and drew a line, or drew a line starting from the robots. This is the same sequence of gestures used for simple movement of the robots. Similarly, in the square formation task, many users simply drew a square. If the square overlapped the robots, this gesture could also be interpreted as a square-shaped lasso selection of the robots within the square. As presented in the experiment, this overlap was extremely likely, as the robots in the square formation task were evenly distributed over the screen, and so there was very little space to draw a square in that would not include at least one robot. In order to determine how these strategies they could be disambiguated from other gestures, the sequences of gestures used by each user were examined and grouped by the overall strategy employed.

For the square formation task, Table 4.12 shows the counts of each strategy. "Draw Square" refers to the user simply drawing a square shape on the screen. "Single Moves" means that the user moved individual robots, usually with drag or select and drag gestures, until each robot was positioned on the border of a square region. This strategy was used by 5 users in the 10-robot case and one user in the unknown case, who treated the corners of the cloud depiction of the swarm as individual points to be moved. In cases with more robots, single moves become extremely tedious to perform. "Select and Draw" means the user selected the robots and then drew the square shape. "Draw Parts" was the strategy of

Strategy	Count
Draw Square	10
Single Moves	6
Select & Draw	5
Draw Parts	5
Voice & Draw	3
UI	3
Hold & Draw	2
Stretch	2
Screen Edge	1
Voice	1
Writing	1

Table 4.12: Strategies used to form robots into a square formation.

drawing individual parts of the square, such as lines or corners, rather than the whole square in one gesture. "Voice" and "Voice & Draw" mean that the user commanded the robots to form a square, and in the latter case, drew a square to indicate where the square should be formed. "UI" refers to the use of UI widgets, such as a "form square" or "draw formation" button, possibly prior to indicating a location for the action to occur in. "Stretch" strategies are where the user treated the group of robots as an elastic structure, and pushed or pulled it into a square shape. The "Screen Edge" strategy was used by having the robots press up against the edges of the visible area of the screen, resulting in lines at each edge of the space, and then moving the resulting lines towards the center of the screen to form the square. This strategy would not work in an area without boundaries, or with non-rectilinear edges. "Writing" refers to a user who selected the robots and then wrote the characters "SQ" with their finger on the screen to command them.

The most common gesture used was to simply draw the square in the desired location. However, as mentioned previously, this strategy was ambiguous when the square overlapped the displayed location of the robots.

Table 4.13 shows the strategies used in forming the line formation. As

with the square formation task, the users who used single moves were all in the 10-robot condition. The four users who used single moves all also used single moves when presented with the square formation task. The strategies that differ from the square formation case are "Squeeze", "Voice & Other", "Tap", "Sweep/Push", and "Select & Tap". "Squeeze" means the user used a pinching gesture to squeeze the robots out into a thinner group, as if they were shaping a soft material. Interestingly, this style of haptic interaction with the shape of a robot swarm has been explored as a potential user interface method in other work, for a limited set of tasks [McDonald, Colton, Alder, and Goodrich, 2017]. "Voice & Other" is similar to "Voice & Draw", but rather than drawing the line, the user made a chopping gesture with their hands parallel and slightly separated over the screen to indicate the location. The "Tap" strategy user suggested double-tapping the robots as a preset command to form the line, but did not make contact with the screen to indicate where the line would be formed. The "Sweep/Push" strategy treated the robots as if they were a collection of small objects that could be pushed around the surface, and pushed from the edges of the group to move them into a line. To "Select & Tap", the user selected all of the robots and then tapped a series of locations in a line to indicate the desired position. While they did this, the user held one finger down at the beginning of the line, and indicated that they could just hold the finger down and draw the line, as with the "Hold & Draw" strategy for the square formation task.

Unfortunately, as with the square formation case, the two most popular strategies are also the ones that are ambiguous, in this case with the strategy commonly used to move the robots along a path to a location. In some cases, the location of the line could be used to disambiguate the user intent. If a line starts from the robot group and ends elsewhere, it would be a command to follow the line to its end, whereas a line that starts and ends away from the robot group would be

Strategy	Count
Draw Line	14
Select & Draw	7
Single Moves	5
Voice & Draw	2
Hold & Draw	2
Voice	2
Stretch	2
Squeeze	2
Select & Tap	1
Sweep/Push	1
Screen Edge	1
Voice & Other	1
Tap	1
UI & Draw	1

Table 4.13: Strategies used to form robots into a line.

a command to form a line formation. However, this means that if the user wants the robots to form a line from their current location to another location, they first have to move the robots away from the desired start point of the line.

User Strategies for Manipulation

In this experiment, manipulation refers to two tasks where the user was instructed to move a crate to a designated area of the screen. In one version of the task, the robots were located in the lower left of the screen, and in the other they were dispersed over the screen. Users in the 10, 100, 1000 and unknown number of robots conditions saw both versions of the task. Because a single robot cannot be dispersed over the screen, users in the 1 robot case only saw the version where the robot started in the lower left area of the screen.

"Surround and Move" refers to strategies where the user moved the robots to surround the crate and then move it to the target area. "Move Crate" means that the user did not command the robots at all, but performed gestures on the

Strategy	Count
Surround & Move	18
Move Crate	14
Drag Through	7
UI & Move	3
Surround, No Move	3
Voice	1
Voice & Drag	1
Select & Move	1
Tap Targets	1
Write	1

Table 4.14: Strategies used to move the crate in the non-dispersed condition.

crate, such as dragging from its location to the target area, or tapping the crate and then the target area. These gestures were the same as the gestures that the users would use when commanding robots, so these gestures could be read as more general motion commands, rather than being specific to robot control.

"Drag Through" strategies were gestures where the user selected robots and then dragged a path for them that passed through the crate towards the target area, so that the robots would encounter the crate on the way and push it along. Interestingly, "Drag Through" strategies were more common in the task where the robots were not dispersed, but instead were in the lower left corner of the screen. This seems to imply that when the robots are closely grouped, they are viewed as a single unit that can be dragged through the crate to push it. However, of the seven "Drag Through" users, three were in the one robot condition, three were in the unknown condition, and one was in the thousand robot condition. Because six of the seven users were in conditions where the robot or group are presented as a single visual object, either a single robot or a single cloud, the idea that the swarm is treated as a single unit because of being presented individually but in a small region does not seem very well supported.

"UI & Move" strategies combine some elements of user interface widgets, such as "pick up" or "move object" buttons, combined with movement gestures such as drags to indicate what was to be picked up or where it was to be moved. "Surround and No Move" describes strategies where the users used robot positioning commands to surround the crate, but did not issue commands to move from the crate to area A. It is possible that the users felt that surrounding the crate adequately conveyed to the system the desired action. "Voice" strategies used verbal commands, while "Voice & Drag" supplemented verbal commands with drags to indicate the desired path of the crate. "Select and Move" means that the user selected the robots, then dragged the crate.

"Tap Targets" tapped on the crate, then the robots, then the target area. If user commands are understood to be a grammatical construction, with subjects being commanded to perform an action, e.g. "These robots, go to this location", then this example is syntactically unusual for English, as it would be more like "This object, be moved by these robots, to this location" instead of "These robots, move this object to this location". While examples like this would be possible to automatically "rephrase" under the assumption that the robots are the only element of the system that can act, and all other objects are acted upon, it would be difficult to extend such automated rephrasing to more complex environments. Such rephrasing would also not be justified in the presence of only one counterexample, but may be relevant for cultures with different common orderings of subjects, verbs, and objects. The user who used "Write" drew abbreviations naming the objects. The full command was "C $\rightarrow$ A", indicating that the crate (which was unlabeled, but starts with "C") should be moved to the target area (called "Area A" and labeled with an A).

The strategies used in manipulation did not generally result in the users

Strategy	Count
Surround & Move	15
Move Crate	13
Clear & Drag	3
Drag Through	2
Surround, No Move	2
UI & Move	1
Voice & Drag	1
Voice	1
Select & Move	1
Write	1

Table 4.15: Strategies used to move the crate in the dispersed condition.

creating novel gestures specifically for manipulation tasks. Instead, they used the same sort of gestures that they had been using for the other tasks, in sequences to cause the system to perform the intended task. Most sequences of user gestures followed a subject-verb or subject-verb-object style of command, such as selecting a group, then specifying an action for that group, and possibly an object for that verb to be done to. One user, however, did extend the grammatical structure of the commands in an interesting way. The user stated that if a path were drawn with one finger, it meant the robots should follow that path, but if it was drawn with two fingers, it meant that the robots should follow that path while maintaining their current formation. This use of an "adverb" in the grammatical structure meant that the user could form the robots into a scoop around the crate, and then have them move to area A while staying in the scoop formation that would drag the crate with them.

The strategies used in the dispersed case were the same as those used in the non-dispersed case, except for "Clear & Drag". "Clear & Drag" strategies only occurred in the dispersed condition. For these strategies, the users made gestures to clear the robots out of the path between the crate and the target area, and then

issued movement gestures to the crate. The robots between the crate and target area were perceived as being in the way.

In both the dispersed and non-dispersed case, "Move Crate" and "Surround and Move" were the dominant strategies, accounting for over half of the strategies used. The "Move Crate" strategy does not specify any particular method for the robots to accomplish the task, but surrounding the crate does. In future work, it may be useful to ask the users exactly how they imagine the robots performing the task, as surrounding could be a preliminary step in caging manipulation, or it could be that the users simply assumed that the robots had to be shown the way to the crate before they could do anything with it.

Robot Count in Unknown Number Case

In the condition where the exact number of robots was unknown, and the swarm was depicted as a cloud, participants were asked to say how many robots they felt the cloud represented. Generally, the participant expectation was that the cloud represented tens of robots, but simply averaging the responses would not be useful, as one participant said that it "could be millions". For those that did answer with a single number or range, the answers were 10-20, 10-"hundreds", 12, 7, 10, 2-10, 10-12, 2-"millions', 5-10, 8, and 50. If the endpoints of ranges are treated as answers, with "hundreds" and "millions" set to 500 and 5,000,000 respectively, the median answer is 10, but the mean and standard deviation are not illustrative of the user responses. Rejecting "hundreds" and "millions" as outliers gives a mean of 11.86, with a standard deviation of 11.04, which matches the common intuition that the swarm contains at least 2 members and many have tens, but not likely hundreds, of robots in it.

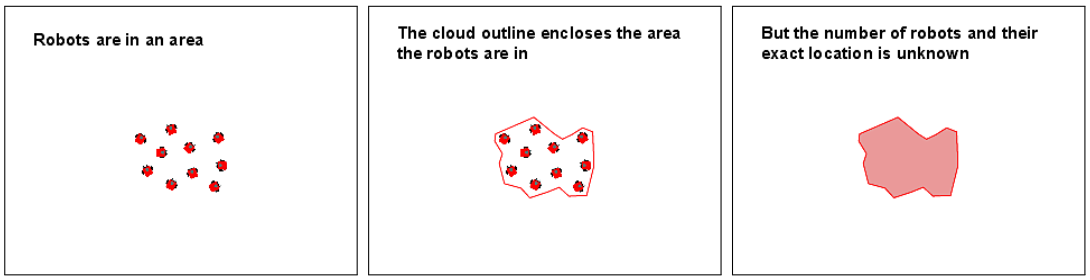


Figure 4.3: Instructional slides for the unknown number of robots condition, showing cloud representation of robot swarm.

Participant comments may shed some light on the source of this intuition. One participant indicated during the study that they interpreted the corners of the polygonal cloud as possible robot locations, and so drew an idea of the scale of the swarm from the number of corners. The cloud has 11 corners, 7 convex and 4 concave, which is consistent with the estimated robot count. Another participant said their estimate was informed by the instructional slides preceding the test, which depicted a number of robots, the outline around them, and the resulting cloud, as shown in Figure 4.3. These slides depicted 10 robots, and so may have biased participants to expect that the cloud had 10 robots in it. However, multiple participants stated that the cloud represented an unknown number, or estimated more or less than 10 robots.

Selection Behavior

At the end of the experiment, users were shown an image of a robot swarm, with a dotted line around it depicting a finger drag path around some of the robots. The users were told that the intent of this gesture was to select robots, and asked to indicate which robots in the image were selected, or in the unknown number of robots case, whether the gesture selected all of the robots or left any out. Users in the single robot case were shown the selection image with ten robots in it, because

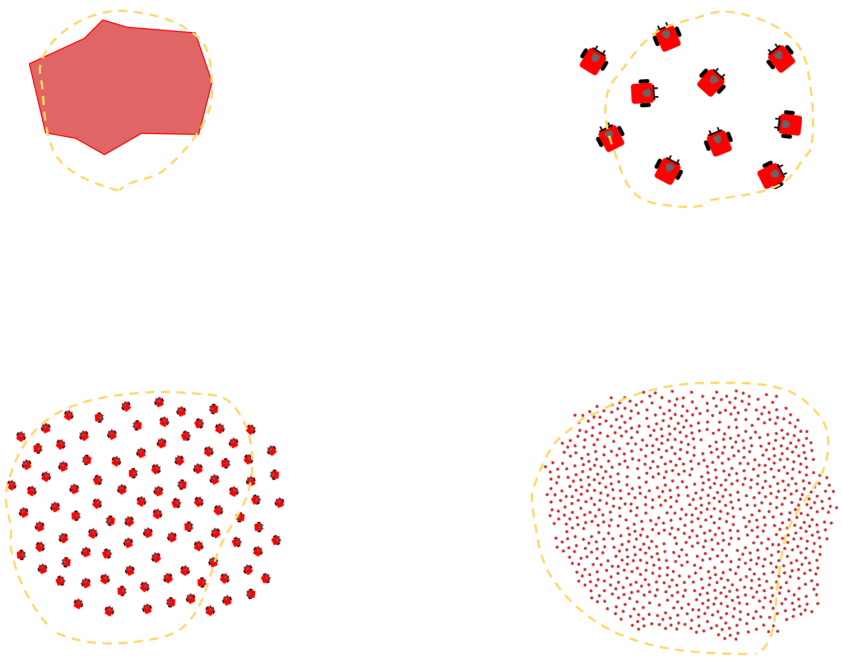


Figure 4.4: Images for selection strategy question.

the with one robot, there are not enough robots to have a selection that may exclude some robots and include others.

In the unknown number of robots case, 7 participants indicated that all the robots were selected, 3 of the participants indicated that some robots were left out, and one participant indicated that whether the selection included all of the robots could depend on the task.

Because the 1 and 10 robot cases saw the same selection image, they are reported together. Twelve participants indicated that robots that were inside the selection or touched by the selection line should be considered selected. Two participants indicated that robots depicted with half or more of the robot inside the circle should be considered selected. One participant indicated that the robot

Condition	Completely inside	Half or more in	Touching line	Other
10	0	2 (10%)	12 (60%)	3 (15%)
100	1 (10%)	5 (50%)	3 (30%)	0
1000	2 (20%)	1 (10%)	7 (70%)	0
Totals	3	8	22	2

Table 4.16: User responses for whether robots on the edge of a selection should be included. The unknown case is omitted because the relationship of individual robots to the line is not visible in that case.

half out of the circle should *not* be selected. One participant indicated that robots on the border should be included or not included in the selection, depending on the task. One user indicated that only the first robot touched should be selected, which is consistent with control strategies that move each robot individually. The remaining three responses were not recorded due to researcher error.

For the hundred robot case, 3 participants indicated that robots inside or touching the line were selected. Three participants said that robots should be mostly inside the line to be counted, with one participant stating that robots should be 80% or more inside the line to count. Two participants indicated that robots should be more than halfway inside the line to be selected. One participant stated that only robots completely inside the line should be counted.

In the thousand robot case, 7 participants indicated that robots inside or touching the line should be selected. Two participants indicated that robots must be completely inside the line to be selected. One participant indicated that robots mostly inside the line should be selected.

Overall, participants generally err on the side of inclusion, rather than exclusion of robots from selection. If a user interface is required to include or exclude ambiguous elements from a selection, it appears that including ambiguous elements will satisfy users. User comments also suggested that ways to amend selections before further commanding the robots would be desirable.

It is interesting to note that two participants, one from the unknown number of robots case, and one from the 10 robot case, said that in cases of ambiguous selection, the system should add or remove robots from the selection based on the task. As the system cannot predict the task it is about to be commanded to perform, it is likely best to err on the side of selecting too many robots, rather than too few.

NUI Metaphor Failure

During this experiment, there were some cases where the users expectations of what was possible in the interface indicated a sort of "metaphor failure" in the user interface. Natural User Interfaces, of which multitouch screens are an example, are supposed to be able to leverage users' understanding of the physical world, and how objects behave in it, to build affordances for objects on the screen. For example, a volume knob can be displayed as an actual knob, and the screen can react appropriately to attempts to rotate the knob. However, people know that the objects on the screen are not knobs, switches, and so on, but pictures of those things, drawn by the computer. As a result, the affordances are mixed. The knob may afford turning, but it also affords dragging around the screen or deletion, which a knob on a real radio does not afford.

In this study, users were tasked with stopping the robots, which had begun to move around a wall to a target area. One user dragged the wall in front of the robots, and another user asked if they could move the wall. The wall was intended to represent an actual wall, which does not afford moving in the physical world, but it was represented in the experiment as a thick black line. It may be that the users would have not attempted to move the wall if it was more clearly

represented as e.g. a stone or brick wall, and so had connotations of excessive weight. On the other hand, the users may have regarded it as what it actually was, an image of a wall on a computer screen, and decided that since images can be moved around the screen, the wall image can too. Attempting to experimentally determine if the representation affects how the user interacts with the wall would require an experiment better designed to examine that affordance, as out of 50 participants, only 2 even mentioned the idea of moving the wall.

Chapter 5

UI Design and Implementation

In order to convert the user gestures collected in the gesture collection experiment into a user interface, a subset of the gestures was selected to support the majority of the user-defined gestures.

In some cases, the user defined gestures were ambiguous, either across users, or with a user across tasks. Where design decisions were made to work around these ambiguities, they are described.

Selection of Gestures for Control of Swarms

For position commands, drag, tap, and "other" would cover 94.8% of the gestures. Unfortunately, the "other" commands are not a single gesture, but include a diverse collection of gestures, such as pushing with the side of the finger or making a picking-up, carrying, and setting-down motion over the screen. As a consequence, attempting to implement all the gestures in the "other" category would add significant complexity to the gesture recognition in order to support gestures that were rarely used. Many of the "other" gestures were also not recognizable by a

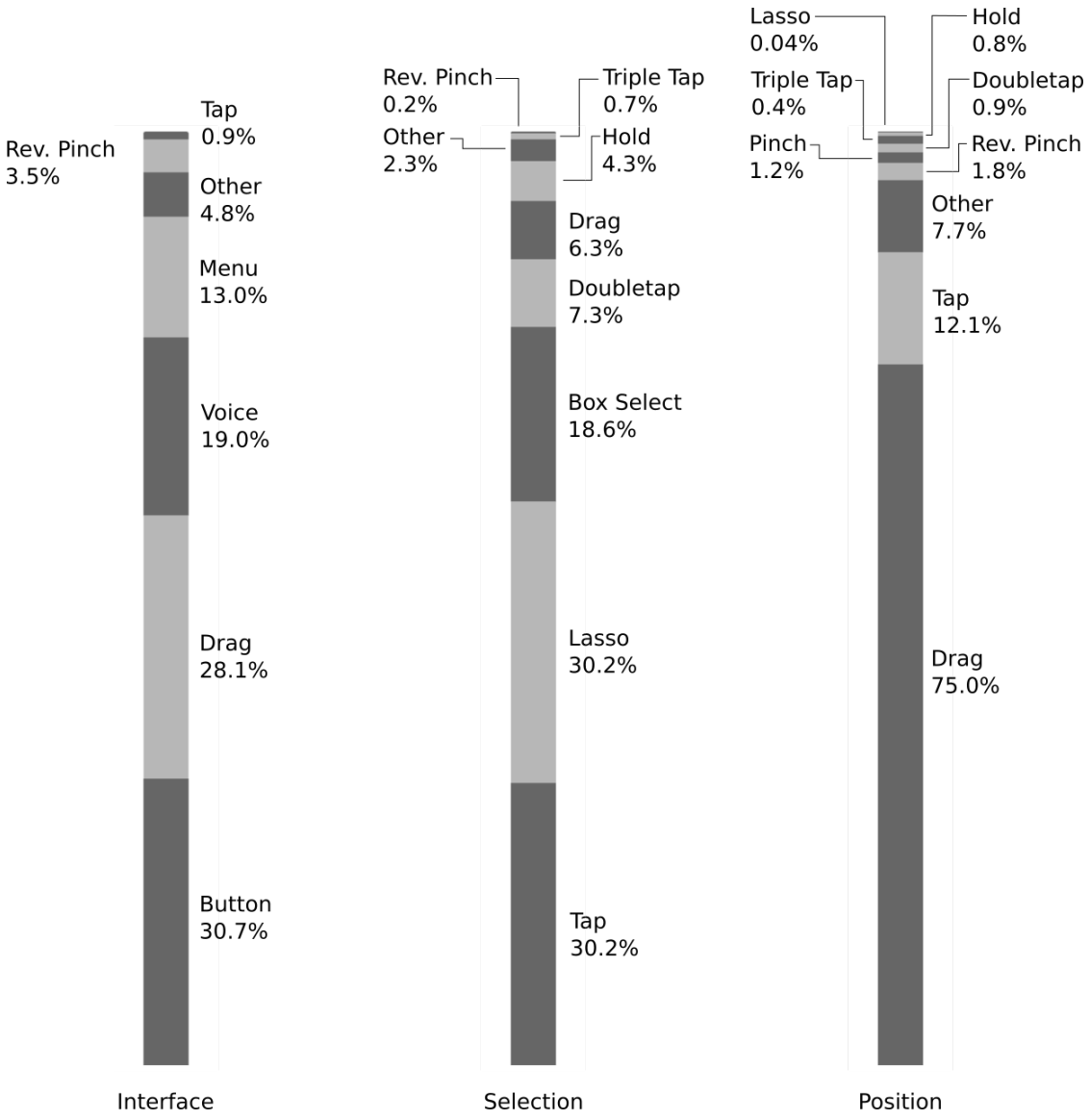


Figure 5.1: UI gestures, selection gestures, and position gestures.

multitouch surface, such as the user dividing the robots by parting their hands as if opening a bag in the air above the surface. Since it does not contact the multitouch surface, such a gesture cannot be recognized without additional hardware. If all forms of tap are considered the same when used as position commands, taps make up 14.2% of the position commands, and together with drag, cover 89.2% of the position commands.

Lasso was used as a position command by one user, who used it to command the robots to disperse in the "disperse" task. The user noted that the direction of the lasso disambiguated it from lasso as selection, so lassoing clockwise would select robots and lassoing counterclockwise would disperse them.

Achieving 90% recognition of the most common UI widgets would include buttons for special functions, handwriting recognition, voice commands, and other menus on the screen (totaling 90.8%). However, a successful menu-based UI would not be composed simply by taking the sum of all of the user interface designs suggested by users, as there would be significant redundancy in the UI commands, and in ways of bringing up the UI for interaction. Instead, the tasks that the users invoked the UI in most often were examined, and the requested UI functions were considered for inclusion.

Both voice commands and handwritten commands together account for 47.1% of the user interface elements used. While the broad categories make up nearly half of the UI elements, the individual uses of the gestures did not display a significant unity within the broad categories. One user used symbols drawn over the swarm as commands, so a circle drawn within the swarm area meant "patrol", and would be followed by an indication of the area to be patrolled. Another user wrote commands, such as writing out "0.5" to indicate that robots should divide in half, "SQ" to indicate that they should form a square, or "C $\rightarrow$ A" to indicate that

the crate (C) should be moved to area A. The majority of the drawn commands were attributable to a single user, but two users drew an 'X' over the defective robot in the "mark defective robot" and "remove defective robot" tasks and two users used an 'X' and an 'S' in the "stop the robots" task.

The selection of a set of handwritten commands that would be usable for a majority of users would be a task at least as difficult as finding a user-defined set of gestures. In addition to the set of English letters, the interface would have to consider symbols such as arrows, and apparently decimal numbers as well. The experiment described in this thesis did not collect sufficient examples of handwritten commands to extrapolate a useful control scheme from them. Because of this lack of information, and the fact that most handwritten commands were proposed by a single user, the resulting interface does not contain support for handwriting recognition and symbol interpretation.

Interestingly, voice commands were similar to both handwriting, in terms of distribution of user choice, and gestures, in terms of syntax. One user used voice commands for all tasks, but ten users also used voice commands for at least once. The distribution of voice commands is like the handwritten commands, in that one user used handwritten commands frequently, but a few other users used a few of them for some tasks. The accidental biasing of users towards use of voice for formation tasks was discussed earlier. In addition, two users stated that they would like to have a vocal stop command for the "stop the robots" task, because it could be issued quickly and without having to accurately touch the moving robots. These users did not issue other voice commands, as the other tasks were not perceived to be as urgent as the "stop the robots" task.

The syntactic similarity between the voice commands and the gesture commands is because the gesture commands frequently take the subject-verb-object

Task	Voice Commands
Line	5
Square	4
Crate (dispersed)	3
Divide color mix	3
Stop	3
Crate	2
Move to A	2
Disperse	2
Merge	2
Split	2
Patrol A	2
Divide	1
Divide color 1	1
Divide color 2	1
Move wall	1
Remove defective robot	1
Mark defective robot	1
patrol screen	1

Table 5.1: Use of voice commands by task. The use of high numbers of voice commands for the formation tasks, line and square, was likely biased by the text of the instructional slides.

ordering of spoken sentences. Some parallelism in the structure of spoken commands and gestures is unsurprising, as in human-to-human communication, vocal expressions and gestures are theorized to arise from the same internal representations [McNeill, 1985]. For example, selecting the robot group indicates a subject, and drawing the path indicates the verb ("go this way"). Objects are optional or implied, as going to a location does not have a clear object that the robots are instructed to act upon. In some cases, the subject is implied as well. For example, some users would make gestures intended to move the robot group as a whole by simply dragging the path, without selecting the subject first. In such a case, the implied subject was all of the robots. This sort of implication is more complex than simply assuming that if no robot is selected, all of the robots are the subject. Some users divided the robots into two groups by drawing a dividing line, and then dragging two paths, one to one side of the screen and the other to the other side of the screen. In this case, the implied subject was the half of the robots on the same side of the dividing line as the drag, but no selection gesture preceded the positioning drag to indicate this.

Reverse pinch was used as an interface command to zoom in or out of the view of the robots. This was done in the "mark defective robot" and "remove defective robot" tasks. The user who made this gesture was in the 1000 robot case, and stated that they wanted to zoom in because the defective robot was a small target and close to other robots. There was no explicit zoom or change of viewpoint task, but users expected the functionality when they had to interact with a small target.

Tap, lasso, box select, doubletap, drag, hold, and "other" cover a total of 91.9% of the selection gestures. As discussed above, the "other" category is not practical to implement. If, instead, all forms of tap are considered the same for

purposes of selection, they comprise 42.5% of the selection gestures, and together with lasso and box select, cover 91.3% of the selections.

Drag as a selection gesture refers to the user placing their finger on one robot, and then dragging it from robot to robot, adding each touched robot to the selection. It is ambiguous with the position drag, where the user places their finger on a robot and then drags a path for the robot to follow, although the distinction could be made by having a path that intersects multiple robots become a selection, rather than a position command. Unfortunately, this attempt at disambiguation would itself become a source of problems if the user attempts to move the entire swarm by placing a finger in the middle of it and dragging to a new location, as it is likely they would intersect more than one robot as their finger leaves the swarm.

Ambiguities in Gesture Commands

As discussed briefly in the previous section, some combinations of gestures selected by the users were ambiguous. This is to be expected, as the users did not know all the tasks in advance, and so might use a gesture in one task that they then felt was better suited to a later task. The users might also not regard all the tasks as having to use a consistent and unambiguous representation for each gesture, or not remember all the gestures they had previously used. However, for automated conversion into programs, it is important that a sequence of gestures have some way to be unambiguously recognized.

In the line formation tasks, some users selected the robots and then drew a line to indicate the location of the line formation. If the line formation started at the same location as the robots, this gesture sequence is the same as the selection and drawing a path sequence that was frequently used to indicate that the robots

should move as a group along the path. If the line formation started elsewhere, it could be disambiguated, because motion along a path almost always started on or near the selected robots. The user drawing a line over a group of robots could be interpreted a number of ways: as a dividing line, as a box selection, as a path for a single robot to follow, as a path for a group of robots to follow, and as the location of a line formation.

Because a number of other, more explicit commands are available for dividing robots into groups, such as moving one group away from the other, the division line interpretation was rejected. A line drawn over robots is treated as a box selection, unless it begins on a single robot. In that case it is interpreted as a path for that robot to follow. For the purposes of this work, beginning "on" a robot was defined as the beginning of the line being within 80 pixels of the location of the robot on the screen. This number of pixels was selected because it is the approximate width of a human finger on the screen used in this work. Obviously, this distance will vary with the resolution of the screen, but the mechanism for obtaining the resolution and physical size of attached screens varies with operating systems, and is outside of the scope of this work, aside from noting that this parameter may vary. The selection of parameters like this for an interface intended for real-world, rather than experimental use, should be guided by understanding of human factors and UI design principles, such as Fitts' Law [Fitts, 1954].

Similarly, some users divided the robots into two groups by drawing a line separating the groups. Typically, this line started outside the group and passed through it, so it could be disambiguated from instructing the robots to move along the line. Unfortunately, it would be difficult to separate it from drawing a line for the robots to move to in order to create a formation.

To command the robots into a square formation, especially in the dis-

persed cases, many users simply drew the square formation over the robots. If a lasso select is also available, some distinction must be made between the lasso select and the square formation when they are both drawn over the robots, as they are both closed forms drawn over the robots. One possible method is to look for peaks in the distance from each point on the perimeter of the gesture to the centroid of the gesture. A square would have four peaks, while a circle would have no obvious peaks. However, this sort of recognition means that a square lasso select, or an arbitrarily-shaped lasso select that happens to have four peaks, would get misinterpreted as a formation gesture, while a command to get into a circular formation would be interpreted as a lasso selection. It would be preferable to be able to make arbitrary formations, and arbitrarily-shaped lasso selections, and have them be disambiguated by some other method. One method that users proposed was to have a "draw formation" button, which would then cause the next form that the user drew to be treated as the perimeter of the formation. Some users also held one finger down at the start of the formation, while tracing the perimeter with the other finger. Using a multi-finger gesture has the advantage of not adding UI elements, but is not easily discoverable by a user examining the UI, and so would require some form of training.

In order to patrol area A or the screen border, users frequently dragged the robots as if issuing a basic motion command. Some users indicated that there would have to be an additional signal to the robots to keep moving on the patrol route, once it was assigned, but there was not an agreement as to what that command should be. Of the users who indicated a need to convey that the swarm keep patrolling, some repeated the patrol route drag multiple times, others ended the gesture with a tap or triple tap, still others used the direction of the patrol route drag (clockwise or counterclockwise) to separate following the path once from

continuously following it.

Some users reused gestures for different purposes, such as tapping a robot to select it, but also tapping a robot to remove it. Because 30.2% of the selection gestures were taps, it was decided that selection, which occurs often, would be done by tapping, while removal of a robot, which is rare, would be performed by a different action.

Implicit Selection

The distinction between interpreting a line drawn over robots as a command for one robot to follow versus a command for multiple robots to follow may depend on the number of robots present. As seen in Section 4.3 the use of selection gestures was low in the 1000 robot case, and much higher in the 10 robot case, implying that with a large robot group, it might make sense to default to using all robots if none are selected. The interface could even change depending on the number of units being commanded, with small groups requiring that all robots be selected before a command can be applied to them, and larger groups defaulting to issuing the command to all robots if some subset has not already been selected. However, care must be taken to avoid learning effects. If the user has previously been exposed to a condition where the system defaults to selecting all robots, they may develop an expectation that other conditions behave the same, and neglect to select robots, resulting in incomplete commands (or commands being applied to no robots). Interestingly, the opposite case, where the user makes a selection of all the robots, but it is not actually required because it is the default behavior, does not result in a surprise for the user. Because the failure mode encouraged by a system that requires explicit selection is less surprising than one that has implicit select-all, the system was developed to require selection to issue a command to multiple robots.

As described above, issuing a command to a single robot may be done without selecting it first, by beginning the desired path on the robot.

Gesture Complexity

The gestures used by the participants in the experiment could be divided between selection and position gestures, within which there was a great deal of agreement, and more complex actions, such as patrol, formations, picking up or moving objects, removing robots, and selection by color.

For selection, if the system accepts taps (of any form), lasso, and box select, 91.3% of the user selection gestures are covered. For position, tapping (again, of any form) to set goals or way points and dragging paths will cover 89.2% of user position gestures.

Because the more complex tasks had a higher variety of gestures that could be used to convey the user intent, as well as some users performing the more complex tasks by repeated selection and position gestures, attempting to handle this variety of gestures leads to two problems with the discoverability of the interface.

First, since each of the gestures was chosen by a smaller number of the users, some training must be performed to inform users who would not have chosen that gesture. This training could be done with a "cheat-sheet" that users could pull up for reference, or an animated introductory tutorial sequence. These options are both less desirable than having the system be, as much as it can be, self-explaining. They separate the training of the user from the user's interaction with the interface, rather than having the interface make clear what can be done.

Second, if a large percentage of the gestures for the more complex tasks were implemented, in order to provide high coverage of the various options that users chose to perform those tasks, there would be a large set of gestures that have

very specific meanings. Increasing the set of gestures that the system recognizes increases the chances for ambiguities, where the same gesture was selected for multiple roles by different users, and for errors, where the system misinterprets one gesture as another. This interferes with discoverability, as the system may react in different ways to what the user felt were the same actions, confounding the user's ability to learn a mapping between their actions and the results that the system produces.

As a result, the complex tasks were assigned to buttons on the user interface. The use of buttons is more discoverable, as the button simply says on it what it does. Buttons do obscure some of the camera view that forms the main part of the user interface, but this problem is not as severe as it might appear. The area the robots operate in is either bounded or unbounded. If it is bounded, it either fits on the screen or does not fit on the screen, and if it is unbounded, it does not fit on the screen. If the area the robots are in does not fit on the screen, the user likely views it by panning or zooming in and out, and so can pan or zoom so that the area covered by the buttons is not an area that they have to interact with to perform a task. If the area the robots are displayed on and interacted with in does fit on the screen, it can be shrunk slightly, and the buttons can be placed so they do not cover the interaction area. In either case, the design of the interface can accommodate a few buttons without seriously harming the user interaction, although the unchecked proliferation of buttons could lead to design difficulties.

Termination of Commands

Some of the potential gesture commands, such as selection of a group of robots followed by tapping waypoints for them to follow, do not have a clear termination,

as the system cannot tell if the user is done tapping waypoints, or simply has not tapped the next one yet. One potential solution to this problem is to attempt to parse the user command once enough time has elapsed since the last gesture received. Using a timeout to commit the command was rejected for two reasons. First, the timeout can result in the system beginning to take an action that the user did not want. If the timeout is too short, it could result in a program being transmitted to the robots before the user is done specifying it. If it is too long, it may never be triggered, and so all of the user's gestures could be gathered into one, potentially untranslatable, program. Second, the timeout is could be invisible. From the user's perspective, this causes the system to appear to begin acting at random, which is undesirable. The timeout could be made visible, using a countdown timer, which may then make the user feel pressured to act.

In order to avoid having the system act prematurely, the system treats a double tap on open space as an "end of command" signifier, as in Micire [2010]. If the hardware used in the system were capable of detecting it, the user placing their hands in their lap or away from the screen could also be used as a signifier that the user has completed entry of their command. Most users moved back from the screen slightly and removed their hands from the volume above it once they were finished issuing their command, and no users were observed to rest their hands on the screen while not issuing commands. However, this behavior may have been a consequence of the particular arrangement of the screen and user seating, so further work would be required to determine if it is a sufficiently robust indication of command completion.

In addition to double taps, there are limited instances where it is safe for the system to assume that the user's command is complete. Because commands are generally of the form subject-verb-object, a new subject begins a new command.

The exception to this rule is tap selections, which some users used to select multiple robots by sequentially tapping on them. Tap selections are accumulated until a gesture other than a tap selection is entered. After a non-tap action is entered, a new selection gesture will begin a new command.

Acceptable Command Sequences

The commands that use buttons are Patrol, Make Formation, Move Object, Remove Robot, and Select Group. The text of the buttons is written as a verb phrase, as the buttons take the place of the verb in the subject-verb-object (SVO) structure. To preserve the SVO ordering, the command parser expects a subject, specified by a selection gesture, a verb, expressed by a button, and then an object, expressed by another gesture.

For Patrol and Make Formation, the "sentence" reads somewhat like "These robots, Patrol/Make Formation, this location/shape". The selection is performed, then the button pressed, then the location or shape is specified.

In the Move Object command, the sentence is more complex, as both an object to be moved and a location to move it to must be specified. The sentence would read as "These robots, Move Object, this object, to here". This structure is somewhat in conflict with the most common user strategy to move the crate, which was to surround the crate with robots and move the robots, and also in conflict with the second most common strategy, which was to move the crate, with no reference to the robots. The decision to differ from these strategies was undertaken for a number of reasons. The first is that moving the crate with no reference to the robots implies that the subject of the sentence is all of the robots. As discussed previously, implicit selection of the entire swarm has a high potential to surprise the

user, especially if the behavior changes across swarm sizes, while requiring explicit selection does not. The second is that unless the system is aware of which objects in the environment are movable and which are not, there is no way to disambiguate a command to surround an object, and then leave it to go to another location, from a command to move the object by surrounding it. This level of knowledge about a novel environment may not be possible to obtain in real-world situations.

The Select Group and Remove Robots buttons are not commands to the robots. They are commands to the system. Select Group selects robots by some common feature (in the user test, common group membership was depicted by color). Remove Robot instructs the system to mark a robot or robots as not to be used, and so they are excluded from having commands issued to them. The system is the implied subject of the sentences "[System], Remove Robot, these robots" and "[System], Select Group, these robots." Implying the subject of a command to the system is acceptable in a way that implying robots as a subject is not, because there is one control system, and so the subject is not ambiguous. The interface can be viewed as similar to a desktop computer, where it is generally assumed that commands invoked through that computer's UI are for that computer to do.

The commands that are not invoked through buttons are the positioning commands, to move single robots or groups. The division between buttons and "pure" gesture commands was made because there was higher agreement between users on the basic movement commands than on the more complex commands. Additionally, some users implemented the more complex commands by sequences of basic motions. This was especially apparent in formation commands with lower counts of robots, where some users issued individual movement commands to each robot, which ended at a location within the formation.

Positioning commands consist of a subject and a verb phrase gesture

that reads as “These robots, go here.” However, there are a number of ways that these can be expressed. The selection can be by any of the selection methods. The position commands can be by tapping a location, or by dragging a path for the robots to follow. However, this raises the possibility that the dragged path is a line or loop over other robots, and so resembles a group selection. There are two possible ways to treat this sequence.

The fact that there is an end-of-command marker, the “period” at the end of each sentence, means that the user could make a sequence of taps on robots, lassos, and box selection gestures, ending with either something that could be a path, a lasso, or a box selection and then the end-of-command marker. The system would then interpret the last lasso or box selection as a path, and so the whole command as a sentence with a compound subject, “These robots and this robot and..., go here”.

The other possibility is having each selection create a new command. If that is the case, it becomes impossible to patrol an area with robots in it, or have one robot move through a group of robots, as those gestures are a lasso or box selection that starts a new command. The exact ambiguity is not between all selections and path commands, but between group selections and path commands. Tap selection of single robots is unambiguous, under the assumption that collisions are bad, and so it is never desirable to command one robot to another robot’s exact location.

This raises the possibility of a third selection method, where single/tap selects can be chained to select multiple robots. If tap selections are allowed to chain, it does create an instance of a selection not terminating an old command and starting a new one, creating an inconsistency with the rule that starting a new selection begins a new command, and so implicitly ends any previous command.

If tap selections are not allowed to chain at all, then tapping multiple robots one after another only selects the last one, and results in a sequence of incomplete programs consisting only of selections. It also eliminates a behavior that some users did show, selecting a small number of robots by tapping on the individual robots. Because it admits very detailed multi-selection and supports a selection style that users chose, tap selections were allowed to chain with each other, but not with other selections.

Having selections end the previous command and start a new one, combined with the existence of lasso selection and box selection, adds some complexity to attempts to send the robots on a path that looks like a lasso or box selection. Determining whether a loop-like gesture or a box-like gesture is intended as a path in the previous command, or the beginning of the next command, depends on whether it is immediately followed by an end-of-command gesture. If the resulting stack of gestures consists of some form of selection, a box or lasso selection, and then an end-of-command, then the box or lasso selection can be replaced by a path gesture, and the program becomes valid. However, if the stack contains some form of selection, a box or lasso selection, and e.g. a path gesture, then the first selection is an erroneous program consisting only of selection, and can be dumped from the stack.

Similar heuristic rewriting of the stack could be extended to remove erroneous waypoints resulting from triple-tapping instead of double-tapping to end gestures, but adding these sorts of heuristics poses something of a threat. There exists some rewriting of the user input that will eventually result in an acceptable sequence of gestures, even if it is a complete replacement of all of the user input. However, at some point, the meaning of the rewritten stack will deviate from what the user intended. While using such a stack of commands will result in the

generation of a program for the robots, it will not be a correct program, from the user's point of view. In such a case, it would be better to have the parsing of the user input fail. The determination of what level of alteration of the user input is acceptable is not the core subject of this research, but is an interesting question.

Simultaneous Actions

Expecting a sentence-like form for commands does not take as much advantage as it of simultaneous actions possibly could. The multitouch surface used in this work can track 20 points at once, and some surfaces can track even more. As a result, the hardware will allow users to, for example, perform selections with both hands at the same time and then draw a path with each hand. Sentences, on the other hand, are generally spoken serially, instead of in parallel, so the analogy to language does not provide a convenient heuristic for determining which subjects go with which verbs. While two selections performed at the same time will overlap, one is almost certain to finish before the other, and so could be interpreted as a selection following another selection. Even if overlapping selections result in two groups both being selected independently, if the user then taps two goal points and ends the command, the result is ambiguous. If the two goals were tapped one after the other, they could each be a goal for one selected group, or a sequence for both selected groups to visit. If the two goals were tapped at the same time, perhaps they are each a goal for one group, but which group should go to which goal is unspecified. While there are heuristics that could be used to guess user intention in these cases, such as always dispatching the closest selected group to a given goal, the system does not have the information to properly guess the user's intent.

However, two-handed gestures accounted for only 2.52% of the gestures

used. As a consequence, the loss of the ability to perform simultaneous gestures, in a single-user context, does not result in a large loss of functionality. In a multi-user context, it would be desirable to have users be able to perform interactions simultaneously, but such an extension would also require additional information to associate contact points on the multitouch surface with particular users.

Abandoning the use of simultaneous gestures also allows the unambiguous use of the scatter gesture for triggering dispersion. If simultaneous gestures are allowed, then performing the scatter gesture over a group of robots is detected by the system as several simultaneous attempts to drag robots and draw paths at the same time. If the gesture is performed over open space, it is detected as 4-5 paths being drawn at the same time. With simultaneous gestures, the command sentence selects some group of robots and sends them multiple paths to possibly follow, with no order in which they should be followed. However, without simultaneous gestures, the multiple contacts can be combined as a single scatter gesture.

Representation Of The Command Language

The command input language can be defined formally, given the constraints above. The Augmented Backus-Naur Form (ABNF) description of the language is given below. Using the scatter gesture for dispersion is indicated in ABNF as 4*5(dragRobot—path) to require that there be at least four and at most five fingers used to make the gesture. An alternate approach would be to have the gesture detection coalesce paths and robot drags that were close enough in space and overlapping in time, and present them as a single “scatter” gesture.

$$\langle \text{sentence} \rangle \models \langle \text{cmd} \rangle \text{endCmd} \tag{5.1}$$

$$\langle \text{cmd} \rangle \models \langle \text{patrol} \rangle \mid \langle \text{makeFormation} \rangle \mid \langle \text{moveObject} \rangle \mid \langle \text{removeRobot} \rangle \mid \langle \text{disperse} \rangle \mid \langle \text{goHere} \rangle \tag{5.2}$$

$$\langle \text{patrol} \rangle \models \langle \text{selection} \rangle \textit{Patrol} \langle \text{path} \rangle \tag{5.3}$$

$$\langle \text{makeFormation} \rangle \models \langle \text{selection} \rangle \textit{Make Formation} \langle \text{path} \rangle \tag{5.4}$$

$$\langle \text{moveObject} \rangle \models \langle \text{selection} \rangle \textit{Move Object} \langle \text{selection} \rangle \langle \text{path} \rangle \tag{5.5}$$

$$\langle \text{removeRobot} \rangle \models \textit{Remove Robot} \langle \text{selection} \rangle \tag{5.6}$$

$$\langle \text{disperse} \rangle \models \langle \text{selection} \rangle \langle \text{scatter} \rangle \tag{5.7}$$

$$\langle \text{scatter} \rangle \models 4 * 5(\text{dragRobot} \mid \text{path}) \tag{5.8}$$

$$\langle \text{goHere} \rangle \models \langle \text{selection} \rangle \langle \text{path} \rangle \mid \text{dragRobot} \tag{5.9}$$

$$\langle \text{path} \rangle \models \text{tapWaypoint+} \mid \text{dragPath} \tag{5.10}$$

$$\langle \text{groupSelect} \rangle \models \textit{“Select Group”} \text{tapSelect} \tag{5.11}$$

$$\langle \text{gestureSelect} \rangle \models \text{tapSelect+} \mid \text{boxSelect} \mid \text{lassoSelect} \tag{5.12}$$

$$\langle \text{selection} \rangle \models \langle \text{gestureSelect} \rangle \mid \langle \text{groupSelect} \rangle \tag{5.13}$$

The terminals of the language are the gestures recognized by the system, that is to say selection by tapping, box, or lasso, waypoint tapping, dragging a robot along a path, dragging a path, dispersion, the buttons and double-tapping to end a command. The terminals in italics in the ABNF representation are the names of the buttons.

Chapter 6

UI Design for Trained Users

The work on which this study is built was motivated in part by the idea of using robots in search and rescue (SAR) operations, to explore damaged buildings, enter unsafe areas, and provide additional sensor data and coverage while reducing risk to first responders. First responders are trained on the equipment they use, but such training may be infrequent, and as a result, complex control systems may not be remembered when the time comes to use the equipment [Micire, 2010]. Because a SAR situation may develop rapidly, there is little or no time available for retraining, and so an interface for use by first responders would prioritize easy learning of the interface and use with minimal training over richness of the possible commands.

However, SAR is not the only domain in which swarm robots might be used, and so considering only the gestures which maximize the ability of untrained users ignores domains where it is possible to train the users extensively. In such a context, adding complexity to the gesture set could allow for additional expressiveness in the interface.

One such high-training environment is the space program. Many SAR

responders are not primarily SAR professionals, but work primarily in other capacities. NASA astronauts' primary job is service as an astronaut, and they spend a significant amount of time in training for missions, including using mock-ups of the interfaces of systems that they will use during their missions. For example, operation of the remote manipulator on the International Space Station (ISS) begins with 30 hours of "Generic Robotics Training" in a virtual environment, which is used to qualify them for future, more specialized, training on the specific arm used on the ISS [Liu, Oman, Galvan, and Natapoff, 2013]. Some astronauts do not pass this test, and as a result, are not qualified to use the manipulator. NASA has developed the TLX (Task Load indeX), which is a widely accepted subjective metric for the physical and cognitive workload imposed on the human operator during a task [Hart, 2006]. As a consequence, NASA is a good example of an environment where highly-trained users could be expected to interact with a more complex interface, and where interfaces could be evaluated to determine if they reduce or increase the workload on the user, as in [Fong, Bualat, Edwards, Flückiger, Kunz, Lee, Park, To, Utz, Ackner, et al., 2006].

On-line Training

Because user experience of video games does have some effect on their use of the user interface described in this work, it is interesting to examine other aspects of video games, and how they might be applied to user interface design for swarm robotics.

It has been argued that the use of game-like user interfaces is inappropriate for the design of applications, but better suited to training material [Thomas and Macredie, 1994]. The basis for using a game-like user interface for an application

is that games are engaging, and as a result, using a similar interaction paradigm should yield an engaging application. However, the dissimilarities in user motivation make this approach less useful. Games are played for the sake of their own play, rather than to accomplish a task outside of the game. Challenge and complexity are controlled in the game to make the play more rewarding, but extending this to applications for external tasks means making the tasks more difficult than they have to be, which will be a difficult proposition for end users to accept. Additionally, work and play are strongly divided culturally, and making one like the other will be resisted by users. Thomas instead proposes that computer-based training materials are a better target for game-like user interfaces. Like games, training materials are intended for high involvement over short periods, and are interacted with for themselves, rather than as part of the completion of an external task.

The majority of the work on video games and learning is centered around the outcomes of games developed to be educational. However, even games that are intended to be purely entertaining still educate their users in the use of the game itself. Modern games frequently include a tutorial level at the beginning of the game in order to familiarize the user with the controls of the game. The tutorial level provides instruction for the user in performing game actions, and assesses whether those actions have been performed, in order to control progression to the next section of the tutorial. For example, a game that includes horseback riding as part of gameplay may provide the horse to the player during the tutorial level, and instruct the user in how to use the controls to mount the horse and direct its movement. Games do not always provide tutorials, and the lack of tutorial content, or the low quality of tutorials when present, has been cited as a potential usability issue in games [Pinelle, Wong, and Stach, 2008].

The tutorial level of video games is usually somewhat separate from

the main content of the game, in that actions in the tutorial have limited or no consequences in future gameplay. This raises a potential issue for implementation of game-style tutorials in a robot control system. If the system allows untrained users to control real robots in a tutorial mode, the user may make an error that does have permanent consequences for the future use of the system, such as causing a collision between robots. If, instead, the system allows the user to operate a simulation, the design and development overhead of implementing the tutorial is quite large, for a component of the system that the user may use very rarely or never. This dilemma may restrict pre-use tutorials to training the user as to which gestures the system can recognize, and only permit a sketch of the possible robot reactions to those gestures.

These forms of training center on the gameplay of a specific video game. However, as discussed in the section "Video Game UI Design", there are commonalities in user interface designs across games, particularly within a given genre of game, and similarities exist between the top-down views used in Real Time Strategy (RTS) games, and the top-down swarm view used in this work. Indeed, some users remarked that the interface was "like Starcraft", which is an RTS game, and used Starcraft-like box selection gestures. This sort of similarity has been leveraged to control the IRobot Packbot using a controller from a Sony Playstation or Microsoft XBox, rather than the Packbot OCU. The Packbot OCU is large and has a large set of individual controls for each function of the robot. After significant use, Packbot operators learn complex tricks to maximize their ability to control the robot [Micire, Desai, Drury, McCann, Norton, Tsui, and Yanco, 2011]. IRobot developed a method to control the packbot using a game controller instead, because soldiers already had practice using it from playing video games [Glaser, 2010].

Rather than including tutorials, the system could be developed with a

basic level of functionality based on user elicited gestures, as in this work, and attempt to recognize certain relatively inefficient patterns of user activity. When such a pattern is recognized, the system would propose alternatives that extend beyond the easily-discovered gestures to offer greater finesse or complexity. For example, in the user experiment, some participants performed the Move to Area A task in the 10 robot case by selecting each robot in turn and dragging it to area A, while other participants made a selection gesture and then a single motion gesture. If the system detects a repeated pattern of single robot interactions, it could display a hint to the user that selection of multiple robots can be done, and indicate how to use it. In order to minimize annoyance, the user should be able to dismiss the notification, or prevent future notifications if they feel that they are sufficiently proficient in the use of the interface.

The data set collected in the user gesture experiment provides an interesting sample of possible advanced gestures. Rather than being common across a large percentage of the users, these were gestures that a small number of users selected, in an attempt to provide additional expressive power. The rest of this chapter discusses the "long tail" of user gestures, and their possible use as "superuser" gestures after an appropriate training period.

"Other" Gestures

The "other" gestures are those that did not match any of the other codes as described in Appendix A. The largest category of other gestures is gestures that were some form of "sweeping" the robots around, but because of the diversity of these gestures, there was little agreement from user to user. Some users made gestures over the screen, as if sinking their fingers into sand or a pile of small objects

and dividing the pile, or cupping their hands to bring the pile back together. Others used the edge of a finger (typically the index finger) or the heel of the flat hand to push robots, as if they were sweeping them along a countertop. These sorts of physical gestures are in line with the expectation of Natural User Interfaces (NUI) that the system can leverage the user's existing intuition about physical objects to guide their interaction with the screen.

However, detecting and using these gestures is prone to some degree of technological difficulty. Gestures made off of the screen, in the space above it, would require additional sensing to detect, such as a depth camera pointed down at the screen or towards the user from the front (or both). These gestures could be mapped back into the space of the screen by flattening them to the areas that they occur over, but the use of a sensed volume could also provide the path towards operating a UAV swarm in a direct manner. The user could use free space gestures to define boundaries for the swarm, or push the swarm around directly. However, such a user interface does not provide feedback in the same space as the gestures. In the user interface proposed for the gesture elicitation experiment, the response to a 2D gesture occurs on a 2D surface, and is a projection of a less-than-3D area onto that 2D screen. The reason that the area is less than 3D is that while the terrain may have elevation variation, the robots do not leave the terrain. As a result, the user does not have to mentally "reproject" the 2D space into 3D in order to understand the interface. For a swarm of UAVs, the display would have to do as much of this work as possible, to limit the mental labor performed by the user. For example, the UAVs could be displayed at a size related to their distance from the viewport provided by the screen, using perspective to indicate distance. The viewport might also need to be movable, as the user may wish to view the shape of the swarm from multiple angles.

In addition to the possible issues with gestures in space over the screen, the technology of the multitouch interface can constrain the available gestures. The Microsoft Surface tracks contacts as areas with a shape, centroid, and so forth. In software operating on the Surface, finger contact points are typically approximated as ovals, and distinguished from larger contact points such as the heel of the user's hand, by their size. As a result, if the user extends their fingers and places the edge of their hand on the screen, and then slides their hand along the screen, the Surface will report a single oval, much longer than it is wide, moving along the screen.

The multitouch screen used in this work tracks up to 20 contact points, but it treats them as points, each representing a single pixel location on the screen. If the user preforms a sweep with the edge of their hand, the screen will report a large number of individual contacts in the region under the user's hand, rather than a single region of contact. As a consequence, gestures that use a region of the hand other than the fingertip are difficult to recognize on the 3M screen.

Gesture Modification

Participant actions in the formation task provided two interesting gesture commands that used alterations of previous commands to increase the versatility of formation commands. The first was indicating that the gesture was intended as a formation command by holding one finger down on the robots while drawing the formation with the other hand. This gesture arose from one participant's initial use of a single-finger drag to move the robots in the initial "Move the robots to area A" task. For the line formation task, the participant initially used the same gesture, placing a finger on the robots and then dragging the line on the screen. The participant then recognized that a simple drag was ambiguous with the motion command, and so

placed one finger from each hand on the robots, holding one still and dragging the other one out to indicate the line for the robots to form. The gesture was initially explained as "pinning" one end of the robots and "stretching" them out along the line. On the next task, commanding the robots to form a square, the participant again placed both fingers on the screen, but drew a square rather than a line. This sequence of removing ambiguity and then extending from the disambiguated gesture resulted in a set of gestures that seems to compose a "formation" gesture (placing and holding a finger) with a description of what that formation should be. This gesture ended up not being used in the gesture set for the robot interface due to the fact that only one participant chose it, but in a context where training was available, it would be relatively simple to train users to perform this gesture.

This sort of combined command-and-modification gesture would probably be easier to remember than sequences of gestures, as the sequence would consist of a larger count of discrete actions, rather than being experienced by the user as a single action. Additionally, the fact that the hold-and-draw formation command has one hand remaining still is likely a good idea from a design perspective. In the physiological research literature, tasks where each hand performs a different action are referred to as bimanual coordination tasks. Bimanual tasks interfere with each other, as in the classic example of patting one's head with one hand and rubbing one's belly with the other. Interference between tasks of differing difficulty in a bimanual coordination task was seen in the case of aiming, where the hand making the easier motion was shown to slow to match the hand making the more difficult motion, rather than the more easily-completed motion being finished first [Fitts, 1954]. For an overview of the neurological basis of bimanual coordination and the various factors that influence interference between tasks, see [Swinnen and Wenderoth, 2004]. Having each hand performing a different gesture is cognitively

demanding, compared to having one hand remain still or perform the same gesture as the other hand, so designs that allow one hand to remain still will be easier to use than those that require drawing different symbols with each hand, for example.

Another participant gesture that made use of formations was the use of an adverb in the gesture language as described in the section "User Strategies for Manipulation". The participant used one finger to draw paths for the robots to follow, but specified that if two fingers were used to draw the path, the robots should follow the path while remaining in their current positions relative to each other. Modifying path drawing in this way allowed the participant to form the robots into a scoop-shaped formation, and then to move the scoop to position the crate in the crate manipulation tasks.

As with the formation command example above, the gesture is an extension of a previous gesture, but with an addition to disambiguate it. These signals to the system that the command is being modified point to a commonality in the participants' thinking about the design of the gestures. Participants added additional contacts to a base gesture to indicate an extension of the base gesture to a special case. Holding one finger still changed path following into formation, and dragging two fingers changed path following into path following in formation. This combination of additional fingers with a base gesture suggests that other base gestures could be similarly extended with additional fingers to activate different, but related functionality. For example, some of the tasks displayed a mix of orange and red robots in one area (for an example, see Figure B.18), to attempt to elicit user gestures for sorting robots. A single-finger selection could select all robots, and adding fingers could be used as the extension to select each different color of robots in turn, with e.g. two fingers selecting the red robots and three fingers selecting the orange robots.

Gesture Direction

Another area in which gestures may be treated with finer granularity is the direction in which the gestures are made. A few participants in the gesture elicitation experiment made verbal mention of this distinction, or used it in their control schemes. One participant used a clockwise lasso as a robot selection gesture, but a counterclockwise selection gesture to indicate dispersion. Another participant indicated that lasso selection should be done clockwise, while patrol areas were selected with a counterclockwise closed loop, similar to a lasso. The participant did not indicate how they would perform a patrol that moved in a clockwise direction around the patrol area, or if the direction that the gesture was made in was intended to constrain the patrol to move in the same direction as the gesture.

Box selection also has the potential to be performed in four different orientations, with the gesture starting from the top left, top right, bottom left, or bottom right corners of the box. One participant made use of this distinction, comparing it to the box selection in the Autodesk Solidworks CAD software. Solidworks has two modes for box selection. If the mouse moves from left to right, the selection includes only objects that are contained entirely within the box. If the mouse moves from right to left, the selection includes all objects that the box overlaps to any degree. The participant used box selections starting from the top right to separate part of the robots from the others, and box selections starting from the bottom right to select only robots that had been marked as being in a particular group. Most other participants only used box selections starting from the top left of the box area, so for a UI supporting the distinction between the different forms, starting from the top left should be the basic gesture, and box selections starting from the other corners of the box would be reserved for more specific forms

of selection.

It is worth noting that there are inherent physical limits on the methods of distinguishing gestures explored so far. Adding more fingers to a gesture is limited by the multitouch sensing technology and the user's supply of fingers. Lasso can only be performed in two different directions, and box select only has four different corners that the selection gesture can start from.

Gesture Velocity

Another possible distinction between gestures is the speed with which the gesture is made. One participant in the gesture experiment made a distinction between dragging a robot, which is performed slowly, and flicking a robot (to remove it), which is performed rapidly, with the intent of metaphorically tossing the robot off the edge of the screen. It seems unwise to alter the meaning of a gesture based on its speed, as in the case of movement becoming deletion if performed quickly, because a user may accidentally perform a gesture too slowly or too quickly, and so obtain a different result than they expect. However, the velocity with which a movement gesture is performed might reasonably be used to convey to the robots that the motion is to be performed quickly, while a slow gesture might indicate that the robots should proceed slowly. Research would have to be performed to ensure that the increase in velocity does not result in a loss of accuracy beyond what the user is willing to accept.

Scaling of the velocity from the user commands to the robot commands also presents a possible issue, as the user interaction point may traverse the map more quickly than robots can traverse the real terrain. In a relatively extreme case, if a map of the continental United States is displayed on a typical touchscreen, the

user could easily swipe their finger from one coast to the other in a few seconds. No practical mobility platform can be expected to match this velocity. As a result, the robot speed and user gesture speed may be related in a potentially complex way by other factors, such as the scale of the map view, leading to difficulty for the user in understanding how fast the robots are being commanded to move.

Assessment of Training-Oriented Gestures

If the system were developed with this style of gesture set, assessment of the validity of the design choices would differ from the assessment of a system based on gestures intended to be useful to untrained users.

In the case of a system intended to be useful to untrained users, the system's primary metric of effectiveness would be the ability of untrained users to complete tasks at all. In the experimental study, some users performed the formation task by repeating the single-robot motion gesture to move each individual robot into a position on the formation. This control scheme would allow the user to complete the task, assuming the single robot motion gesture was supported, but it is relatively inefficient compared to using a formation command button once. Many of the other tasks could be performed in this way, as sequences of single robot move actions, so as long as the user discovered that interaction method, the system could be used, albeit slowly.

For a system that permits a longer training period, factors such as efficiency and memorability of the commands would be important, as well as the length of time to become proficient. Efficiency could be quantified as gestures required to perform a task, with a lower gesture count indicating a higher efficiency. To return to the previous formation example, moving ten robots into formation

with single robot motions requires ten gestures, while using a formation button requires a single selection gesture, a button click, and a single gesture to draw the formation, for a total of three user interface interactions.

Testing the memorability of the system would require testing of the same user population after training, and again after a period of time not using the interface, to determine how many of the gestures were learned during the training, and how well they were retained over the gap of non-interaction time. Memorability interacts with training time, as it would be expected that a more memorable gesture set would allow users to become proficient more quickly than one that is difficult to remember. One factor that may affect the memorability of the instructions is similarity across modifications of gestures. For example, if a two-finger box select selects only robots of a specific color, a two-finger lasso selection should also only select robots of a specific color. As a result, the user can learn that two fingers is the modification that makes the action color-specific, and combine that modification with basic gestures to create more precise gestures. To extend that example, use of two fingers in a formation gesture might indicate that only robots of a specific color participate in the formation. However, enforcing this kind of consistency may result in implementation of unneeded gestures that complicate the gesture space. For example, maintaining consistency may result in commands such as sending only robots of a specific color to move a box, when the main concern to the user is that the box be moved, not what color the robots doing the work are.

Missing Gestures

Neither the consideration of "superuser" gestures nor the interface designed from naive user gestures considered certain types of user interaction. For example, the

tasks specified in the experiment did not include gestures intended as commands to the user interface itself, such as commands to change the viewpoint from which the user was observing the robots. One participant did suggest the use of reverse pinch as a zoom gesture, as is frequently used on cell phones, for a task where the user had to interact with one robot in a group. By zooming in, the user expected to have that robot cover more of the view, and so be easier to interact with. The user interface also did not ask the user to alter the color of individual robots, despite having some situations where the robots were divided into groups by color.

The absence of commands such as viewpoint changes and alteration of robot colors was intentional, so the user could have a feeling of interacting directly with the robots, rather than interacting with the interface. Adding a set of commands that are only to the interface, and not to the robots, creates a layer of intermediation. Adding a second set of commands also creates additional opportunities for ambiguity, as the commands to the user interface and to the robots must be separate, so that the interface can determine which commands influence it, and which commands are intended to be converted into programs for the robots. The conversion of user gestures to robot programs was of more interest for this work, but in the future it would be useful to determine what commands users might want to issue to the interface, and how they can be separated from commands to the robots.

The tasks presented in the experiment were intended to have an overlap with the tasks from [Micire, 2010] for purposes of comparison, but also to have a reasonable degree of coverage of actions the user might want to have robots perform. However, there is no task for sensor overwatch, or any other task that requires that the robots be pointed in a certain direction. As a consequence, the gesture set is somewhat agnostic to gestures that would allow the user to specify a heading for

the robots to face. It could be argued that in this particular experiment, as the user was told to assume that the robots could execute the task they were given, that the robots will end up pointing whatever direction they need to point to complete the task. As a result, if the task was sensor overwatch, the robots would be positioned such that their sensor fields overlapped the desired area.

This lacuna in the range of gestures elicited by the experiment points to a more interesting element of the experiment design for future research. Heading is a quality of an individual robot (although, clearly, a swarm can point to a common heading), and so heading gestures could be elicited in a single robot case. It may be useful to consider what actions can be reasonably regarded as single robot actions, such as facing a direction, turning in place, controlled movements, and so on, and compare the gestures selected to perform those actions on a single robot with the same actions, applied to a swarm. In addition to providing a very fine-grained treatment of control gestures, the extension of the gestures to swarms may reveal interesting qualities of the participants' thinking about the swarm. For example, if a swarm is commanded to rotate in place, do users expect each robot to move on an arc around the centroid of the swarm, or each robot to rotate in place to a new heading? In either case, how is the expectation reflected in the gesture used? Based on the results of this work, particularly in the unknown number of robots or "cloud" case, the participants' understanding of the swarm as a single entity would likely override their thinking of it as a large number of individuals, and so they would likely expect the swarm to reorient with some robots moving on arcs around the centroid of the swarm, rather than each robot rotating in place.

Gesture Coverage

Related to missing gestures, that is to say, gestures that were not elicited by the design of the experiment, is the question of gesture coverage. Because there was no constraint on the gestures the users could choose, all of the users performed (or described) some form of interaction for each task they were presented with. There was no case where a user was presented with a task, but could not complete it because they did not have a gesture available, because any gesture they could imagine and physically perform was available.

However, once the user gestures were codified into an interface, there is a limitation on which gestures are acceptable and which are not. As a consequence, a situation may arise where the available gestures in the interface do not provide the user with a way to complete the task. In this case, the coverage would be incomplete: the set of gestures does not cover all the possible tasks.

Determining that all possible tasks are covered by a fixed set of gestures may not be a useful problem to attempt to solve. Without a method of determining the likelihood of each situation, and the cost of failing to respond effectively to it with the given interface, significant effort would be expended on situations that my simply never arise. For a real-world SAR application, a preferable approach may be to provide a set of gestures covering a range of abstractnesses, with simple motion to a point at one end, and more complex behaviors such as formations, specialized forms of dispersion, and area searches at the other end. By providing different granularities of control in this manner, the user can use the more complex behaviors when they recognize a situation in which such behavior is useful, and attempt to cross gaps in the gesture coverage by combining the simple behaviors. Some hope for the validity of this approach is to be found in the use of single robot

movement gestures, repeated for each robot, to send the robots to formation by some participants in the 10 robot case. Rather than suggest a "formation" gesture, these participants combined gestures they had already created for motion to a point, and used them to perform the formation task. In the case of a gap in the gesture coverage, similar ingenuity and the availability of relatively fine-grained basic actions may allow users to complete the task despite the lack of coverage. However, some care must be taken to ensure that the basic actions scale well with the swarm size. The use of single robot moves to enter formation is possible with more than ten robots, but becomes prohibitively tedious.

Chapter 7

Implementation of Swarm Actions

Finally, the commands specified by the user must be conveyed to the swarm. It is important to include the path specified by the user in the path planning for members of the swarm. By following the path laid out for it, the behavior of the swarm is made legible to the human user. Compliance with the requested path indicates to the user that the swarm is under their control, and allows them to assess the progress of the system towards the goal. If, instead, the system uses some other form of path planning, it will initially appear to not be doing what the user requested, even if the end state does eventually become what the user intended. What factors contribute to this legibility and how best to balance them with other operational constraints of the system is beyond the scope of this paper, but has been examined in the context of manipulation, navigation around people, and combined in mobile manipulation around people [Beetz, Stulp, Esden-Tempski, Fedrizzi, Klank, Kresse, Maldonado, and Ruiz, 2010; Kruse, Pandey, Alami, and Kirsch, 2013; Dragan, Bauman, Forlizzi, and Srinivasa, 2015]. The work on legible manipulation also distinguishes between predictable motion, which makes sense to

observers based on a known goal, and legible motion, which allows observers to infer an unknown goal. Extension of legibility and the emotive content of motion to swarm robots has also been recently investigated, but not in relation to human control of the swarm [Dietz, E, Washington, Kim, and Follmer, 2017].

The user input may also contain information that is not available to the swarm, but is available to the user. This information may be necessary for the successful completion of the task, and so is not to be discarded lightly. For example, the optimal path in terms of minimal travel distance may be blocked by some transient condition, especially in the case of disaster response, such as fire or flood, and so the user may direct the robots to take a longer route to the goal. As such, the user's path selection can be viewed as an attempt to convey information about the desirability or utility of a given path to the robot, and so following the path given by the user is preferable to not following it. Discarding this information in favor of the shorter path could result in unit loss and mission failure.

All of the valid expressions possible in the command language should be converted into programs for the robots, or the user must be usefully informed as to why it was not possible. The synthesized program should result in convergence of the swarm's overall behavior to the desired result. Clearly, in a developing situation in the real world, success may become impossible, and so there is not a practical way to guarantee that a particular valid command sequence will result in a particular desired state of the world. However, certain minimum bounds on the problem may be able to be used to determine if a desired task is certain to fail.

Localization

The initial vision for this work was to have minimal sensing and no localization. Minimal sensing results in a lower cost per robot, as it reduces the amount spent on sensors. Not relying on localization, especially localization via approaches such as GPS, which are easily blocked or interfered with, allows the resulting algorithms to be useful in situations where global localization is not available.

However, a user interface that can display the swarm from an overhead view assumes that there is some form of localization, at least in terms of relative distances and bearings between the robots. Without that information, the display of a group of robots in a way that matches their actual locations in the world is impossible, and displaying them incorrectly is likely to mislead the user. Even the unknown number of robots case assumes that the outer bounds of the swarm are at least approximately known.

Fortunately for this work, there are a number of approaches to creating a local coordinate frame using robots with relatively simple sensors. A local coordinate system can be created by communicating agents with a limited, known sensing range but no other concept of distance [Bachrach, Nagpal, Salib, and Shrobe, 2004]. One agent acts as a seed, and sends messages to its neighbors. The neighbors propagate those messages, incrementing a hop counter in the message, and ignore messages with a higher hop count to stop the message from propagating backwards. As a result, each agent knows that its distance from the seed is at most the communication radius times the minimum hop count it has received. By using multiple seeds, each non-seed can calculate its position based on trilateration to the seeds. Extending this method to use received signal strength as a proxy for distance, rather than the constant known sensing range, improves the accuracy of the

approach. The Kilobot swarm uses a range-only sensor along with communication to create a local coordinate system [Rubenstein, Cornejo, and Nagpal, 2014b]. A set of four stationary seed robots act as the origin of the coordinate system. As robots surround the four known robots, the new robot localizes from them and then begins broadcasting its own position. Each new robot that joins the group uses trilateration to localize itself based on the robots that it can communicate with.

In addition to range-only approaches, bearing-only approaches to robot localization have been created, including an approach that uses the bearings to landmarks to create a vector field to drive the robot to a goal [Loizou and Kumar, 2007]. This approach is even able to guide an individual robot in the presence of moving landmarks. It does not, however, produce a coordinate frame that is shared among the robots, unless they somehow agree on the landmarks to use. A scale-free coordinate system can also be created using only bearing sensors and inter-robot communication, to arrive at a coordinate system that is shared among the robots [Cornejo, Lynch, Fudge, Bilstein, Khabbazian, and McLurkin, 2013]. The addition of scale to such a system requires something beyond only bearing measurements. One way to do this is to use structure from motion (SfM) to estimate inter-agent distances, and use the measured distances to determine the scale for the network of robots [Spica and Giordano, 2016]. Alternatively, a pair of beacons of known location could provide the scale information required to convert scale-free coordinates into a metric coordinate system.

Finally, systems that combine range and bearing information are common in swarm robotics [Caprari et al., 1998; Mondada, Bonani, Raemy, Pugh, Cianci, Klaptocz, Magnenat, Zufferey, Floreano, and Martinoli, 2009; Arvin et al., 2009; Farrow, Klingner, Reishus, and Correll, 2014]. Systems with range and bearing can use trigonometry to calculate relative positions of other robots, and so

can bootstrap a coordinate frame by using a method such as spreading inhibition to elect an origin robot, which then informs its neighbors of their positions in its coordinate frame. The neighbors can then correct their position estimates based on communication with each other, as in range-based localization schemes, and inform their neighbors of their positions, propagating the coordinate system outward from the origin.

As there already exist a large number of ways for robots to arrive at a local coordinate system without GPS or sophisticated sensors, the creation of a coordinate frame was determined to be outside of the scope of this work.

Vector Field Path Following

Once the robots have developed a coordinate frame, the space the robots are in can be decomposed into a set of cells, and the program for the robots can be expressed in GCPR as checks on which cell the robot is located in and what the desired behavior for the robot in that cell is. The desired behavior for each cell is based on the user input on the interface device. Since the locations of the robots in the local coordinate system are known, and the location of the visualization of the robots on the user interface device is based on their positions in the local coordinate system, the user's input can be mapped into the local coordinate system to guide the robots. The combination of the spatial decomposition and the user input results in, essentially, a discrete vector field. By having the vectors in the field point towards a user supplied path, or along the path, the robots can follow the path using only their location in the local coordinate frame to guide them.

The space is decomposed into squares of uniform size. The size can be varied, but smaller squares result in a longer program for the robots and longer

runtime for the decomposition algorithm. To determine the desired vector for a the cells of the spatial decomposition, a multipass algorithm is used.

The first pass assigns the vectors for those squares containing a point of the user specified path, or on the line between two points of the user path. Squares containing a point are assigned a vector pointing to the next point on the path. Squares between two points are assigned a vector pointing from the square center to the point closer to the end of the path. After this step, all points on the path have a vector pointing along the path.

The second pass assigns to all squares neighboring the end point of the path a vector pointed towards the end point of the path. As a result, the end point of the path becomes an attractor that robots attempt to reach.

The third pass assigns to all squares that are adjacent to assigned squares a vector which is the average of all of the values of the assigned neighbors. This pass broadens the path so that it is greater than a single square wide, and ensures that the vector field around the path is smooth.

The final pass is repeated until no square is assigned during a repetition. On each repetition, any square that is not assigned, but has assigned neighbors, is assigned the average of its neighbors and a vector from the center of the square to the closest point that is on the path. As a result, squares off the path drive the robot towards the path via the shortest route. When this pass is complete, every square of the decomposition of the space has been assigned a vector pointing in the direction that a robot in that square should move.

The resulting decomposition can then be converted into a GCPR program by having each guard be a check on the position of the robot, and assignment of the robot's desired heading based on the grid square it is in. A GCPR clause tells robots that are off their desired heading to rotate to that heading, and robots which

are on their desired heading to move forward.

Dispersion can be implemented in terms of vector fields as well. In the case of path following, every robot that was intended to follow the path would be issued the same decomposition of the space, but would follow different routes, as they start from different locations. To disperse the robots, each robot receives a different path, starting from their current location and moving to their new dispersed location. As described above, the robots can avoid each other while attempting to reach their new positions, and return to following their assigned paths once they are clear of each other. In implementation, the new locations were chosen by uniform random selection of points over the area the robots were in, but other approaches, such as positioning the robots so that each robot could communicate with at most two other robots, or on a regular grid in the local coordinate frame, would be amenable to the same implementation.

These algorithms for path following and dispersion were implemented in ROS and tested in the ARGoS multi-robot simulator [Pinciroli, Trianni, OGrady, Pini, Brutschy, Brambilla, Mathews, Ferrante, Di Caro, Ducatelle, et al., 2012]. As a basic method for driving robots with some form of localization along a path, and creating a representation of the user input that can be represented in GCPR, a discrete vector field approach works. However, it is limited in a number of ways that eventually resulted in this approach being discarded.

Composition with Obstacle Avoidance

Reactive obstacle avoidance was initially added to the vector field program by only following the desired heading if the robot detects that it is not near any obstacles, and reacting to avoid the obstacles rather than follow the vector field if an obstacle

is detected. This approach does have the weakness that the user can, for example, draw a path which passes through an obstacle. In a known environment, obstacles could be surrounded by repelling vectors, but in an unknown environment, this sort of command will cause the robots to approach the obstacle under vector field control, turn away from the obstacle until it is no longer visible, return to vector field control, and return to the obstacle. Even in a known environment, the local minima caused by "U" or "C" shaped obstacles present a problem for vector field based navigation.

There exist a family of algorithims, called "bug algorithms", which provide complete path planning in a priori unknown environments with minimal sensing, under reasonable bounds, such as that the number of obstacles is finite and the goal is reachable. The bug family is large, and some of its members require sensing which may not hold in all conditions, such as location information or infinite-range distance sensing, but many do not. I-Bug, in particular, requires only the ability to detect a gradient which towards the goal and the ability to circumnavigate obstacles by e.g. wall following [Taylor and LaValle, 2009].

Generally, bug algorithms have two cases, the rules for motion in unobstructed space, and the rules for moving around obstacles. Rules for motion around obstacles frequently combine wall following around the perimeter of the obstacle with a leave condition that causes the robot to stop wall following and return to moving in open space. In the initial bug algorithm paper, the leave condition for Bug1 is to depart the obstacle from the point closest to the target, which requires circumnavigating the obstacle once to find that point [Lumelsky and Stepanov, 1987].

However, simply applying a bug algorithm to the movement of each robot in the swarm may result in undesirable behavior in a number of ways. First, many

bug algorithms rely on wall following to pass around obstacles. If the other obstacle is another robot, operating under the same algorithm, the robots may begin to circumnavigate each other. If the leave condition for the circumnavigation is never met, this behavior would persist indefinitely. Under the leave rule discussed for Bug1, if one of the robots is moving slightly faster than the other, the resulting path of the two robots would be a spiral moving in some direction in space. Since Bug1 uses return to the same location to detect circumnavigation, and the spiral would never return to the same location, the leave condition would never be satisfied.

Unfortunately, composition with a vector field indicating the user-specified path to the goal with a bug algorithm for obstacle avoidance can break the guarantees of completeness that make bug algorithms appealing. For example, assume that the goal is a point with a vector field pointing towards that point from all locations in the operational area. A tempting bug algorithm for passing around obstacles is to circumnavigate the obstacle until the vector field points away from it, then leave and return to vector field control. For a simple polygonal obstacle between the robot and the goal, this results in reaching the obstacle, navigating some distance around it, and then departing it again on the side closer to the goal. However, if the goal is surrounded by a right-handed spiral-shaped obstacle such that any straight line from the goal intersects the obstacle, the goal becomes unreachable. When a robot hits the spiral, it will wall follow in some direction. If it uses a right-handed wall follow, it will reach the outer lip of the spiral and depart the obstacle following the vector field. It will then hit the obstacle again, since the vector field across the mouth of the spiral points towards the goal in its center, and begin wall following again. A left-handed wall follow would reach the goal, but could be defeated by a left-handed spiral. Changing to randomly-selected left-handed or right-handed wall following changes the shape of the obstacles that can trap the robot, but does not

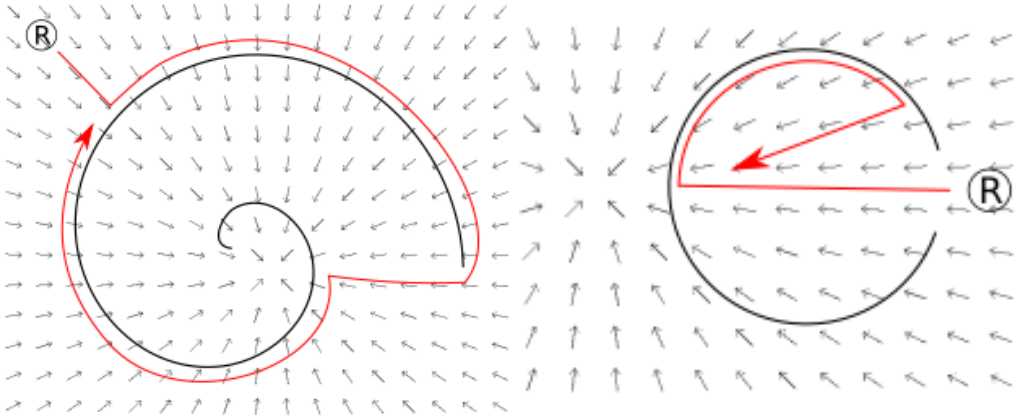


Figure 7.1: Simple obstacles that result in looping behavior for a bug algorithm that combines wall following with leaving the obstacle when the vector field points away from the obstacle.

remove the possibility, so long as the leave condition is simply that the vector field point away from the obstacle.

Neither of the cases depicted in Figure 7.1 has an unreachable goal, in that the goal is a member of the same set of points as the open space of the environment, but attempts at obstacle avoidance preclude the robot reaching it.

Code Generation Refinement

Rather than rejecting bug algorithms because they cannot be usefully combined with a global vector field, the vector field representation of the space was rejected. Vector fields have other problems, beyond unsatisfactory composition with complete motion control planners such as the bug algorithm family. A vector field cannot represent a path that crosses itself, such as a patrol route that is a loop. In the continuous case, a vector field is represented by a function that maps all points in the space to a vector representing the desired robot heading at that point. However, because a function produces exactly one value, each point can only have one heading, while the intersection point of a path has two headings, at different points in time.

For a vector field broken up into discrete grid cells, the same problem applies, but to regions rather than points.

One potential solution to this problem with vector fields is to have multiple vector fields, or multiple values at each point, and allow the robot to maintain a program counter, which it uses to determine which field or which value to use at a given point. The counter is incremented on departure from a vector field grid cell, and then on return to that cell, the new value of the program counter is used in a guard, which results in the robot using the second value of the heading.

For example, the GCPR statements:

(self.is_in((1.74, 1.92), (1.24, 1.42)) and pc_is(1), self.set_desired_heading(2.33), 0.9)

(self.is_in((1.74, 1.92), (1.24, 1.42)) and pc_is(2), self.set_desired_heading(1.03), 0.9)

will result in the robot using the heading 2.33 when it passes through the grid region if the program counter is 1, and 1.03 when it passes through the grid region with the program counter set to 2 (while, unfortunately, not defining a desired heading if the program counter is set to some other value).

Using a program counter to select possible headings for a discrete vector field grid cell has the drawback that the program counter must be incremented on leaving the cell, as incrementing it while remaining in the cell will cause a change of heading. As a result, the generated program must have guards on all cells surrounding the cell, which increment the program counter when triggered. Further adding to this complexity is the problem of sensor noise. If sensor noise causes the robot to erroneously detect that it has entered the neighboring cell, then the program counter could be incremented without the robot actually leaving its current cell. On the next update of the noisy position sensor, the robot could then "return" to the current cell, and change direction due to the incorrectly implemented program counter.

As a consequence of the complexity attendant on repairing the inability of the vector field to represent paths containing loops, the underlying representation of the conversion of user paths into robot programs was changed to avoid the use of vector fields.

Approaching a Point

The basic action of the robot is to move from one location to another. Under the assumption that there exists some form of localization, which must hold in order to support the type of user interface described in this work, the simplest approach to navigation to a point is for the robot to rotate to face that point and then move forwards towards it. Once the point is reached, the robot should stop.

Reaching a point is more complex for swarm robots than it is for single robots. Some work assumes, for theoretical development, that robots are points, and as they occupy no area, any number of them may congregate at a mathematical point. In the real world, while a point may be described precisely, it is usually sufficient to arrive within some small distance ϵ of it to say that the robot has reached the point. However, robots also take up space in the real world, and so as more and more members of the swarm arrive at the point, the later arrivals may be precluded from actually approaching to within ϵ of the point. In this case, it is desirable to have a definition of arrival that expands to allow robots to approach the point as closely as they can and stop when that condition is met.

Completeness of Navigation

As described earlier, bug algorithms are complete, in that they navigate the robot to a point, or determine that the point cannot be reached. A review of eleven bug

algorithms can be found in [Ng and Bräunl, 2007]. Bug algorithms also generally rely on only local sensing and minimal data storage, and so are appealing for use in swarm control. In this section, the basic bug algorithms are extended to determine if it is possible to provide local-sensing-based complete algorithms for the tasks from the user test.

Unfortunately, most of the bug algorithms have the requirement that the obstacles in the environment are not moving. Indeed, the presence of moving obstacles results in navigation becoming undecidable without knowledge of the future movement of the obstacles, as an obstacle can move to occupy the robot's goal. Unless it is known whether the obstacle will move off the goal in the future, it cannot be determined whether the goal is unreachable, or just not currently reachable.

The TangentBug algorithm has been extended to handle moving obstacles, given a number of constraints [Kamon, Rimon, and Rivlin, 1998; Tomita and Yamamoto, 2009]. The main constraint that affects the use of this algorithm is that the obstacles are constrained to be moving at a velocity that is slower than that of the robot. This constraint is required because if the robot is circumnavigating the obstacle, and the obstacle is moving faster than the robot, then in the time that the robot requires to circumnavigate the perimeter of the obstacle, the obstacle will have moved a distance greater than its own perimeter is long, and the circumnavigating robot will have moved with it. As a consequence, the circumnavigating robot might not return to its own previous path and cross it, which is the condition that Tomita and Yamamoto use to determine that the robot should leave the obstacle.

At first, this would appear to be a problem for swarm robots, because if the robots are the same, they will be moving at the same speed. If one moves in a straight line, and the other attempts to circumnavigate it, the circumnavigating

robot will never cross its own path for the reason described above, and so never leave. However, if the robots are using the same bug algorithm, this trap will not be sprung, because each robot will attempt to circumnavigate the other. If they attempt to circumnavigate each other in opposite directions, they will spiral around each other, leading to at least one of the robots crossing its own previous path, and triggering the leave condition of the algorithm. If they attempt to circumnavigate each other in the same direction, they will come to a position side-by-side, as neither can outpace the other, but neither will pull away from the other because they are attempting to follow each other's perimeters. In this case, they are pointed towards the goal, because in the absence of an obstacle, Tomita and Yamamoto's modified TangentBug orients the robot towards the goal, and so before they encountered each other, the robots were oriented towards the goal. The robots will then approach the goal, and one will arrive, while the other circumnavigates the first until it detects that it cannot arrive at the goal.

Tomita and Yamamoto do not deal with the decidability of their modification of TangentBug because they constrain the goal point to be not within an obstacle, and so reachable by the robot. The original TangentBug will navigate the robot to the goal if it is reachable, or circumnavigate the obstacle, returning to its starting point, whereupon it detects that the point is unreachable [Kamon et al., 1998]. Since Tomita and Yamamoto constrain the goal to not be within an obstacle, the original TangentBug will reach it.

In the case of moving obstacles, if the goal is covered by an obstacle, the obstacle is either moving or not moving. If the obstacle is not moving, the arriving robot will circumnavigate the obstacle, return to the hit point, and stop, having detected that the goal cannot be reached. If the obstacle is moving, the modified TangentBug will not return to its original hit point, which is either left behind or

covered by the obstacle, but will eventually cross itself, and leave the object towards the goal. If the obstacle is still covering the goal, this process will repeat until the object is not covering the goal anymore, and the robot will reach the goal.

In the case of swarm robots, as described above, some mechanism may be needed to determine that the goal is occupied, possibly by other swarm members, and to stop at a location near the goal. The stopping condition of the original TangentBug in the case where the goal is unreachable extends naturally to swarm robots.

If a robot is the first to arrive at the goal, the goal is not occupied, so the robot occupies it and stops. If a robot is not first to arrive at the goal, the goal is occupied by a robot, which is stopped. The new arrival treats the stopped robot as an obstacle, circumnavigates it, returns to the original hit point and so detects that the goal is unreachable, and stops. If multiple new arrivals get to the stopped robot(s) at the same time, the conditions above hold, and so they eventually either cross their own paths while trying to circumnavigate another robot that is also circumnavigating an obstacle, and so leave and return (and so become later arrivals), or complete a circumnavigation and return to their own starting point and stop (becoming part of the obstacle). Unfortunately, this method of handling late arrival will cause the robot cluster to grow in the direction from which most of the robots arrive.

If a maximally dense cluster of robots is desired, the unreachability check of TangentBug can be extended. Once the goal is determined to be unreachable, the robot performs another circumnavigation of the blocking obstacle to find the closest point to the goal, and returns to that point. That point has either been occupied by another robot, in which case the robot repeats this step, or occupies that point if it is free. Because every iteration of this step fills the point closest to

the goal with a robot, the resulting cluster of robots is packed as close to the goal as possible.

Path Following

As discused previously, it is desirable to have the robots follow user-defined paths rather than simply determining what the end position of the robots is to be and heading directly to that point. The reasons for this are twofold:

1. Following the user path makes it clear to the user that the robots are doing as the user directed.

2. The user path may be chosen as a consequence of information that the user has and that the robots do not, and so satisfies some property other than simply arriving at the goal, such as avoiding a dangerous area or providing a sensor sweep of an unexamined area.

A user path can be followed by a bug algorithm by breaking the user path into a series of points, and using a bug algorithm to navigate to each point in turn. Because the bug algorithm is complete, it can either reach the target point, or determine in finite time that the point is not reachable. Motion to the goal point is complete, as it is simply TangentBug navigation to a point and TangentBug navigation has been demonstrated to be complete under reasonable constraints. Motion to the point prior to the goal point can either reach the prior point, or detect that it is unreachable. In either case, the algorithm then transitions to attempting to reach the goal point, which has been demonstrated to be complete. Since motion to the point before the goal is guaranteed to transition to a complete approach to the goal, it does not alter the completeness of the algorithm. The same logic holds

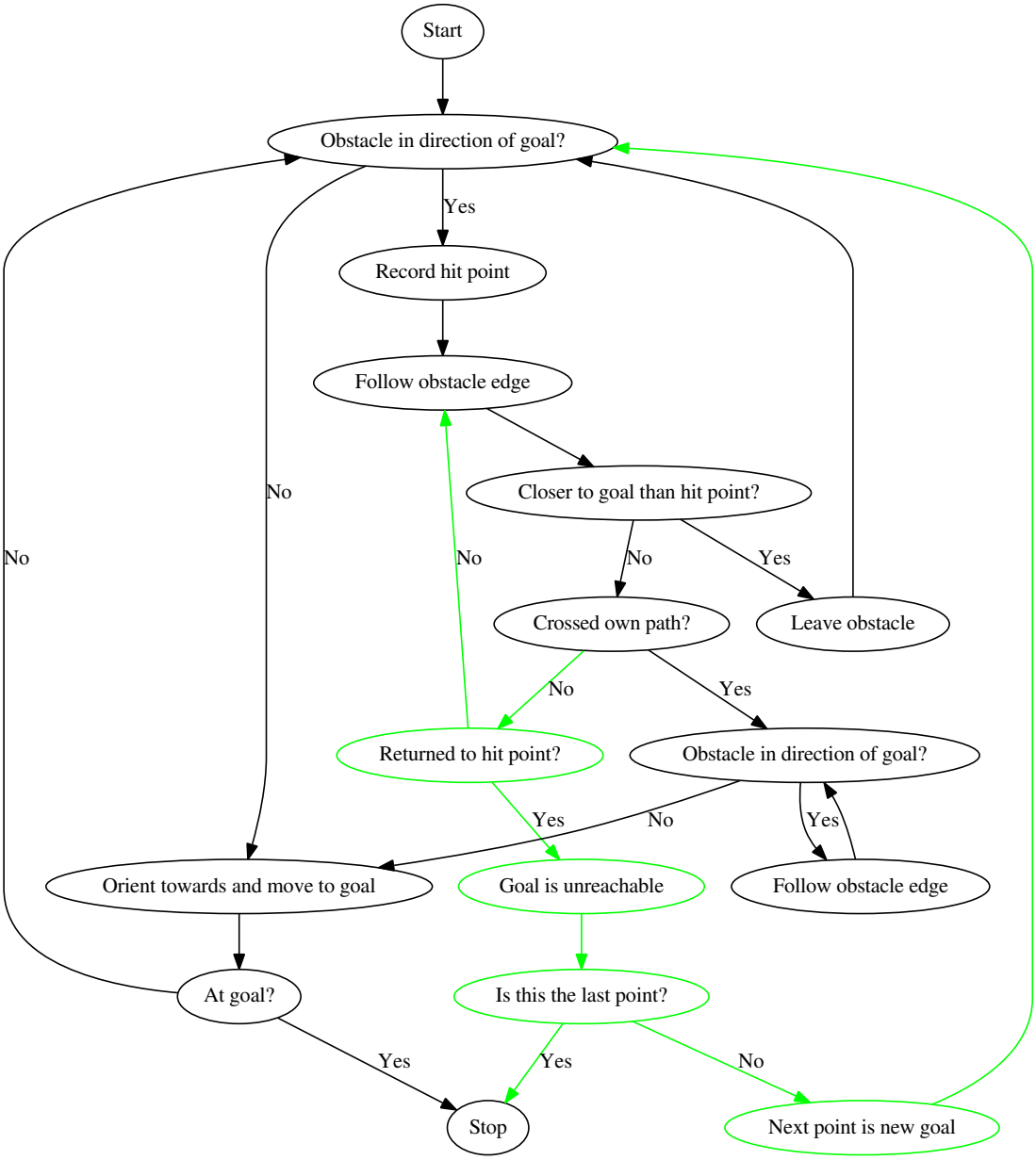


Figure 7.2: The proposed modifications (in green) to the Tomita and Yamamoto TangentBug algorithm for user path following in cases with multiple robots, some of which may be treated as moving obstacles. This flow chart does not include the option for maximally dense packing at the goal described in the text.

for each point prior to the attempt to reach the goal point, and so the entire path following algorithm is complete.

However, the path followed by the robot may miss one or more of the user-defined points, and in fact may miss all of them, if, for example, the entire user path happens to be within a previously undiscovered obstacle. The upper bound on the length of the path that TangentBug will produce is given by

$$P_{max} = ||S - G|| + \sum_i \prod_i + \sum_i \prod_i \times \#Minima_i$$

where $||S - G||$ is the straight line distance from the start to the goal, $\prod_i$ is the perimeter of the i^{th} object, over all objects i in the disk with radius $||S - G||$ centered at the goal, and $\#Minima_i$ is the count of local minima in distance to the goal on the perimeter of the i^{th} object. In the case of iterated TangentBug, this bound is then summed over each goal point. While the perimeters of the objects are constrained to be finite in order for TangentBug to converge, they may be very large, and so send the robot far from the user path. Whether this deviation is acceptable or not depends on factors outside of the scope of *a priori* algorithm development, such as the parameters of the mission that the user is sending the robots on, and so is not amenable to solution here.

Formation

In the user interface tests, the formation task was to cause the robots to form a formation, but did not involve motion in formation. It was not specified whether the formation was to be a form of aggregation which constrains the robots to be within the perimeter of the desired formation, or if the robots were required to be on the perimeter of the formation, with no robots inside or outside of the formation

area.

In the case of robots with localization, the formation can be sent to the robots as a list of points, similar to how a path would be sent to the robots as a list of points, and the robots can select points that are either inside that formation or on its edge, as needed, and then follow paths to those points. However, such a naive algorithm could result in the loss of the completeness properties of the bug algorithm if failure to reach the target point is handled poorly. If the robot simply selects a new point each time its target point is not reachable, and no point in the formation is reachable, then the algorithm will loop forever.

Instead, the path following behavior can be adapted to generate a path that will position the robot on the perimeter of the formation, or determine that such a positioning is impossible. The formation algorithm consists of two phases, approach to the perimeter and positioning along it. Approaching the perimeter is done as in path following, with the perimeter points serving as the list of points on the path. As with path following, the robot will attempt to reach each point along the perimeter in turn, and if none of them are reachable, the algorithm will terminate, having determined that the entire perimeter of the formation is inaccessible. However, once the robot reaches any point on the perimeter, there are two options for the algorithm. The robot can stop, as it is now on the perimeter and so "in formation". Any subsequent robot following the path will then treat that point on the path as blocked by an obstacle, and proceed to attempt to reach the next point. As a result, each robot stops at the first unoccupied space on the path, or fails to find any unoccupied spaces and terminates upon failure to reach the last point in the path. Alternatively, the robot could stop when it reaches the last point, or, on failure, backtrack to the last point that it could reach.In either case, as with basic path following, the algorithm is complete.

Interestingly, the Bug2 algorithm from the original work on bug algorithms is a better choice for the basic bug algorithm used this style of formation than the modified TangentBug algorithm [Lumelsky and Stepanov, 1987]. While TangentBug does have a lower bound than Bug2 on the maximum path length, Bug2 uses a path called the m-line which is the line segment passing between the start point and the target point. For a polygonal formation expressed as a list of points, the m-lines between each subsequent set of points form the perimeter of the formation. As a result, any point which is on an m-line and in free space, rather than within an obstacle, is a valid point for the robot to stop on and consider itself "in formation".

The use of bug algorithms as outlined here does not address the problem of dispersion along the edge of a formation. A reasonable expectation of the behavior of a swarm commanded to form, for example, a square shaped formation is that the robots would be approximately evenly distributed along the perimeter of the square, at least where there were no obstacles to prevent it. Having the robots all located around one corner of the square or along one edge, while technically on the perimeter, would be less satisfactory. There are a number of possible approaches to spacing along the formation perimeter that might be adopted, depending on the robots' sensing abilities. Since the basis of this work includes a central user interface which generates the robot programs and transmits them to the individual robots, one way to distribute the robots would be to base the decomposition of the formation perimeter into segments based on the number of robots available, and have each robot's final destination be a different point on the formation perimeter. If the programs followed the entire perimeter as closely as possible before stopping, they would preferentially take their final path point, and fall back to points along the formation if their final goal was unavailable.

If the robots can sense each other, they could disperse along the perimeter

according to some higher-level rules for setting their own goals, but using bug algorithms to navigate. For example, if it is known that enough robots exist to cover the perimeter tightly enough that every robot can see a robot to its left and right, bounded by some limited sensing radius, robots could move on the perimeter to keep the distances between the robots to their left and right equal, treating points where obstacles intersect the perimeter as robots. However, if the unobstructed perimeter of the formation is greater than the twice the robots' sensing radius times the number of robots, gaps in the perimeter will lead to oscillation as robots attempt to fill them. Lacking a global knowledge of the situation, the robots cannot determine that the perimeter is unfilled.

Patrol

Patrol is distinct from the other tasks because it has an infinite loop in it by design. The robots patrol the area, and once they finish the patrol, they begin again. A complete algorithm is one which will either obtains the correct result, if possible, or indicates failure. In either case, a complete algorithm terminates, while a patrol does not. However, the navigation algorithm can simply be the same as the path algorithm, only looping over the list of goal points rather than passing through each of them once. While not complete because it does not terminate, each attempt to reach a point on the patrol path is complete, in the sense that it either reaches the point, or determines that it cannot be reached. The algorithm moves on to the next point in either case, rather than terminating.

In a non-dynamic environment, obstacle-free environment, the algorithm will cause the robot to arrive at every point of the patrol path, since there are no obstacles to prevent it. Adding obstacles raises the possibility that the robot may have to deviate from the patrol route to circumnavigate obstacles. As a result, while

the robot will attempt to reach all of the points in the patrol loop, it may miss any or all of them due to obstacles. In this case there needs to be some sort of criterion for how bad the deviations from the patrol route can be and still be regarded as a satisfactory patrol. Since this criterion is most likely something related to the overall goals of the user, such as detecting the approach of some other actor to the patrol area, the failure of the criterion is unlikely to be detected by the individual robots. Instead, the detection of failure to reach patrol points should be propagated back to the user in a way that allows the user to decide if the patrol path is still acceptable.

Adding dynamism to the environment raises a concern with the modifications of TangentBug required to allow it to operate in a dynamic environment. In the modified TangentBug, a robot crossing its own path is caused by a dynamic obstacle, but in the case of a patrol, the robot constantly crosses its own path. In order to detect the path crossings, the modified TangentBug of Tomita and Yamamoto keeps track of its own past locations, and detects the intersection of its current location and next position with each line segment between each of its own past locations. However, this problem is not as dire as might be expected. Because the path following is considered an iterative application of the modified TangentBug, the buffer of previous path locations is cleared on beginning the next iteration, which is caused by either arriving at a goal or determining that the goal is unreachable. This also greatly reduces the storage requirements for the path, as not all points along the path are required to be stored, and allows the user path to have loops, which initially motivated the use of modified TangentBug.

Dispersion

Unlike formation, the primary concern of dispersion is that the robots be evenly spread throughout the area, rather than within or along a specific perimeter. Dispersion can be treated as movement to points, but the selection of points can be handled in a number of ways. If the robots can communicate and are localized, they can agree on a dispersion based on the space covered by the swarm and a desired density. As with formation, each robot could be assigned a target point to which it should move. However, in the presence of obstacles, attempts to approach the target points could be thwarted by inter-robot interference. For example, if many of the dispersion points are within an obstacle, and one is in the only, narrow corridor into that obstacle, the arrival at its assigned point of a robot assigned to the point in the corridor will block all the others from arriving at their dispersion points. Because of this problem, it is preferable to have each robot be able to determine whether the dispersion is at least locally complete, rather than rely on simple arrival at a point to regard itself as "dispersed."

Unfortunately, this change renders dispersion more complex than formation. While a single robot may have some way to detect that it is "in formation", such as being at a point within a polygon, it has no way to detect that it is dispersed, and indeed, it makes little sense to talk about a single robot being dispersed.

Dispersion is, instead, a state of the swarm as a whole, rather than the state of a single robot. If the robots can sense each other, it becomes easier, because they can at least determine if they have neighbors, or possibly how far their neighbors are away, and so can examine local conditions of dispersion. If all robots can detect that in their local area, they are well-dispersed, and all robots act to be locally well-dispersed, then the swarm as a whole will tend towards being

well dispersed. For the purposes of further discussion, the definition of "dispersion" is that for some integer threshold α and some finite sensing radius r, there are no more than α other robots within r. With α of zero, this is the same as the dispersion described as a basic behavior by Matari [Matari, 1994]. This definition has some limitations, such as regarding a single robot "swarm" as permanently dispersed. For the purposes of this argument, we will assume robots are points, but aside from some degenerate cases, the argument can be shown to generalize to robots with area. Point robots raise the problem that if α is allowed to equal the number of other robots in the swarm, then all of the robots in the swarm are permitted to be arbitrarily close to each other within r, and so the criterion will consider a state with all robots occupying the same point to be "dispersed". This is counter to the intuitive understanding of dispersal, and so α is constrained to be less than the number of robots in the swarm, other than the robot measuring α.

It may be that dispersion cannot be performed, for example due to limited space in the area to be dispersed into. In order to derive a complete local-sensing-only dispersion algorithm, the algorithm must be shown to move every robot to a location where the robot can detect using local sensing that it is dispersed, or determine that such a location does not exist.

For two robots and $\alpha = 0$, there must exist two points in the space such that the distance between those points is greater than r. Because the sensing range of the robots is r, a robot located at a point cannot tell if a point greater than r distance away is unoccupied, and so the algorithm requires some form of non-local sensing, either communication with other robots or map building. In fact, in order to determine that there exists no such pair of points in the space, the robots must be aware of all the accessible points in the space, which amounts to global, rather than local knowledge. For two robots, α may not be increased, as it would then

violate the constraint that α be less than the number of other robots in the swarm.

If the number of robots in the swarm is increased by one, the number of points in the space that are required to be greater than r apart increases by one. If α is increased by one, allowing one robot to be within r of another robot, then the problem reduces to the two robot case with $\alpha = 0$ and the third robot able to be placed anywhere. However, as the two robot case requires global sensing to determine whether there exist two points that are greater than r apart from each other, it is still not a complete local-sensing-only algorithm.

For any number of robots M and $\alpha < M$, split the swarm into a group of α robots and the remaining $M - \alpha$ robots. Because α robots may be within r of each other, put them all at the same arbitrarily selected point. The remaining $M - \alpha$ robots must be placed at points that are more than r from that point, or else the group of size α will have more than α robots within r of them. As with the two-robot case, determining that there does not exist an accessible point greater than r distance from any point in the space requires global information about the space. As a consequence, a complete local-sensing-only dispersion controller for point robots does not exist. It either requires global information, in which case it is not local-only, or it cannot determine that a satisfactory target point does or does not exist, and so it is not complete.

Making the robots take up space, rather than being points, can result in the condition that α robots simply cannot fit within the disc of radius r around one robot (for example, if the robot radius is r), and so even close-packing of the robots is considered "dispersed". Leaving aside this degeneracy, if $\alpha + 1$ robots can fit within r of one robot, and α is constrained to be less than M, then the extra robot is required to be greater than r from the surrounded robot, and the same argument holds with regard to determining if a point greater than r distance from

the surrounded robot exists or not for all possible placements of the surrounded robot.

Despite the fact that a complete, local-only dispersion controller cannot be found, there are controllers for dispersion of robots that are proven to converge, and to operate without localization [Correll, Bachrach, Vickery, and Rus, 2009]. This style of dispersion does require that the robots be able to detect and communicate with each other. With non-communicating, non-localized robots that can detect range to other objects, e.g. via intensity of reflected light, dispersion over an area can be accomplished by having all robots move to keep all objects that they can detect within a fixed range. Even simpler, robots could travel towards open space until they detect no surrounding objects and then stop, or stop if they detect no open space. Varying the detection range will the density of the swarm, both in terms of distance from each other and from obstacles.

Swarm Manipulation

One interesting minimum case for transportation is the use of granular convection to transport an object that is larger than the robots to a goal, without inter-robot communication or sensing beyond a weak repulsion from the goal [Sugawara, Correll, and Reishus, 2014]. However, this form of box-pushing has some limitations, such as the space being enclosed, and the shape of the transported object being such that it cannot become trapped against the sides of the enclosed space.

Another possible approach is the use of occlusion-based transportation [Chen, Gauci, Li, Kolling, and Gros, 2015]. The robots push on the object if the object is between them and the goal, and so occludes their view of it. By adding a moving goal, as in the case of path following via bug algorithms as discussed earlier, this transportation method can even cope with environments where the eventual

goal is not visible from the starting point. The user-supplied path, expressed as a sequence of goals, provides information on the obstacle-free route through the space. Again, this form of transportation is prone to failure in the case of the object arriving an a position where one face of it is against a wall, and so no robot can get behind it to push it off the wall. It is also only proven to work for arbitrarily shaped convex objects. Certain pathological concave objects can actually result in a net force away from the goal.

The use of simple sensors can be used to coordinate multiple robots to move a box to an illuminated goal [Kube and Zhang, 1996]. Kube and Hong used sensors to track a light on the box and distinguish the box from other, unlit boxes, which were treated as obstacles. If distinguishing obstacles is unnecessary, the robot could simply use two light sensors, and push forward if the lower one is occluded by the box and the higher one can see the goal. However, such a simple robot will push against obstacles that cannot move, and other robots.

It would be good to determine if manipulation in this style, which is to say simple pushing rather than caging or pulling manipulation, admits a complete controller as with path following and formation. To be complete, a controller would have to, in finite time, either move an object to a goal, or determine that it is not possible to move the object to the goal. In reality, it is possible that the object cannot be pushed at all, or that constraints on its dynamics render it impossible to push it to the desired location. For example, it is possible for a human to push a car with its gearbox in neutral forward or backwards, but not sideways. This human example is quite illustrative, as there are conditions where it is possible (car in neutral and human pushing on front or rear of car), and conditions where it is not possible (car in park, human pushing car sideways, car in neutral but against a curb). Asserting that the task is not possible would require testing all

gears and pushing locations, and failing in each of them. As the complexity of the environment grows, the space of possible pushing arrangements increases, possibly in ways the robot cannot detect. Additionally, some attempts may appear initially successful, but lead to failure states, such as pushing an object into a hole that it cannot be pushed back out of. Because of the potential explosion of problems with pushing, the environment must be constrained to one that is simpler to reason about, and then made more complex gradually to locate the minimal complexity that causes the creation of a complete, local-sensing-based controller to fail.

For the purposes of this argument, it will be assumed that the object can be pushed in any direction, and that there are sufficient robots present to push it. In unlimited free space, it is clearly possible to push such an object to the goal. The robot must be able to locate the object and goal in some way, but as discussed by Kube and Zhang, the required sensor precepts may be fairly simple even for real robots Kube and Zhang [1996].

In bounded free space, there is the possibility that the object can be placed against the boundary of the space in such a way that no pushing force can remove it from the wall again. For example, an oblique triangle with one face against a straight wall is pushed towards the wall by forces normal to its exposed faces. Of course, under some conditions of friction and pushing forces, the triangle may rotate away from the wall, but as the length of the wall-facing side grows, the more normal the forces become to the wall itself, and so at some point the friction against the wall will exceed the pushing force of the robots on its sides. Using only local sensing, it is possible to detect this sort of stuck condition, and indicate that the attempt has failed. If the robot detects that it is pushing the object, but the robot is not moving, then the object is also not moving. If the robot then circumnavigates the object and is able to pass from the object to an

obstacle and back, then one side of the object is against the wall. Note that this may require the robot to also circumnavigate the boundary of the space, if the object is against a wall, which may be large, but so long as the boundary is finite, such a circumnavigation will take finite time. It also assumes that the robot can detect its own motion.

In an unbounded space with obstacles, the object may be able to be pushed to the goal if there is an unoccupied path that the object can pass through in order to arrive at the goal. While having a clear path is a necessary condition, it is not sufficient. For example, if a square object is being pushed down a corridor that is exactly wide enough to admit the square, and has a 90° bend in it, the square will fill the bend and become stuck, as no pushing force on either of its exposed faces will move it in either direction along the corridor. Nonetheless, the corridor is wide enough to admit the square, and so even if traversing the corridor is required to get to the goal, it does not violate the condition that there be an unoccupied free path that the object can pass through in order to reach the goal. Further, being able to detect such a path requires being able to check assertions about distances across sets of points throughout the space and the shape of the object, and so global, rather than local, sensing or knowledge.

By adding various levels of communication, more complex sensing, and global planning to the capabilities of the robots, autonomous box pushing can be performed under different conditions. A review of these techniques is beyond the scope of this section, but for recent results see [Tuci, Alkilabi, and Akanyeti, 2018; Rahimi, Gibb, Shen, and La, 2018; Alkilabi, Narayan, and Tuci, 2017]. Occlusion-based transportation, on the other hand, is effective with minimal sensing, and so at an advantage for simple and inexpensive robots. Rather than attempting to construct a local-sensing based complete path planner, the system can take

advantage of the user input as a suggestion of the best path along which to move the object. Users frequently positioned robots near the object or used a dragging action of the box itself in the interface, so the path for the robots to follow is defined as the list of points that the user dragged over. Each individual robot's program can be based on a bug-style algorithm, using local sensing information to determine what the robot should do. However, since determining that the box can be pushed all the way to the goal or terminating with failure would require non-local knowledge of the world, this algorithm is not complete. First, the robot must arrive at the box. Assuming the user has provided the location of the box, motion to it can be handled as with path following.

Once the robot is at the box, it alternates between two behaviors, edge following and pushing. It starts with edge following, which has three exit conditions. If a robot detects that the line from the robot to the goal has gone from not passing through the object to passing through the object, then the robot stops circling the object and begins pushing against it. If a robot goes from the surface of the object to the surface of an obstacle and back while edge following, then the object has been pushed against an obstacle, and the robot stops because pushing has failed. If a robot returns to the point from which it started circumnavigating the object, then the robot detects that the object has stopped and is over the goal, so the robot stops because the pushing has succeeded.

So in the edge following case, there are three exit conditions, two of which end in termination of the program. The third exit condition causes the robot to transition to pushing the object. Pushing has two exit conditions, detection of stagnation or the m-line no longer pointing into the object. The m-line no longer pointing into the object is caused by the object being pushed over the goal by the robots. This is the weakness of this algorithm, as with a small enough collection of

robots, such as one robot, the robot will push the object until the robot arrives at the goal, rather than being stopped by robots pushing from all sides resulting in the forces on the object being balanced. Once the robot arrives at the goal, the m-line no longer points into the object, so the robot will circle the object until the m-line points into the object, push it across the goal, and repeat this process. As a result, the algorithm becomes more stable and more likely to end with the object on the goal as the number of robots increases.

Stagnation occurs in one of two cases: either the forces exerted by all robots are balanced, and so the object is not moving, or the object is against an obstacle, and so the object is not moving. Stagnation is detected simultaneously by all robots, under the assumption that the object is rigid. Non-rigid objects are significantly more complex to manipulate. When stagnation is detected, all robots begin circling the object. Since the object is not moving, because all robots have stopped pushing it, the robots will all either detect that the goal is within the object, as above, and stop with success; detect that the object is against an obstacle, and stop with failure; or find a point where the m-line transitions and begin pushing the object again.

If a new robot arrives while the robots are circling, it has no effect, as it will initially also begin circling the object. Since the other robots began circling because of stagnation, if there is a point that the m-line transitions, then the stagnation was caused by an obstacle, and so the new robot will either detect the obstacle and stop, or push the object against the obstacle as the others had been doing. If there is no point where the m-line transitions, then the stagnation was caused by balanced forces, and so the new robot will detect that the goal is inside the object and stop.

Completeness Under Poor or Absent Localization

The majority of the user tests were done with a visual representation of robots that can localize, and so their locations, at least relative to each other and nearby objects, could be displayed to the user. The cloud/unknown number of robots case, however, described robots which are not able to report a location that can be mapped into a useful display for the user. Unfortunately, lack of localization implies the lack of a number of other mechanisms. As discussed in Section 7.1, local coordinate systems can be derived from range sensing, bearing sensing to other robots, and of course combinations of range and bearing sensing. As a result, robots that cannot arrive at a local coordinate system also cannot sense each other's range or bearing, or they could use that information to derive a local coordinate system. While the robots could measure distance, using their own odometery, they cannot know the distance to anything that they haven't been to already, and cannot store locations such as hit points, which are used in bug algorithms.

Even if the localization is not absent, bad localization can eventually become a problem for correctness of bug algorithms. If the localization is assumed to be noisy, then returning to the hit point while circumnavigating an obstacle becomes returning to within close enough to the hit point, according to the sensor (assuming gaussian or similar noise). However, if the threshold is too wide, the robot may get close enough to the hit point to think that it has completed the circumnavigation, when it has really almost completed the circumnavigation, but missed a small door in the obstacle, which is not circular but actually an almost-closed, "C" shape. In this case, the robot will incorrectly determine that the goal is unreachable. The robot could also fail to detect returns to the hit point, especially in the presence of biased noise or localization failures, and so would never leave the object. Sensor

noise is, in general, a problem for bug algorithms [Ng and Bräunl, 2007].

Pheromone approaches are tempting, as they use gradients of intensity rather than actual location to represent information about the location, but they present a conundrum to the assertion that the robots cannot localize [Taylor and LaValle, 2009]. If there exists an object in the world that the user can precisely control and know the position of, such as a pheromone emitter, then why can the user not just use that to either do the task or bootstrap a coordinate system? Conversely, if the user cannot accurately place the transmitter or pheromone emitter, then how can they hope to use it to control the swarm?

However, there is a line of work in robotics that uses non-metric maps, such as sequences of landmarks, which consist of sensor precepts, possibly classified into groups, and motions or connections between them [Mataric, 1991; Franz, Schölkopf, Mallot, and Bülthoff, 1998]. As the available technology has developed, the landmarks have moved from rings of sonar sensors to include representations derived from 360° photographs [Tapus and Siegwart, 2005; Goedemé, Nuttin, Tuytelaars, and Van Gool, 2007]. The integration of such a graph-based representation with a user interface is a very interesting topic, and one that does not appear to be discussed in the literature until very recently [Landsiedel, 2018]. The extension of topological maps to include semantic information allows the robot to operate under directions that are very intuitive to humans, such as "Take the first left after the Dunkin' Donuts," rather than "Go 32.5 meters, stop, and turn -1.02 radians." The robot's view of the space could be presented to the user in a number of ways, such as a chronological list of places visited, and attendant images of them, or a flexible graph of the related images and the topological connections between them. If the representation of landmarks is such that each robot will derive essentially the same representation at a given point, then such a graph extends naturally to

multi-robot navigation. Each robot builds its own map, but by sharing maps and finding overlapping landmarks, they may gain the ability to navigate to places that they have never been, but were visited by other robots.

In the interests of completeness, the next few subsections explore the possibilities for performing the user experiment tasks under a non-metric map. While topological maps do represent a form of localization, it is not a form that can be converted into a representation to a user in the same way that metric localization would permit. The map represents locations and connections between them, but not angles and distances.

Motion to a Point

In an interface presented to the user as a graph with the landmark points at the nodes, return to a known point is planned using graph search over the map, and executed using visual servoing or known sequences of actions to move from node to node.

Motion to a point outside of the areas the robot has explored becomes difficult in a topological map. Indeed, since there are no landmarks in unexplored area, all unexplored area is the same to the robot. However, there are some methods of potentially moving to user-commanded areas that have not been explored. If the robot has come within sensor range of the area, then some sub-segment of a 360° image can be used as a representation of the target. When the robot returns to the area that it saw that image, the matching features in the 360° image from that area provide the heading to move towards the target, and so the robot can, essentially, servo towards something it has seen before, but did not approach. Even if the robot has not seen the target, if the user has an image of it, the robot can explore areas it has not seen before, looking for the target. The topological map

gives the robot the ability to recognize areas it has been before, and avoid them while searching for something that it has not seen before.

This sort of searching assumes that the interface can send a description of the desired target region. For humans, this kind of navigation is quite common. To return to the example from before, in the instruction "Take the first left after the Dunkin' Donuts," there is no need for the person issuing the direction to offer an image of the specific Dunkin' Donuts that they mean. As long as both people have enough overlap in their idea of what a "Dunkin' Donuts" is, the instruction is understandable.

Path Following

While the user cannot see the exact path taken by the robots, as the map does not have a metric space for the points of the path to be defined in, the robot can still follow user-specified paths in the graph. The path could be represented by a sequence of desired sensor precepts and actions, similar to human directions like "Go past Mark's office and turn right after Terry's office." As with moving to a point, motion along a path that goes outside of the map is problematic. Since unexplored regions do not have a known connectivity, such directions might be specified incorrectly. To return to the previous example, Terry's office might not be past Mark's office, relative to the position of the robot, and so a route that passes from the robot's location, to Mark's office, and then to Terry's office does not exist. In the case of a metric map with possible obstacles, some subgoals might be able to be dropped so that the system can move on towards a known final goal. The metric quality of the map allows attempts to move from arbitrary locations to others, by including information such as direction and distance, but in a sequence of desired sensor precepts, if one precept is not detected, such as "Terry's office", there is no

clear method to recover.

Formation and Patrol

As with path following, in a non-metric map, the idea of forming a specific shape, such as a square, becomes somewhat ill-defined. After all, angles and distances are defined in a metric space, and so while the robots might be in a square, the map might not display them that way, and they might not be aware of it themselves. However, certain formations that could be defined over a non-metric map may still have some utility. For example, formations topologically equivalent to circles could be formed by occupying cycles on the map, and so regions of the map could be segregated from others by an enclosing ring. By looping over such a cycle, the robots could be said to be patrolling the perimeter of a region, in the sense that all nodes adjacent to the patrol area would be regularly visited by a patrolling robot. Similarly, if a "front" has to be guarded, the robots could be commanded to positions such that they divide the graph, and indeed, choke points could be detected to optimize the guarding of regions.

Dispersion

Dispersion is one of the more important abilities for robots using a topological map to have. In order to form the map, the robots would have to move through the space and explore it. Assuming the robots have some form of range sensing, as would be used for obstacle avoidance, they could simply scatter stochastically, choosing random headings and following walls while avoiding locations that they have seen before. The fact that the robots have the ability to recognize places and so avoid locations that they have visited before means that they can be quite efficient at spreading over an area. Assuming the area is closed, a traversal of it can be

represented as a tree. The robot breaks any cycles that would appear in the traversal by refusing to return to nodes that it has visited before, unless it is backtracking to a node that has areas the robot has not visited before near it. For this sort of traversal "areas the robot has not visited before" could simply mean areas in a node/location that were not detected to be occupied by obstacles. While traversing the area, the robot would enter areas it has not visited before, possibly creating new nodes/locations, and keeping track of unvisited areas to potentially visit in future. Such a tree search for a single robot is effectively depth first search, and is linear in the size of the graph (in this case, the area to be explored). Some bound such as graph distance from the starting point might be desired, as performing a depth-first search in an open environment could lead to runaway robots. Adding other robots that can share maps allows the area to be explored in parallel. As each robot receives updates to its map from other robots, the areas that they consider unvisited will be removed by the other robots visiting them. Once such a map is generated, the robots can disperse over the map either by agreeing on a dispersion, such that each robot only has some fixed number of robots in adjacent nodes, or simply moving so that no node of the map is occupied by more than one robot.

Swarm Manipulation

Because a purely topological map does not have metric information, reasoning about the dimensions available over path to push an object on is not possible. If the map were enhanced by metric information, even just the narrowest point along each edge and node, then graph search techniques could be used to determine if pushing a box along a path were impossible, although not to determine if it were possible, due to the 90° bend problem discussed above. The problem of topological mapping without localization is somewhat orthogonal to the problem of pushing a

box, as the topological map may be able to answer some questions about pushing an object, but does not provide useful information on the local interactions required to actually perform the pushing. In the most abstract case, an intangible 360° camera with mobility can perform all of the other tasks listed above, with the same ease as a mobile robot, but clearly cannot push anything, so this kind of mapping and control does not provide much information to guide swarm manipulation.

However, since the approaches that use 360° cameras have the ability to use the camera information to provide a heading towards a given node, and the nodes could be chained to provide a path, a combination of information about which nodes and edges are locally obstacle-free and occlusion-based transportation could be used to guess at a path that is locally obstacle-free over its length, and a pushing direction for each segment of the path. Such a controller would not be complete, particularly in a dynamic environment, nor totally local, as it relies on a global map, but the task could at least be attempted without metric localization.

Chapter 8

Interface Implementation

The user interface for the control system uses the Kivy window management library. The UI uses ROS for its infrastructure, with the various components implemented as ROS nodes and communicating using ROS messages. The user interface layer receives images from the robot arena overhead camera, and displays them, appropriately overlaid with buttons, to the user. The topmost layer receives user interactions in the form of contacts, and converts them to ROS touch events. The multitouch device emits contact points that contain their location on the screen in cartesian coordinates, as well as identifiers for each contact. Kivy's multitouch device handling adds other event information, including whether the contact is a double or triple tap, when the touch was last updated, and whether the event has ended (the finger has been lifted), how far the touch has moved since the last update, and whether the touch is associated with the activation of certain hardware, such as the mouse scrollwheel. The majority of this metadata is not used. For process in later layers of the gesture recognizer, only the start, end, and update times, whether the event is a double or triple tap, the location of the point, and

its identifier are passed as a ROS message. Button presses are also converted into ROS button messages which contain the time of the button press and the name of the button which was pressed.

The user interface layer publishes touches, which correspond to single points of contact in both time and space as observed by Kivy. In order to collect points into strokes, a degree of cleaning of the input is needed. The primary source of error in strokes is stick-slip motion of the user's finger on the screen. When the user's finger slips, sometimes the screen regards the finger as having left the surface and returned quickly, breaking the stroke into two separate gestures. The ROS node destutter.py maintains a dictionary of lists of touches in the order that they arrived, indexed in the dictionary by the touch ID assigned by the user interface. As each stroke ends, it is compared to the other ended strokes. If the space in time and distance between the end of the recently completed stroke and the beginning of another stroke is small enough, the two strokes are merged into a single stroke. The parameters for the space and time thresholds for merging were set empirically, by intentionally causing stuttering contact with the screen and examining the reported time and distance between the beginnings and ends of contacts. At present, if the beginning and end of successive strokes overlap by less than 0.01 second, or are less than 0.15 second apart, they are considered candidates for merging. To be merged, candidates must also end and begin no more than 80 pixels apart. This value is approximately the width of a fingertip on the 3M screen, and is greater than the observed distance between stuttering strokes.

Because a currently-ended stroke might be merged with a stroke that hasn't begun yet (that is to say, the user's finger might be in the air during the stutter of a stick-slip movement), strokes that have ended are not published unless there is no in-progress stroke that they could possibly be merged with. Every 0.2

seconds, the stroke cleaning node checks all the ended strokes against all the in-progress strokes. If the ended stroke's endpoint is closer than the distance threshold from the beginning of all of the active strokes, it could be merged with the active stroke. If the ended stroke is close enough to an active stroke's beginning to be considered, the time threshold is checked to determine if they could possibly be merged. If there is no active stroke that the current stroke could possibly be merged with, once that active stroke ends, then the current stroke could be published. If the current stroke's end time is greater than the merge gap threshold in the past, then it is impossible for a new stroke to begin in time to be merged with the current stroke. Since there are then no active strokes it could be merged with, and it is impossible for a stroke to begin in time to merge with it, the stroke is then published.

Gesture Recognition

Strokes are recognized as gestures by a set of separate recognition modules. The initial design concept had been to have successive layers of recognition modules add more and more abstract information to the gestures, and arbitrate cases of ambiguity in gestures, but as the selected gesture set was intended to minimize ambiguity, this arbitration lead to additional complexity without any real gain in functionality or ability. It should also not be assumed that these gesture recognizers are the best method to perform the task, but they were sufficient during development, and investing significant time in improving them was beyond the scope of this project.

If a gesture has fewer than 10 points or is fewer than 10 pixels across, it is classified as a tap. This classification is used because the user's finger distorts as it presses on the screen, and so can be registered as motion over a very small distance, even though the user was not dragging their finger. Gestures are classified

into taps, lines, arcs, or closed shapes by the angle between the beginning and end of the gesture around the centroid. The thresholds for the angle around the centroid were <1 radian to be classified as a circle, <2.5 radians to be classified as an arc, and 2.5 radians or greater to be classified as a line. This sort of recognizer has problems with, for example, a circle drawn with a very tight scribble of points at one end. Placing the centroid in the center of the bounding box of the shape would result in it being recognized, correctly, as a circle, but the actual centroid as calculated from the points would be placed close to the dense knot of points at one end of the line, thus skewing the angle around the centroid.

Gestures are then compared to information about the location of robots relative to the gesture on the screen to determine the meaning of the gestures. The gesture detectors are lasso and box selection, paths (lines that are not lasso or box selection), tap selection, tap as waypoint, and dragging of individual robots.

The lasso gesture detector receives gestures from the gesture classifier and from the AprilTag detection node. If the gesture starts on a robot, it is considered a drag, where the user puts their finger on one robot, and drags from that robot to another location. If the gesture is not a drag, but is a closed shape, and there are robots located inside it, then it is a lasso selection. The lasso select gesture recognizer initially attempted to fit an oval to the points of the closed shape, but this approach was rejected because it does not work well if the user attempts to draw a concave polygon, such as a banana shape, in order to select some robots and not select others from a group of robots. Instead of using a fitted oval, the closed shape is treated as a polygon, and all robots inside the polygon are considered selected. If there are no robots, it is treated as a path, the generic classification for gestures with no other classification. If the gesture is not a drag, but is a line, it could be a box selection or a path. A box selection is a line whose bounding box

includes at least one corner of an AprilTag. This requirement for inclusion, rather than including only AprilTags with their centers, or 3/4 of their points inside the bounding box, was chosen because the user survey on inclusion in selections that intersected the robot indicated that erring on the side of including partially-selected robots was preferable.

If a path begins on a robot, it is the "drag robot" gesture, which is typically used by experiment participants to move an individual robot. The criteria for deciding that a path begins on a robot is the same as that used for deciding if a tap is intended to select a robot, as described below.

Tap gestures are handled separately from strokes. If a tap gesture is on a robot, it is treated as selection of that robot. Taps that are not on a robot are treated as potential waypoints. For the purposes of making this distinction, a tap located within 80px of a robot is considered on the robot. The value of 80 pixels was chosen because it is the approximate width of a fingertip on the interface device, and so should be adjusted for devices of different resolutions. Alternatively, the mapping of pixels to real-world dimensions provided by the AprilTags could be used to calculate a conversion factor between the size of the screen and objects displayed on it in pixels and real-world meters, and the distance specified in terms of that conversion factor. Doubletap is the end-of-command gesture, and so is not treated as a waypoint or as a robot selection gesture.

The gestures are then passed to the robot program generator, which adds them to a stack as they arrive. When the user terminates the gesture by either issuing an explict end-of-gesture double-tap, or by issuing a new selection gesture, which ends the previous command and begins a new one.

Translation Into Programs

At the outset of this work, it had been hoped that there existed some form of transformation from the language defined by creating a formal description of an unambiguous subset of the user gestures to a potential language of robot behaviors. While the language of robot behaviors is itself not terribly well-defined, approaches such as the flavors of AutoMoDe and Supervisory Control Theory hint that the output language would likely be able to be represented as a DFA or PA, and so the resulting programs could then be amenable to analysis using a model checker such as PRISM [Kwiatkowska, Norman, and Parker, 2011].

However, the user gesture language as defined by this work is actually quite vague, when it comes to commanding a robot to perform the expected actions. Alan Perlis has been quoted as saying "When someone says 'I want a programming language in which I need only say what I wish done,' give him a lollipop.", but regrettably, the user gesture language is just such a language, in part due to the design of the experiment [Perlis, 1982]. Users were told to assume that the system was capable of understanding their orders, so they merely had to indicate what they wanted the system to do, and it would then do it. A system where the computer does what the user wishes done is more difficult to implement than the recipients of lollipops expect, because they rely on a large amount of *a priori* information shared between the person issuing the command, and the system executing it.

For example, in the box-moving task, the users would frequently move the robots to surround the box, and then move the robots to the goal area. However, one might expect that robot control programs would attempt to avoid obstacles. Without the knowledge that boxes are acceptable to push against, no motion of the box would occur, because the robots would treat it as an obstacle to be avoided.

Even this knowledge shows the limitations of the gesture as a way of conveying a program to a robot. The user data set does not have clear gestures for conveying that box-pushing is desirable, how to recognize the presence of a box, how to tell one box from potential other boxes, or how to convey that any particular object can be pushed, rather than just boxes. Instead, users assumed that the robots understood, as the user did, that the box was a thing that could be moved, and so did not have to be told.

Because the user gestures did not convey all of the information required to perform tasks, there is not a transformation that could operate purely on the user input to produce a program as output. Instead, the output program combines algorithms chosen by the system developer with parameterization from the user input. Because the system was intended to operate in a potentially unknown environment, the bug, dispersion, and occlusion-based transportation algorithms discussed in the previous section were used. If the environment were known, or communication were assumed to be reliable, other algorithms could be used. Indeed, the translation layer could be modified to switch algorithms based on the parameters of the swarm it is creating programs for and their environment.

The development of the translation layer was performed in a manner similar to a compiler, which permitted the planning and other algorithms to be built into output programs by the translation layer. The input language was the user gestures, including which robots were selected and which user-specified paths were created. These inputs were used to parameterize the chosen algorithms, and the robots selected were used to determine the distribution of the resulting programs. As a result, this work does not end up breaking away from iterative hand-coding, it just moves it from being done as a way of controlling the swarm, to being done as part of the creation of the control interface. For the motivating example from the

introduction, urban search and rescue, this is acceptable, as it does not require the end user to program the swarm. It is also somewhat risky, as the resulting system may not have the flexibility that end users require.

Implementation Details

User gestures arriving at the translation layer are stored in a stack until an end-of-command gesture arrives. The gesture sequence is then translated into a program that is parsed by the Lark parser library. Lark is an Earley parser, and so can parse all context-free grammars, although the current gesture language is not sufficiently complex to actually need this level of power. The resulting parse tree is then walked to generate GCPR programs that implement the algorithms described in Chapter 7.

Basic movement to points is implemented using the variant of TangentBug. Path following and formation combines the basic movement to points with a sequence of GCPR instructions that implement a program counter, and set the goal based on the program counter. As each point is either reached or determined to be unreachable, the goal is advanced to the next point. For motion along paths, the goals are set to points along the path, ending at the final goal. Formation allows the robot to stop at reachable points on the formation, but not unreachable points.

Patrol also uses modified TangentBug, but instead of terminating when the program counter, and so the goal, reach the final position, the program counter and goal are reset to the start point of the patrol. As a consequence, the resulting program intentionally contains an infinite loop, but it can be interrupted by assigning a new program to the swarm.

Dispersion is implemented using a minimalistic range sensor. Each robot can detect if there are other robots within a fixed range. If there are more than

two robots in the range, the robot moves forward, avoiding obstacles. If there are exactly two robots in range, the robot stops. If there are less than two robots in range, the robot executes a U-turn and drives in a straight line, avoiding obstacles, until one of the other situations occurs. This algorithm is a GCPR implementation of the α-algorithm of Winfield *et al.*, and so shares its strengths and weaknesses [Winfield, Liu, Nembrini, and Martinoli, 2008]. Notably, the algorithm is not certain to prevent the separation of the swarm into subswarms that are not connected. More sophisticated programs, possibly using more communication, can prevent these issues, but since the requirement of the behavior is that the robots disperse, rather than that they maintain a particular level of network connectivity, there is no need for this particular implementation to enforce connectivity. Indeed, simply moving the robots to random locations uniformly selected would "disperse" them. However, selecting points would require foreknowledge of the area to disperse into. The α-algorithm was chosen instead because it can operate in a previously unknown area, and because the resulting distribution looks even, visually. A randomly selected set of points from a uniform distribution may place two robots right next to each other, which, while "uniform" in a statistical sense, would appear uneven to the user, and not satisfy their intuitive understanding of dispersion. If dispersion in swarm robots continues to be a problem of interest, it is likely worth investigating the tradeoffs between speed of convergence, quality of dispersion, and user satisfaction with various methods.

Manipulation was implemented as a simple version of occlusion-directed transport. If the robot is not near the target object, it attempts to move to the target object while avoiding obstacles, using the modified TangentBug algorithm. When the TangentBug algorithm detects that the goal is unreachable, because it is inside of the mobile object, the program switches to occlusion-based manipulation.

The robot wall-follows around the object until a line from the robot to the goal intersects the object, and then pushes in that direction. While in this mode, the robot continually updates the direction of the goal, and switches between getting in position and pushing the object, as needed. As with the original occlusion-based manipulation, this algorithm is ignorant of obstacles on the opposite side of the object from the robot, and so can get stuck. However, as the system can accept a path for the object to be pushed on, the user can attempt to specify a clear path for the robots to move the object along.

Interpretation of Programs

The robot algorithms were implemented as GCPR programs, and interpreted by a separate process for each robot.

The GCPR interpreter starts with a single-line GCPR program that halts the robot, so all robots are stationary until they receive a new program. Programs are sent to each robot as ROS messages. When a robot receives a program, it begins executing it immediately. The GCPR interpreter runs at 100Hz, so 100 times per second, it evaluates the guards of each GCPR tuple, makes a list of all the commands whose guards evaluate to be true, and executes each command. The GCPR interpreter also implements a number of convenience methods such as is_near_front(), which returns true if any of the sensors on the front of the robot detect an object near them. The convenience methods are boolean, and intended to be used to create guards by boolean combinations of convenience methods with each other and with direct assertions about the sensors or other states of the robot. These methods are regarded as conveniences because they could be implemented as direct boolean statements about the sensor information of the robot, but the

resulting GCPR statements would be extremely verbose.

The interpreter also contains the implementations of the commands for the GCPR statements. The complexity of a command in GCPR is not limited. Because the system operates in the real world, at some point commands must result in changes to the state of physical actuators, and so the most atomic interactions might be simply the setting or clearing of given digital outputs. However, there is a trade-off between the complexity of the resulting commands and their utility to human users for writing programs. Carrying the complexity of the GCPR commands to the other extreme, the entire program could be implemented as the GCPR tuple (true, move_to_goal(), 1.0), which, under any condition, with probability 1.0, calls a command that moves the robot to the goal location. The resulting GCPR is quite compact, but the complexity is moved to the command implementation, where it becomes difficult to evaluate with formal methods.

In seeking a balance between excessive conciseness in the expression of commands and excessive complexity in their implementation, the current implementation provides basic motion in lines, arcs, and rotation in place, and turning to face a given direction. There are also a number of methods to set and clear variables such as hit points, used in bug algorithms, and program counters, used in a number of GCPR implementations. All other complexity of the commands is contained in the GCPR statements which combine and control the motions, rather than in the motions themselves. For a future system, it may be more interesting to implement the swarm primitives suggested by McLurikin or Nagpal as commands themselves, and then allow more complex GCPR programs to use those commands [McLurkin, 2004; Nagpal, 2004]. However, it then becomes incumbent on the implementer to ensure that the command implementations are correct. Having simple motions as commands eases correctness checking, as the motion will be performed, under the

assumption that the motors are functional and the robot's path is unobstructed.

If the commands are only outputs, the flow of information within the GCPR interpreter also becomes somewhat clearer. The sensor data is updated via callbacks triggered by ROS sensor information messages arriving, and is then used to evaluate the guards. Depending on which guards are true, commands are called to produce output. If the commands themselves are required to operate on sensor precepts, then the result of calling a command can change depending on the sensor information available when the precept is called. Increasing the complexity of the commands also increases the time they take to execute, and so may present issues for operating the GCPR interpretation loop at 100Hz. The current implementation can very quickly set the desired motor speeds for a given motion and then return execution to the next command. All of these factors argue in favor of keeping the GCPR commands close to motor primitives for the robot system, and keeping complexity in the control program in the GCPR code, rather than the commands.

Interface Testing

In order to test the gesture recognition and translation components of the system, the recorded participant gestures from the gesture collection experiment were played back into the gesture interpretation pipeline. There exist two major ways that user gestures can fail to pass through the pipeline and result in a robot program. The first is that a gesture can fail to be recognized as what the user intended it to be. The gesture recognizers do not discard gestures as unrecognizable, but a stroke that does not meet the criteria for a drag, box selection, lasso, or any kind of tap is regarded as a path. If a user makes a selection gesture such as a box select, but the system does not recognize it as a box selection, then any gestures following it will

not have a set of robots to be applied to, and so the sequence of gestures will not be able to be converted into a program. The second failure mode is that the user gestures are recognized correctly, but do not form a sequence that can be converted into a program.

The first kind of failure, recognition failures, was detected by replaying the recorded user gestures and recording the output of the gesture recognizers in response to those gestures. The recognitions were then compared to the human coding for the user and task that generated the recorded input. Under the assumption that the human coding is correct, if the recognizer output differs, it indicates a defect in the recognizer. However, it is important to note that the participant input gestures were not used to train the gesture recognizers, and that some participant input gestures were not implemented in recognizers. For example, sweeping gestures using the edge of the participant's hand were not part of the resulting gesture language, and so no recognizer exists to recognize these gestures. Failure to detect such a gesture is expected.

Detection of the second kind of failures, invalid gesture sequences, was detected by recording the reaction of the parser to the sequence of gestures created by the recognizers. It is possible for a sequence of gestures to create multiple parse attempts, by having, for example, a selection and path gesture followed by a second selection and path gesture. The second selection terminates the first command and begins the second, and so the first parse attempt operates on the first selection and path, and the second parse attempt operates on the second selection and path. In cases where a gesture sequence resulted in multiple parse attempts, each parse attempt was treated as a separate trial, rather than allowing one failure to cause the entire task to be treated as a failure.

Due to a hardware misconfiguration during the user test, multitouch

contact points were only recorded during the first thirty of the human subject tests. For those thirty tests, any example gestures that the participant made were removed to prevent failures caused by the participant proposing alternative methods during a sequence of gestures, and so creating redundant gestures within the sequence. As discussed above, a gesture to end the command was required in cases where a new command did not end the previous one, and so each sequence of user gestures had a end-of-command gesture added to its end. Any end-of-command gestures that already existed in the gesture set were removed. Removing these gestures allowed the sequences of recognized gestures to also be passed into the translation layer without incorrectly terminating the gesture sequence.

Not all of the participants made gestures that would be expected to be recognized by the gesture recognizers. The implemented recognizers recognize box selection, lasso selection, tap selection (taps on robots), tap to set waypoints (taps on empty space), and drawn paths. As discussed above, the other functions had limited agreement between users, and so were implemented as buttons on the UI. Because user UI interactions were in various locations around the screen, they are not able to be directly converted to UI interactions with the interface as implemented. However, as UI interactions only constitute 3.4% of the recorded interactions, this is not a large loss.

Tasks where the user interactions fell outside of what the UI would be expected to recognize serve as negative examples. They are unlikely to be correctly recognized, and so should not form acceptable sequences of commands. Despite the fact that this constitutes a failure, it is a desirable result, because if they are recognized and converted to robot programs, they are unlikely to accurately capture the user's intent.

Causes of Failed Recognition

The gesture recognition elements of the user interface were not the primary thrust of this research, and so were primarily implemented in order to be able to provide gestures to the gesture compiler to attempt to generate programs. The effort to implement gesture recognition from scratch was undertaken because at the time, it was felt that collecting, assessing, and potentially training available gesture recognition systems was likely to take more time than implementing a simple recognition pipeline. It is difficult to assess the percentages of correct gesture recognitions as opposed to incorrect ones, as the ground truth data, the coding of the gesture experiment data by human coders, uses different categories for describing some of the gestures than the recognition system was developed to target. For example, a pinch gesture is frequently recognized as two drag gestures. In the single robot case, for the task of bringing the robots together in a group, two robot drag gestures would bring the robots together, and so have the desired result. However, the recognition is still incorrect, as the gesture was a pinch, which the recognizers do not recognize because it is not a part of the gesture language. Similarly, the coding does not distinguish between drags starting on a single specific robot and drags starting within the robot swarm, although their context of other actions frequently makes the user intent clear. The recognition pipeline does distinguish drags starting on a robot from drags starting anywhere else, but because of a threshold in how close the start point has to be to the robots being on it, a drag that visibly does not start on a robot (and so may be coded as a non-robot drag) could still be recognized as a robot drag because it was within the threshold. However, despite these difficulties, some causes of failed gesture recognitions were sufficiently common to merit description below.

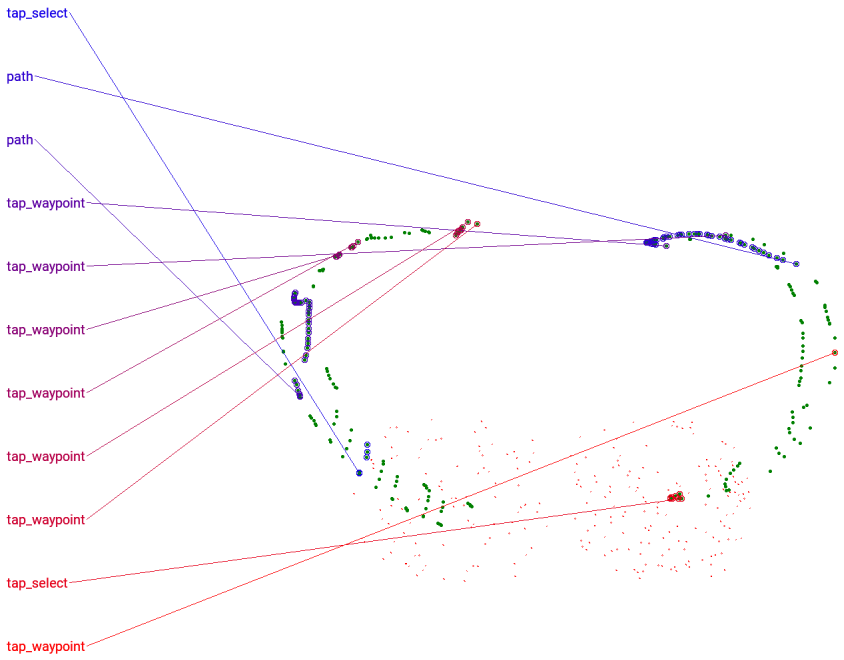


Figure 8.1: Result of a failed recognition, showing laddering between strokes and the system's guesses at the classes of each gesture.

There are a number of causes of failed recognition of gestures when the recognizers are provided with recorded participant inputs from the experiment. One cause is that the gesture made by the user is simply not one that the gesture recognizers can correctly recognize, and so any recognition they generate is incorrect. Because the pinch gestures and the 5-fingered scattering motion used to disperse the swarm by some participants are not implemented in the gesture recognizers, the pinch gesture is usually recognized as either two separate paths, or as two attempts to drag robots, or one of each. The 5-finger scatter is similarly recognized as 5 separate gestures, which may be paths if they are not over robots, or gestures to drag individual robots if they are.

When participants made gestures that had broad contact areas, the multitouch screen rendered them as ladder-like lines of nearby points. In Figure 8.1, the parallel lines of green points, especially visible on the left side, are points

reported by the multitouch screen. They do not, however, represent the location of the user's fingers. The user was using two fingers, located close to each other, to make the two arcing gestures visible in the data. Because the two points were located too close to each other to be reliably detected as separate points, the screen reported a single, interpolated location instead of the location of each fingertip.

Also apparent in the output in Figure 8.1 are substantial gaps in the lines of points constituting the participant's gestures. These are locations where no points were recorded, likely due to stick-slip motion of the participant's finger, or a very light touch resulting in only intermittent detection of contact. The destuttering functionality in the earlier stages of the gesture pipeline was intended to eliminate these gaps, but it has fixed thresholds for how large the gap can be before the strokes are considered separate. Raising the thresholds so the gaps can be larger, in space or time, increases the risk that unrelated gestures will be erroneously combined.

The stutter and interpolation-induced separation of the participant's gestures into multiple strokes results in the detection of multiple gestures for a sequence that the participant intended as one gesture. On the right-hand gesture of Figure 8.1, the tap selection gesture is followed by a tap-waypoint gesture, even though they should have been regarded as parts of a common gesture. Tap selection followed by tapping a waypoint, and then the end-of-command gesture would constitute a valid program, and so there is a possibility that stutter and laddering could result in valid, but undesired, input.

In addition to spuriously-detected gestures, stutter can cause gestures to not be detected. Lasso detection requires that the angle between the two ends of the gesture around its centroid be relatively small. If a section of the points at the beginning or end of the gesture is missing, the angle may be too open to be

detected correctly as a lasso. Obviously, a missing range of points anywhere else on the lasso could also break it into two separate arcs, neither of which would be sufficient to be considered a lasso on its own.

In addition to light touches being regarded as intermittent contacts, another source of noise in the participant inputs was participants resting their fingers on the edges of the screen. This behavior was not common, and was generally coded by human coders as an action that was not intended to be an interaction with the screen, as if the user had accidentally bumped it. However, multitouch screens are prone to the Midas Touch problem, first seen in gaze-based UIs, where any contact on the screen (or any gaze direction) is considered an input for the computer [Jacob, 1990].

A more subtle Midas Touch problem afflicted participant attempts to use group selection. There is a possible ambiguity between group selection gestures and single-robot drag gestures. If a user places their finger on a single robot, and drags a circle starting at that robot, the gesture is interpreted as a command to have the robot drive around in a circle. However, if the user draws a circle around a group of robots, the gesture is interpreted as a lasso selection of that group of robots. Because the robots might be depicted as small on the screen, and so the user might not hit them precisely with a fingertip, it is possible to begin a robot drag gesture approximately a fingertip-width off the robot as well. However, this threshold led to a great many of the participants' attempts at box selection being interpreted as commands to drag a single robot through the group, and attempts at lasso selection to be interpreted as commands to have a single robot move around the outside of the group. This error also went the other way, where attempting to draw a figure, such as a box, in an area where there were already robots, would be recognized as a lasso selection, rather than a box. Combined with stutter, the problem becomes even

worse, as a single box selection could be broken up into multiple drag commands issued in rapid sequence. As selection finishes one command and begins another, each of these should be treated by the gesture translator as a complete and valid program, leaving a trail of chaos through the middle of the swarm.

Similarly, multiple robots could have overlapping areas that would cause a tap between them to be regarded as a tap *on* one of them, and so a selection of that robot. As a result, it may become impossible to command an individual robot to move to a new position within the swarm, as tapping the target location would simply select a nearby robot.

To compare the recognized gestures to the data from the user experiment, the gestures from the user experiment were cleaned of example actions, leaving only the gestures that the users intended as their primary means of controlling the robots. A comparison of the counts of the detected gestures and their corresponding gestures from the coded data can be found in Table 8.1.

UI gestures, "other" gestures, voice commands, and pinch gestures were not captured by gesture recognizers, and so are not directly comparable to anything produced by the gesture recognizers. The gesture recognizers detected 2786 total gestures, while the non-example user gestures only accounted for 1541 gestures. Stuttering is likely the cause of this drastic increase, as it frequently broke single gestures into multiple gestures. As an extreme example, the rightmost gesture in Figure 8.1 shows a single two-finger drag being recognized as a path, three waypoint taps, and a select tap, for a total of five detected gestures from one real gesture.

The recognizers make a distinction between drags starting on (or near) a robot, and drags starting in other areas of the screen, which are referred to as "paths". The roughly equal proportion of paths and drag robot gestures are likely caused by stuttering breaking a drag robot gesture into a drag robot gesture at its

	Detected Gesture	Count	Coded Gesture	Count
	Drag Robot	635	Drag	877
	Path	685		
	Tap Waypoint	799	Tap	304
	Tap Select	594		
	Lasso Select	9	Lasso Select	115
	Box Select	67	Box Select	55
			UI	83
			Other	55
			Voice command	16
			Pinch	36
Total		2786		1541

Table 8.1: Overall comparison of detected gestures versus those from the coding of the participant data. The gaps on each side are for distinctions that are present in the coding and absent in the detection, or vice versa, such as the distinction between tapping waypoints or tapping robots, versus simply making a tapping gesture.

beginning, and a path near its end, with a gap in the middle caused by stuttering.

As with drags, the recognizers distinguish between taps on or near robots, and taps elsewhere, while the coded data does not. Because of the stuttering, many of the detected taps are fragments of paths, which became tap selects if they were on or near a robot, or waypoints if they were away from a robot.

Of the nine detected lassos, three were correct. The usual cause of an incorrectly detected lasso was a path around the screen for a patrol, or a path for a box formation, being detected as a lasso because it contained robots. Missed lasso detections were the result of three problems. The first is the previously mentioned tendency of the lasso selection recognizer to regard a lasso which began too close to a robot to be drag robot gesture, thus reducing the count of lassos while increasing the count of drag robot gestures. The second problem is, again, stutter, breaking up lasso selections into drag robots, paths, waypoints, and tap selections. The final problem is that the lasso gesture was detected based on the angle between the

beginning and end of the gesture, with the centroid of the gesture as the vertex. If a user gesture ended too far from the start, or overlapped the start by too much, then the angle fell outside of the threshold used for lasso detection, and so was treated as some other gesture.

Interestingly, box selection showed an increase over the coded data, rather than a decrease. Of the 67 detected, 17 were detected correctly, and the remaining 50 were spurious, and were not present in the coding. These were usually caused by stutter creating an incorrect path that included robots in its bounding box, and so was recognized as a box selection of those robots. In cases where users drew a cut line to separate the robots for the divide group task, the line was frequently regarded as a box selection, because the users were careful to begin the cut line well outside the robot group.

Of the box selections in the coding, 38 were not detected. The main causes of box selections failing to be detected were stutter causing the gesture to be recognized as other gestures, or the gesture beginning too close to a robot, and so being recognized as a drag-robot gesture.

Lest it appear that all of the gesture recognitions were fraught with error, in some cases the gesture detection worked perfectly and resulted in reasonable outputs, despite the fact that the participant input was collected well before the recognizers were written, and much of it was never selected for inclusion in the eventual gesture pipeline. For example, see Figure 8.2, where eight robot drag gestures were correctly detected.

In light of the gesture recognition's sensitivity to context, as well as the problems with stutter and light touches, this component of the system could be improved in a number of ways. The problem of combining intermittent contacts might be approached by searching across a gap in the direction of motion implied

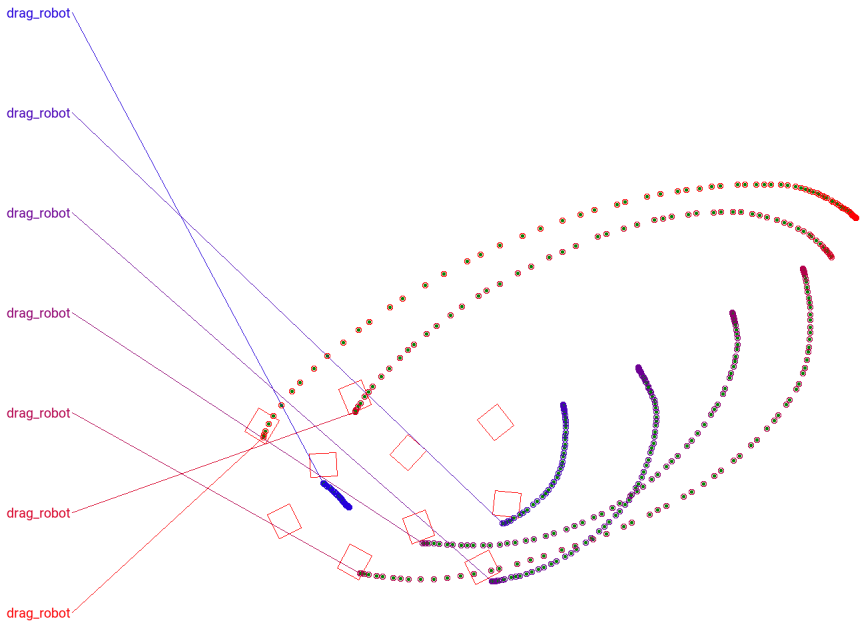


Figure 8.2: Successful recognition of all gestures, moving eight of the robots to form a line with the other two.

by the last few contacts, and allowing an approximate match for points on the opposite side of the gap, assuming the implied direction of motion was similar, and the gap crossing time was in approximate agreement with the calculated velocities between the last few points leading to the gap. Approaches such as this would require significant tuning of the parameters for the approximate matching, and so might work better for some users than others, as each user may have a different style of movement. Also, such approaches require a buffering of user input to allow the matching, and if the buffering becomes too long, will feel unresponsive or "laggy" to the user.

Such a buffer might also be useful for deciding between two gestures that are ambiguous. For example, if a gesture might be a box select or might be a command to move a single robot from the edge of the swarm across it to the other edge, and the next command is a gesture from the swarm center to a

location away from the swarm, then it is more likely that the intended command was box selection of the entire swarm and movement to the selected point, than movement of a single robot and then movement of another single robot to another point. However, even this is not certain, and so validation testing with users would be required to determine if making such a decision is always correct.

Incorporating visual feedback to the user, particularly of selection, but also of what portion of a path was detected and would be used in the resulting command, may also mitigate these problems. If the user made a gesture intended as selection, but saw that no robots were selected, they could make an additional gesture to select before issuing their next command. Indeed, the entire list of detected gestures could be made visible to the user, perhaps as a stack along one edge of the screen, and edited by rearranging or deleting gestures before sending to the swarm. The current system dispatches gestures as they are recognized, and counts on the interpreter to convert them appropriately, which has some advantages in terms of responsiveness and simplicity, but disadvantages in terms of flexibility to the user and ability to undo gestures.

Translation Testing

Translation testing is relatively straightforward. Either a sequence of gestures is in the order and of the types needed to be a valid "sentence" in the gesture language, or it is not. The recorded gesture sequences generated by the recognizers from the recorded participant input were sent to a testing version of the gesture interpreter to see if they were detected as a valid program, or rejected.

Four hundred forty-nine of the gestures resulting from the participant input were not accepted by the gesture translator. The primary causes of failure

of a gesture set to be regarded as a valid input program for the gesture translator are the gesture set starting with a path (115 instances) or starting with a waypoint (81 instances). These problems arise any time that either the user did not begin a command with a selection, or the selection was misdetected as a path. As discussed in the previous section, selections could be mis-detected as paths for a number of reasons, including stutter. Waypoints with no prior selection caused 15 failed translations, while paths with no prior selection caused 21 failures. In these cases, rather than the the gesture set starting with the waypoint or path, the waypoint or path occurs at some point within the gesture set, but has no preceding selection. The ultimate cause is similar to the cases where the gesture set starts with a waypoint or path, in that there is a waypoint or path that does not have a "subject" in the sentence for which the waypoint or path is the "verb". In total, this class of errors accounts for 232 of the rejected gesture sets, or approximately 52% of the rejections. This is not entirely unexpected, as the participants in the study were not required to perform selection gestures before waypoint or path gestures, and so many did not. Because the translator expects selection gestures before motion commands, these were not regarded as valid sequences of gestures. Defaulting to automatically considering the command to be for all robots, if no robots were selected, would avoid this problem, while creating the potential for a command to be issued to all robots accidentally. Using a default select-all in this case was considered in the design phase, but rejected due to the potential to issue an undesired command to all robots.

Tap selection was designed to not start new gestures so that tap selections could be chained to select multiple robots, as discussed in Section 5.4. However, as a result, alternating tap selections and commands would result in an error, as the tap selections that did not start the command would result in a translation error.

One approach to deal with this issue would be to allow a tap selection that occurs after an action other than a tap selection to cause the previous gestures to be passed off to the parser as a command, while leaving the existing tap selection in the queue. This behavior would be effective, but would require correction of the misrecognition of waypoints as tap selects noted in the analysis of the gesture recognizers. Taps near robots are classified as tap selections, so that users would not have to precisely hit the robots in order to select them. As previously noted, this can result in erroneous detections of taps intended as waypoints, but located near a robot, as tap selections. Additionally, stutter could cause a path dragged over robots to break up into what would be recognized as tap selections. If these tap selections resulted in the beginning of a new command and the execution of the previous gestures as a command, it could result in undesired movement of the robots.

Drag robot operations do not chain, which is to say that while issuing a single drag robot gesture is accepted, multiple drag robot gestures in a row are not. It is likely a design oversight that drag robot commands are not treated as complete commands. After all, in the design language, they constitute both a selection and a motion command in a single gesture, with an implied subject, as in the English sentence "Go!". Were this oversight corrected, it might result in more of the participant gestures being accepted by the gesture translator.

However, this problem interacts with the misrecognition of lasso and box selection gestures as drag robot operations. Due to the common misrecognition of box and lasso selections as drag robot gestures, many of these accepted gestures would be incorrect, and would result in undesired robot motion. Finally, drag robot gestures were frequently used with the intent that the entire swarm would be commanded to move, but the gesture recognizers treated them as intended to

send a single robot along the dragged path. This caused some rejections of gesture sets because what the user intended as a selection and path was interpreted as a selection and a drag robot gesture, which is not regarded as a valid program because the selection has no meaning for the subsequent drag gesture.

Thirty-five of the gesture sets had no gestures in them. The lack of gestures is not an error, and was caused by the participants not touching the screen during their interaction with the system.

Thirty-seven of the gesture sets produced by running the participant gestures through the gesture detectors were acceptable to the gesture parser. Seventeen of them consisted of a single robot drag. Eleven of the accepted gesture sets were a set of tap selections followed by a sequence of waypoints, one more had a path instead of waypoints. Finally, six gesture sets were treated as disperse gestures, of which only one was actually intended by the user as a disperse and occurred in the disperse task. Recognition of the disperse gesture would probably best be solved by not treating the gestures as a strictly linear sequence, but only detecting a dispersal if all of the four or five strokes required to cause the detection overlap significantly. However, this would require implementation of a parser capable of dealing with simultaneity. The current parser is the Lark library, which is intended for dealing with typical programming languages, which can be represented as a linear sequence of symbols [Lark parser developers, 2018]. An alternate approach would be to include as part of gesture recognition a layer which detects sets of 4-5 overlapping strokes diverging from a central point, and replaces all of the strokes with a single gesture element, which takes the place of the simultaneous gestures in a linear representation of the gesture sequence.

Ultimately, dealing with the issues in the gesture recognizers and gesture translation is an exercise in satisficing. The causes of many of the errors were

primarily effects of design decisions made earlier in the development process in order to support participant behaviors from the user study. For example, the confusion between box selection and drag robot gestures was caused by the ability to select a robot by touching near it rather than on it, which was intended to make user interactions easier. Making the threshold for selection of a robot smaller would reduce these errors, but at the cost of making robot selection more difficult.

Chapter 9

Contributions

Swarm Hardware and Software Platform

A hardware platform for control of small, inexpensive swarm robots was developed. The fact that swarm hardware across the literature has roughly the same general layout for the hardware can be viewed as an example of similar needs leading to similar solutions. Gianpiero *et al.* describe a reference model that is inspired by the E-puck [Francesca, Brambilla, Brutschy, Trianni, and Birattari, 2014b]. The reference model could probably generalize well to other hardware, as the E-puck it is based on also has a ring of IR sensors and differential drive. The GRITSbots, Amir, and Colias all have a ring of IR sensors, as do many other designs. Most swarm robots use differential drive for steering, through either a sealed gearbox or direct drive. A generalized reference model would also be useful for developing SCT generators for robot control, as the set of free behavior models is specific to the robot, and so could be reused for new tasks using the reference system.

TinyRobo aimed to drive the cost of this style of robot down further, by

virtualizing the sensors. The TinyRobo ROS modules include the ability to have configurable virtual laser scanners, range and bearing sensors for inter-robot sensing, and virtual networks, all of which can be easily customized to allow experimentation with sensor noise or failure.

The use of children's toys as a mobility platform does not result in substantial savings, or even in easier assembly of the platform. Toys also have reliability problems that offset their possible utility as mobility platforms. However, if some other drivetrain is used, the power and control module developed in this work is still cheap, small, and easy to use. During the course of this work, 3D printers have dropped substantially in price, so the difficulty of producing custom small mechanical assemblies has been significantly reduced since the time of the development of e.g. the Jasmine micro-robots. As a consequence, the use of a 3D printed robot chassis and the TinyRobo control modules can produce a very inexpensive swarm platform.

Resolving the problems that this work ran into with AprilTags would result in an approach closer to mROBerTO. The motor drive electronics would remain the same, but either fixed color tags or LEDs as identity indicators would be used to track the robots. It has been suggested that a ring of addressable LEDs could be used to convey information from the swarm to the control system, by changing the color of LEDs on the ring or animating them in patterns. Since such a display could also be meaningful to the user, this may be an interesting direction for future research in HSI for co-located swarms.

Because of the modular nature of the system, the ROS stack can be interfaced with simulations as well as with real robots. The main feature making this possible is the fact that the communication with the swarm hardware operates using standard ROS modules as much as possible, and only performs hardware-specific

operations where they cannot be avoided. This flexibility enabled an easy transition to testing in simulation when it became apparent that the toy bases were not going to become a useful platform.

Multitouch Gesture set for Swarm Control

Gestures were collected from 50 users to define a gesture set for multitouch swarm control. Analysis of the data showed that there were variations in the uses of certain gestures as the size of the swarm increased. In particular, the use of selection gestures increased in the 10 and 100 robot cases, but dropped off for the 1000 robot swarm. Tap gestures were more likely to be used for selection in the unknown, 1, and 10 robot cases than the other cases, while group selections were used mainly in the 10 and 100 robot cases, and less in the 1000 robot case. The 10 robot case seems to be the transition point where use of group or single tap selections are equally used.

Showing the area occupied by the swarm as a cloud had very similar use of selections to the one robot case. Box selection was never used, lasso was used rarely, and tap selections were the most common selection gesture by far. These changes to the user's choices of gesture support the hypothesis that the gesture selection does change with the user's perception of the swarm, both the visible number of robots and whether the swarm is rendered as individual robots or a coverage area.

There is evidence from the user survey and gesture selection that video games and other prior experiences with multi-touch interaction devices have an influence on the gestures used. The interface design used in the experiment was initially somewhat like the interface of a realtime strategy game, and as a result,

seems to have cued users who had played realtime strategy games (RTSs) to use styles of interaction common to RTSs. Designers of future interfaces can interpret this as both a strategy and a warning. As a strategy, interfaces can be designed specifically to include design features from games, and so cue the users that the interface supports the interactions that are common in the genre of game that the interface emulates. However, it can also be a warning that if an interface resembles a game, the users will expect those interaction styles, and may entirely avoid other interaction styles that do not fit with their expectations. The total absence of pinch gestures on the part of RTS gaming users is such a missing interaction. At a more abstract level, designers would be wise to avoid leaning too heavily on game-based cues in user interface design, unless they are certain that their users are gamers, or the cue will be missed.

Despite being told that the device is multitouch, most users made very few two-handed gestures, although half of the users made at least one two-handed gesture. Voice commands were more common in the data set collected for this experiment than in previous experiments that allowed users free reign in choice of their command set. It is speculated that this is due to the rise in functionality and prevalence of "voice assistant" technologies in smartphones and home appliances such as Google Home. This transition would be similar to the transition observed in previous work, where people who had smartphones used pinch gestures far more than people who did not have experience with smartphones or similar multitouch devices.

It had been surmised that one possible sign that the user was treating the robots as a group would be that some parts of the group would be neglected. This generally did not occur. Instead, the users used selection gestures that included all robots, and when asked about inclusion in gestures, erred on the side of including

more robots.

There were also relatively few gestures treating the robots as a deformable mass that could be pushed around, like a pile of sand or other small objects. Physical affordances like this were speculated to be more likely, given the direct interaction style of the multitouch screen. Their absence could be viewed as highlighting the users' understanding of the screen as an image, and so not something that supports physically-afforded interactions.

In future, it would be interesting to repeat this work with a condition that does not display the robots in the user interface at all. It is expected that for conditions such as the "move the crate" tasks, the user would simply indicate the crate should move to area A, without concern for which robots perform the moving. However, such an interface would not afford indicating particular robots or groups, so tasks such as dividing the robots around an obstacle may become impossible to perform.

Compilation of User Gestures into Robot Programs

This project shows a basic conversion from user gestures into a set of command programs to be distributed to the swarm robots. These command programs are intended to balance desirable formal properties of the programs, such as convergence within bounded time, use of local-only sensing, and completeness, with reflecting the user's intentions in the observable behavior of the swarm.

In attempting to derive complete program frameworks for the tasks specified in the user studies, it was determined that complete controllers for some simple tasks may not exist. For example, while it is possible to have a complete

controller for motion to a point, it is impossible to have a complete dispersion controller that relies on only local sensing. In cases where completeness could not be determined, or was determined to be impossible, the controllers used are developed to use local-only sensing and if they fail, to do so in a manner that is intelligible to the user.

The translation between user commands and robot programs in this work was developed using a method similar to compiler development, where an input language was defined from the user gesture commands, and an interpreter was developed to take strings in the language of gestures, and output control programs in as statements in an implementation of guarded control programming with rates (GCPR) [Napp and Klavins, 2011]. Previous work in gesture control for small groups of robots was also able to recognize a grammar of user inputs using a finite state machine [Micire, 2010]. It had been initially hoped that there would be a universal abstraction or transformation that could represent the conversion of all gestures into all robot commands representing those gestures, rather than having different sets of behaviors, with different properties, which could be constructed from the user gestures after recognizing those gestures. However, user gestures do not provide a sufficiently explicit means of designing a program in realtime to allow for the full automation of the generation of control programs without assuming an amount of *a priori* knowledge that is unrealistic in practice. For example, while there is a gesture for disperse, the gesture itself does not contain information about how the dispersion should be performed. Similarly, while the gestures used to indicate that an object should be moved to a location contain information about what the movement should be, they do not contain information about how the movement should be accomplished. Because these elements are not present in the user input, any universal transformation that has them in its output would have to

essentially make them up. As a consequence, the translation was implemented in the way that programming languages are usually implemented, with a developer supplying the method that the robots would use to fulfill the user's commands.

There are a number of threads in current swarm robotic research that may eventually permit the engineering of swarm behavior from high level descriptions, many of which are discussed in Section 2.3. One that is not discussed in great depth there is the field of attempts to find mappings between relatively low-dimensioned swarm macrostates and potentially extremely high-dimensional state spaces representing each robot individually. While some work has been done in controlling swarms with controllers operating on macro-level behavior, such as attractors, there is not yet an engineering discipline that would guide the creation of arbitrary swarm behaviors under real-world complexity [Brown et al., 2014]. As this area of swarm robotics develops, user interfaces will have to develop with it, in order to allow humans to operate these systems in a fluent manner.

Chapter 10

Directions for Future Work

The style of experiment described by Wobbrock *et al.* was not used in this work because of the possibility that showing how the behavior is performed by the robots would influence the user's choice of gesture. For example, if each robot moves in turn, it would suggest single-robot interactions, rather than group-oriented gestures. However, participants did seem to expect some level of response or interactivity from the user interface. Without a reaction from the interface, the participants appeared uncertain about their first actions, and so made incomplete sequences of actions. However, since these were actions the user made, they did contribute, though possibly not significantly, to the total set of user actions. Showing the behavior of the system might allow the participants to be confident of making a gesture without the system reacting, since they had, essentially, already seen the reaction.

Another possible way to prevent these hesitant initial interactions would be to simply run the experiment with actual paper prototypes, rather than the prototype interface that participants saw in this experiment. Since the participants

would be able to see that the interface was paper, they would not expect it to react in the way that a computer-based system would. Using paper prototypes would lose the ability to record participant input in the way that the screen did. However, as discussed earlier, a technical flaw prevented recording of 20 participant inputs, and it was not a serious problem for the experiment, as the video recordings were sufficient to recover the participant interactions. If the precise user contact points on the interface were required, a top-projected multitouch interface, such as the Mitsubishi Diamondtouch could be used without projection to record contact points on the paper.

Another possible confounding element of this study was the size the robots were depicted on the screen. As can be seen in Appendix B, the size of the robots was diminished to keep the area of the swarm the same size and fit them all on the screen. This may have made it easier for users to perform single-robot interactions with the sizes of robots used in the 1 and 10 robot conditions. If the robots were all depicted in the same size as the 1000 robot case, this confound would be removed. The small size of the robots in the 1000 robot case, however, caused some users to question their ability to move the crate in the tasks requiring it, as the crate remained the same size. The crate could also be depicted at a size that seemed more manageable for the robots.

Rather than being concerned with the size of the depicted robots, it might be interesting to perform an experiment with a UI that does not depict the swarm location at all. The utility of such a UI is debatable. While it would be impossible to have user feedback in such a UI, or to easily select some subset of the swarm, but it might be useful for commanding operation in an area where the robots are denied localization, or cannot communicate their location to the command computer due to the presence of hostile actors in the area. It may be

that the user gestures would cease to have subjects, and would consist only of the actions they wanted performed, or changes they wanted to see in the displayed environment.

In the proposed UI, the communication channel back to the user from the swarm is implicit. The swarm's behavior, as displayed to the user, allows the user to determine whether the team appears to be doing what the user commanded. Having the swarm follow the path from the user's input was chosen because it was expected that it would provide clearer feedback than biased random walks or other strategies which converge to the desired result, but may not be obvious that they are doing so. However, this decision was not tested. It may be that users would find other robot behaviors just as satisfactory, as long as the desired goal state was reached. Such a test would likely be better done in simulation, as simulation would allow finer control over the robots' behavior and reliability than a hardware swarm.

Some of the design decisions made in the process of this work were reasonable choices when the work began, but more recent work shows alternative approaches that are also promising for the development of software architectures for development of swarm robots and programs to control them. One possible approach is the use of Supervisory Control Theory (SCT), rather than GCPR for the generation of robot control program. SCT allows the generation of supervisors, which map uncontrollable events, such as sensor precepts, to controllable events, such as motor actions of the robot. The "uncontrollable events" are only uncontrollable in the sense that they are outside of the realm of events that the robot can directly cause, not necessarily stochastic or otherwise not amenable to any form of control. SCT has been applied to robotic systems, but without connection to user interfaces that would permit its use by non-programmers [Lopes, Leal, Dodd, and Groß, 2014; Lopes et al., 2016].

Another possible approach is the use of modular swarm robot behaviors, possibly combined with SCT by using the modular behaviors as the controllable events of the supervisor. Using modular behaviors would allow easier debugging than GCPR, as conventional software development techniques could be used to produce the modules, which could then be reused. This is in line with the design philosophy of ROS. Recently, CMUSWARM and ROSBuzz were released, which provide swarm-oriented programming frameworks underpinned by ROS [Arpino, Morris, Nagavalli, and Sycara, 2018; St-Onge, Varadharajan, Li, Svogor, and Beltrame, 2017]. These tools were not available at the inception of this work, and would likely have greatly accelerated the development of the translation layer.

The development of standard software tools for swarm robots would neatly dovetail with the possible development of a reference implementation for the hardware of swarm robots, as described in [Francesca et al., 2014a]. The robots developed as part of this work converged towards the design common to mROBerTO, Colias, Alice, GRITSBots, Jasmine, Amir, and the E-puck robots. This common design is a platform with differential drive steering using a sealed drivetrain or direct-drive. Even the Kilobots can be regarded as differential drive, although using vibration motors rather than wheels. The platforms are all capable of rotating in their own footprint, and equipped with omnidirectional sensing and communication, typically provided by IR sensors. They also all use the PCBs of the robot as the chassis, or a custom chassis which could be produced by 3D printing. From this convergent evolution, it could be argued that a reference swarm robot hardware implementation would have differential drive steering, a sealed drivetrain, and omnidirectional sensing and communication. These reference implementations in both hardware and software would allow researchers to focus on the development of controllers and algorithms for swarms, rather than developing new swarm robots.

It is hoped that this work will also allow future researchers to focus their efforts on fruitful avenues for the development of useful robot swarms.

Literature Cited

H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight Jr, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 43(5):74–82, 2000. 2.3.1

Age of Empires Wiki. Age of empires list of hotkeys, 2018. URL `http://ageofempiresonline.wikia.com/wiki/List_of_Hotkeys`. 2.2.6

M. H. M. Alkilabi, A. Narayan, and E. Tuci. Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies. *Swarm Intelligence*, 11(3-4):185–209, 2017. 7.5.5

J. Alonso-Mora, S. H. Lohaus, P. Leemann, R. Siegwart, and P. Beardsley. Gesture based human-multi-robot swarm interaction and its application to an interactive display. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5948–5953. IEEE, 2015. 2.2.4

J. Alonso-Mora, J. A. DeCastro, V. Raman, D. Rus, and H. Kress-Gazit. Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles. *Autonomous Robots*, 42(4):801–824, 2018. 2.3.7

D. Andreen, P. Jenning, N. Napp, and K. Petersen. Emergent structures assembled by large swarms of simple robots. 2016. 2.2.5

Apple Corp. Use multi-touch gestures on your mac, 2017. URL `https://support.apple.com/en-us/ht204895`. 4.4.2

J. J. Arnett. The neglected 95%: why american psychology needs to become less american. *American Psychologist*, 63(7):602, 2008. 4.1.2

G. Arpino, K. Morris, S. Nagavalli, and K. Sycara. Using information invariants to compare swarm algorithms and general multi-robot algorithms: A technical report. *arXiv preprint arXiv:1802.08995*, 2018. 10

F. Arvin, K. Samsudin, and A. R. Ramli. Development of a miniature robot for swarm robotic application. *sensors*, 1793:8163, 2009. 2.1.1, 3.1.2, 7.1

F. Arvin, J. Murray, C. Zhang, and S. Yue. Colias: An autonomous micro robot for swarm robotic applications. *International Journal of Advanced Robotic Systems*, 11(7):113, 2014. doi: 10.5772/58730. URL `https://doi.org/10.5772/58730`. 2.1.1

F. Arvin, T. Krajník, A. E. Turgut, and S. Yue. Cosϕ: artificial pheromone system for robotic swarms research. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 407–412. IEEE, 2015a. 2.3.2

F. Arvin, S. Yue, and C. Xiong. Colias-ϕ: An autonomous micro robot for artificial pheromone communication. 2015b. 2.1.1

M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, and P. Pillai. Meld: A declarative approach to programming ensembles. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2794–2800. IEEE, 2007. 2.3.6

N. Ayanian, A. Spielberg, M. Arbesfeld, J. Strauss, and D. Rus. Controlling a team of robots with a single input. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1755–1762. IEEE, 2014. 1.2.3

J. Bachrach, R. Nagpal, M. Salib, and H. Shrobe. Experimental results for and theoretical analysis of a self-organizing global coordinate system for ad hoc sensor networks. *Telecommunication Systems*, 26(2-4):213–233, 2004. 7.1

E. Bahgeçi and E. Sahin. Evolving aggregation behaviors for swarm robotic systems: A systematic case study. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 333–340. IEEE, 2005. 2.3.5

J. Beal and J. Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *Intelligent Systems, IEEE*, 21(2):10–19, 2006. 2.3.1

M. Beetz, F. Stulp, P. Esden-Tempski, A. Fedrizzi, U. Klank, I. Kresse, A. Maldonado, and F. Ruiz. Generality and legibility in mobile manipulation. *Autonomous Robots*, 28(1):21, 2010. 7

C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):61–70, 2007. 2.3.4, 2.3.7

S. Bergbreiter and K. S. Pister. Cotsbots: An off-the-shelf platform for distributed robotics. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1632–1637. IEEE, 2003. 2.1.1, 3.1

M. Birattari, B. Delhaisse, G. Francesca, and Y. Kerdoncuff. Observing the effects of overdesign in the automatic design of control software for robot swarms. In

International Conference on Swarm Intelligence, pages 149–160. Springer, 2016. 2.3.7

J. Blake. *Multitouch on Windows*. Manning Publications, 5260 Mac Drive, Grand Forks, ND 58201, 2010. Excerpted on http://nui.joshland.org/2010/03/what-is-natural-user-interface-book.html. 4

M. Bonani, V. Longchamp, S. Magnenat, P. Rétornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada. The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4187–4193. IEEE, 2010. 2.1.2

A. Bowyer. Automated construction using co-operating biomimetic robots. *University of Bath Department of Mechanical Engineering Technical Report (November 2000)*, 2000. 2.3.2

R. A. Brooks. Artificial life and real robots. In *Proceedings of the First European Conference on artificial life*, pages 3–10, 1992. 2.3.7

D. S. Brown, S. C. Kerman, and M. A. Goodrich. Human-swarm interactions based on managing attractors. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 90–97. ACM, 2014. 2.2.4, 9.3

D. S. Brown, M. A. Goodrich, S.-Y. Jung, and S. C. Kerman. Two invariants of human swarm interaction. *Journal of Human-Robot Interaction*, 5(1):1–31, 2015. 2.2.4

D. S. Brown, R. Turner, O. Hennigh, and S. Loscalzo. Discovery and exploration of novel swarm behaviors given limited robot capabilities. In *Distributed Autonomous Robotic Systems*, pages 447–460. Springer, 2018. 2.3.5

D. J. Bruemmer, D. A. Few, R. L. Boring, J. L. Marble, M. C. Walton, and C. W. Nielsen. Shared understanding for collaborative control. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 35(4):494–504, 2005. 2.5

G. Caprari, P. Balmer, R. Piguet, and R. Siegwart. The autonomous micro robot alice: a platform for scientific and commercial applications. In *Micromechatronics and Human Science, 1998. MHS'98. Proceedings of the 1998 International Symposium on*, pages 231–235. IEEE, 1998. 2.1.1, 7.1

J. Carlson, R. R. Murphy, and A. Nelson. Follow-up analysis of mobile robot failures. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4987–4994. IEEE, 2004. 3.3

J. Cham. How robotics research keeps... re-inventing the wheel. `http://www.willowgarage.com/blog/2010/04/27/reinventing-wheel`, 2010. 2.3

J. Chen, M. Gauci, W. Li, A. Kolling, and R. Gros. Occlusion-based cooperative transport with a swarm of miniature mobile robots. *Robotics, IEEE Transactions on*, 31(2):307–321, 2015. 7.5.5

J. Y. Chen, M. J. Barnes, and M. Harper-Sciarini. Supervisory control of multiple robots: Human-performance issues and user-interface design. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 41(4): 435–454, 2011. 2.2

J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960. 4.2.2

A. Colot, G. Caprari, and R. Siegwart. Insbot: Design of an autonomous mini mobile robot able to interact with cockroaches. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2418–2423. IEEE, 2004. 2.1.1

G. Coppin and F. Legras. Controlling swarms of unmanned vehicles through user-centered commands. In *2012 AAAI Fall Symposium Series*, 2012. 2.2.1, 2.3.2

A. Cornejo, A. J. Lynch, E. Fudge, S. Bilstein, M. Khabbazian, and J. McLurkin. Scale-free coordinates for multi-robot systems with bearing-only sensors. *The International Journal of Robotics Research*, 32(12):1459–1474, 2013. 7.1

N. Correll, J. Bachrach, D. Vickery, and D. Rus. Ad-hoc wireless network coverage with networked robots that cannot localize. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3878–3885. IEEE, 2009. 7.5.4

N. Correll, P. Dutta, R. Han, and K. Pister. Wireless robotic materials. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, pages 24:1–24:6, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5459-2. doi: 10.1145/3131672.3131702. URL `http://doi.acm.org/10.1145/3131672.3131702`. 2.3.1

V. Costa, M. Duarte, T. Rodrigues, S. M. Oliveira, and A. L. Christensen. Design and development of an inexpensive aquatic swarm robotics system. In *OCEANS 2016-Shanghai*, pages 1–7. IEEE, 2016. 2.1.1

M. L. Cummings and P. J. Mitche. Predicting controller capacity in supervisory control of multiple uavs. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 38(2):451–460, 2008. 2.2

M. Daily, Y. Cho, K. Martin, and D. Payton. World embedded interfaces for human-robot interaction. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 5 - Volume 5*, HICSS '03, pages 125.2–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1874-5. URL `http://dl.acm.org/citation.cfm?id=820752.821587`. 2.2

K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh. Programming micro-aerial vehicle swarms with karma. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 121–134. ACM, 2011. 2.3.6

J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell. An integrated system for perception-driven autonomy with modular robots. *arXiv preprint arXiv:1709.05435*, 2017. 2.3.7

J. Derboven, D. De Roeck, and M. Verstraete. Semiotic analysis of multi-touch interface design: The mutable case study. *International Journal of Human-Computer Studies*, 70(10):714–728, 2012. 2.2.3

Y. Diaz-Mercado, S. G. Lee, and M. Egerstedt. Human–swarm interactions via coverage of time-varying densities. In *Trends in Control and Decision-Making for Human–Robot Collaboration Systems*, pages 357–385. Springer, 2017. 2.2.4, 2.3.2

G. Dietz, J. L. E, P. Washington, L. H. Kim, and S. Follmer. Human perception of swarm robot motion. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 2520–2527, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4656-6. doi: 10.1145/3027063.3053220. URL `http://doi.acm.org/10.1145/3027063.3053220`. 7

M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, et al. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2-3):223–245, 2004. 2.3.5

M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, et al. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *Robotics & Automation Magazine, IEEE*, 20(4):60–71, 2013. 2.1.2

A. D. Dragan, S. Bauman, J. Forlizzi, and S. S. Srinivasa. Effects of robot motion on human-robot collaboration. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 51–58. ACM, 2015. 7

R. I. Dunbar. Neocortex size as a constraint on group size in primates. *Journal of human evolution*, 22(6):469–493, 1992. 2.4

P. Ehn and M. Kyng. Cardboard computers: Mocking-it-up or hands-on the future. In *Design at work*, pages 169–196. L. Erlbaum Associates Inc., 1992. 4.1

J. Epps, S. Lichman, and M. Wu. A study of hand shape use in tabletop gesture interaction. In *CHI'06 extended abstracts on human factors in computing systems*, pages 748–753. ACM, 2006. 4.4

Europa Universalis IV Wiki. Europa universalis keyboard shortcuts, 2017. URL `https://eu4.paradoxwikis.com/Controls`. 2.2.6

D. Evans. Programming the swarm. *University of Virginia*, 2000. 2.3.4

N. Farrow, J. Klingner, D. Reishus, and N. Correll. Miniature six-channel range and bearing system: algorithm, analysis and experimental validation. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6180–6185. IEEE, 2014. 7.1

P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954. 5.2, 6.3

J. L. Fleiss, B. Levin, M. C. Paik, W. A. Shewart, and S. S. Wilks. John Wiley and Sons, Inc., 2003. 4.2.2

S. Floyd, C. Pawashe, and M. Sitti. An untethered magnetically actuated micro-robot capable of motion on arbitrary surfaces. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 419–424. IEEE, 2008. 2.1.1

T. Fong, M. Bualat, L. Edwards, L. Flückiger, C. Kunz, S. Lee, E. Park, V. To, H. Utz, N. Ackner, et al. Human-robot site survey and sampling for space exploration. In *Space 2006*, page 7425. 2006. 6

G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, et al. An experiment in automatic design of robot swarms. In *International Conference on Swarm Intelligence*, pages 25–37. Springer, 2014a. 2.3.7, 10

G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. Automode: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112, 2014b. 9.1

G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, et al. Automode-chocolate: A method for the automatic design of robot swarms that outperforms humans. *Swarm Intelligence*, 9(2-3):125–152, 2015. 2.3.7

M. O. Franz, B. Schölkopf, H. A. Mallot, and H. H. Bülthoff. Learning view graphs for robot navigation. *Autonomous robots*, 5(1):111–125, 1998. 7.6

D. Freeman, H. Benko, M. R. Morris, and D. Wigdor. Shadowguides: visualizations for in-situ learning of multi-touch and whole-hand gestures. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 165–172. ACM, 2009. 4

Gamepedia. Dota controls, 2018. URL `https://dota2.gamepedia.com/Controls`. 2.2.6

J. Gancet, E. Motard, A. Naghsh, C. Roast, M. M. Arancon, and L. Marques. User interfaces for human robot interactions with a swarm of robots in support to firefighters. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2846–2851. IEEE, 2010. 2.2, 2.2.5

M. Gauci, J. Chen, W. Li, T. J. Dodd, and R. Groß. Self-organized aggregation without computation. *The International Journal of Robotics Research*, 33(8): 1145–1161, 2014. 2.3.5

A. Giusti, J. Nagi, L. M. Gambardella, S. Bonardi, and G. A. Di Caro. Human-swarm interaction through distributed cooperative gesture recognition. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 401–402. ACM, 2012. 2.2.4

B. G. Glaser and A. L. Strauss. *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017. 4.2

W. R. Glaser. The impact of user-input devices on virtual desktop trainers. Technical report, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 2010. 6.1

T. Goedemé, M. Nuttin, T. Tuytelaars, and L. Van Gool. Omnidirectional vision based topological navigation. *International Journal of Computer Vision*, 74(3): 219–236, 2007. 7.6

M. A. Goodrich, B. Pendleton, P. Sujit, and J. Pinto. Toward human interaction with bio-inspired robot teams. In *Systems, man, and cybernetics (smc), 2011 ieee international conference on*, pages 2859–2864. IEEE, 2011. 2.2.4

S. Gross, J. Bardzell, and S. Bardzell. Skeu the evolution: skeuomorphs, style, and the material of tangible interactions. In *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction*, pages 53–60. ACM, 2014. 2.2.3

Y. Guo, M. Hohil, and S. V. Desai. Bio-inspired motion planning algorithms for autonomous robots facilitating greater plasticity for security applications. In *Optics/Photonics in Security and Defence*, pages 673608–673608. International Society for Optics and Photonics, 2007. 2.1.2

C. E. Harriott, A. E. Seiffert, S. T. Hayes, and J. A. Adams. Biologically-inspired human-swarm interaction metrics. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 58, pages 1471–1475. SAGE Publications, 2014. 2.3.5

S. G. Hart. Nasa-task load index (nasa-tlx); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50, pages 904–908. Sage publications Sage CA: Los Angeles, CA, 2006. 6

A. T. Hayes, A. Martinoli, and R. M. Goodman. Swarm robotic odor localization. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 1073–1078. IEEE, 2001. 2.3.2

S. T. Hayes, E. R. Hooten, and J. A. Adams. Multi-touch interaction for tasking robots. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pages 97–98. IEEE Press, 2010. 2.2

J. Hilder, A. Horsfield, A. G. Millard, and J. Timmis. The psi swarm: A low-cost robotics platform and its use in an education setting. In *Conference Towards Autonomous Robotic Systems*, pages 158–164. Springer, 2016. 2.1.1

P. J. Hinds, T. L. Roberts, and H. Jones. Whose job is it anyway? a study of human-robot interaction in a collaborative task. *Human-Computer Interaction*, 19(1):151–181, 2004. 2.5

A. Hocraffer and C. S. Nam. A meta-analysis of human-system interfaces in unmanned aerial vehicle (uav) swarm management. *Applied ergonomics*, 58:66–80, 2017. 2.2

N. R. Hoff, A. Sagoff, R. J. Wood, and R. Nagpal. Two foraging algorithms for robot swarms using only local communication. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 123–130. IEEE, 2010. 2.3.2

E. Hornecker. Beyond affordance: tangibles' hybrid nature. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, pages 175–182. ACM, 2012. 4

C. M. Humphrey, C. Henk, G. Sewell, B. W. Williams, and J. A. Adams. Assessing the scalability of a multiple robot interface. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 239–246. ACM, 2007. 2.2.4

Intel Corp. Drone light shows powered by intel, 2018a. URL `https://www.intel.com/content/www/us/en/technology-innovation/aerial-technology-light-show.html`. 2.2.7

Intel Corp. Olympic drones tech behind the light show, 2018b. URL https://www.intel.com/content/www/us/en/technology-innovation/aerial-technology-light-show.html. 2.2.7

InterWave Studios. Nuclear dawn, 2013. URL https://commons.wikimedia.org/wiki/File:Nuclear_Dawn_-_Oasis_RTS_02.png#/media/File:Nuclear_Dawn_-_Oasis_RTS_02.png. 2.2

R. J. Jacob. What you look at is what you get: eye movement-based interaction techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 11–18. ACM, 1990. 8.5.1

S. Jantz, K. Doty, J. Bagnell, and I. Zapata. Kinetics of robotics: The development of universal metrics in robotic swarms. In *Florida Conference on Recent Advances in Robotics*. Citeseer, 1997. 2.3.5

R. L. Jeanne. Alarm recruitment, attack behavior, and the role of the alarm pheromone in polybia occidentalis (hymenoptera: Vespidae). *Behavioral Ecology and Sociobiology*, 9(2):143–148, 1981. 2.3.2

G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit. An end-to-end system for accomplishing tasks with modular robots. In *Robotics: Science and Systems*, 2016. 2.3.7

M. Johnson and D. Brown. Evolving and controlling perimeter, rendezvous, and foraging behaviors in a computation-free robot swarm. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 311–314. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016. 2.3.5

D. B. Kaber and M. R. Endsley. Out-of-the-loop performance problems and the use of intermediate levels of automation for improved control system functioning and safety. *Process Safety Progress*, 16(3):126–131, 1997. 2.2

S. T. Kalat, S. G. Faal, U. Celik, and C. D. Onal. Tribot: A minimally-actuated accessible holonomic hexapedal locomotion platform. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 6292–6297. IEEE, 2015. 2.1.1

I. Kamon, E. Rimon, and E. Rivlin. Tangentbug: A range-sensor-based navigation algorithm. *The International Journal of Robotics Research*, 17(9):934–953, 1998. 7.5

G. Kapellmann-Zafra, N. Salomons, A. Kolling, and R. Groß. Human-robot swarm interaction with limited situational awareness. In *International Conference on Swarm Intelligence*, pages 125–136. Springer, 2016. 2.2

J. Kato, D. Sakamoto, M. Inami, and T. Igarashi. Multi-touch interface for controlling multiple mobile robots. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 3443–3448, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-247-4. doi: 10.1145/1520340.1520500. URL `http://doi.acm.org/10.1145/1520340.1520500`. 2.2, 2.3.3

S. Kernbach. Swarmrobot. org-open-hardware microrobotic project for large-scale artificial swarms. *arXiv preprint arXiv:1110.5762*, 2011. 2.1.1

J. Y. Kim, T. Colaco, Z. Kashino, G. Nejat, and B. Benhabib. mroberto: A modular millirobot for swarm-behavior studies. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2109–2114, 2016. 2.1.1, 3.3.1

Z. Kira and M. A. Potter. Exerting human control over decentralized robot swarms. In *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*, pages 566–571. IEEE, 2009. 2.2.4

E. Klavins. Automatic synthesis of controllers for distributed assembly and formation forming. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 3, pages 3296–3302. IEEE, 2002. 2.3.7

A. Kolling, K. Sycara, S. Nunnally, and M. Lewis. Human swarm interaction: An experimental study of two types of interaction with foraging swarms. *Journal of Human-Robot Interaction*, 2(2), 2013. 2.2.4

T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013. 7

C. R. Kube and H. Zhang. The use of perceptual cues in multi-robot box-pushing. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2085–2090. IEEE, 1996. 7.5.5

M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011. 8.2

D. Laird, J. Price, and I. A. Raptis. Spider-bots: A low cost cooperative robotics platform. In *ASEE 2014 Zone 1 Conference*. American Society for Engineering Education, 2014. 3.1, 3.3.1

C. Landsiedel. *Semantic Mapping for Autonomous Robots in Urban Environments*. PhD thesis, Technische Universität München, 2018. 7.6

Lark parser developers. Lark parser, 2018. URL `https://github.com/lark-parser/lark`. 8.6

B. Larochelle, G.-J. M. Kruijff, N. Smets, T. Mioch, and P. Groenewegen. *Establishing human situation awareness using a multi-modal operator control unit in an urban search & rescue human-robot team.* IEEE, 2011. 2.5

M. Le Goc, L. H. Kim, A. Parsaei, J.-D. Fekete, P. Dragicevic, and S. Follmer. Zooids: Building blocks for swarm user interfaces. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 97–109. ACM, 2016. 2.1.1, 2.2.4

J. D. Lee and K. A. See. Trust in automation: Designing for appropriate reliance. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 46 (1):50–80, 2004. 2.2

P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005. 2.1.1

M. Lewis, J. Polvichai, K. Sycara, and P. Scerri. Scaling-up human control for large uav teams. *Human factors of remotely operated vehicles*, 7:237–250, 2006. 2.2

B. Lickel, D. L. Hamilton, G. Wieczorkowska, A. Lewis, S. J. Sherman, and A. N. Uhles. Varieties of groups and the perception of group entitativity. *Journal of personality and social psychology*, 78(2):223, 2000. 2.4

B. Lickel, D. L. Hamilton, and S. J. Sherman. Elements of a lay theory of groups: Types of groups, relational styles, and the perception of group entitativity. *Personality and Social Psychology Review*, 5(2):129–140, 2001. 2.4

A. M. Liu, C. M. Oman, R. Galvan, and A. Natapoff. Predicting space telerobotic operator training performance from human spatial ability assessment. *Acta Astronautica*, 92(1):38–47, 2013. 6

S. G. Loizou and V. Kumar. Biologically inspired bearing-only navigation and tracking. In *Decision and Control, 2007 46th IEEE Conference on*, pages 1386–1391. IEEE, 2007. 7.1

Y. K. Lopes, A. B. Leal, T. J. Dodd, and R. Groß. Application of supervisory control theory to swarms of e-puck and kilobot robots. In *International Conference on Swarm Intelligence*, pages 62–73. Springer, 2014. 10

Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß. Supervisory control theory applied to swarm robotics. *Swarm Intelligence*, 10(1):65–97, 2016. 2.3.7, 10

Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß. Probabilistic supervisory control theory (psct) applied to swarm robotics. In *Proceedings*

of the 16th Conference on Autonomous Agents and MultiAgent Systems, pages 1395–1403. International Foundation for Autonomous Agents and Multiagent Systems, 2017. 2.3.7

C. I. Lopez, J. Kuczynski, and H. A. Yanco. Unified human and robot command for disaster recovery situations. In *Technologies for Homeland Security (HST), 2017 IEEE International Symposium on*, pages 1–6. IEEE, 2017. 2.5

V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2 (1-4):403–430, 1987. 7.3, 7.5.2

M. Mamei, F. Zambonelli, and L. Leonardi. Co-fields: Towards a unifying approach to the engineering of swarm intelligent systems. In *Engineering Societies in the Agents World III*, pages 68–81. Springer, 2003. 2.3.3

M. D. Manning, C. E. Harriott, S. T. Hayes, J. A. Adams, and A. E. Seiffert. Heuristic evaluation of swarm metrics' effectiveness. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts*, pages 17–18. ACM, 2015. 2.2.1

Z. Mason. Programming with stigmergy: using swarms for construction. *Proceedings of Artificial Life*, 8:371–374, 2003. 2.3.2

M. J. Mataric. Navigating with a rat brain: a neurobiologically inspired model. In *From Animals to Animats; Proceedings of the First International Conference on Simulation of Adaptive Behavior. MIT Press, Cambridge, Mass*, page 81, 1991. 7.6

M. J. Matari. Interaction and intelligent behavior. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 1994. 7.5.4

S. J. McDonald, M. B. Colton, C. K. Alder, and M. A. Goodrich. Haptic shape-based management of robot teams in cordon and patrol. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 380–388. ACM, 2017. 4.6.1

J. McLurkin, J. Smith, J. Frankel, D. Sotkowitz, D. Blau, and B. Schmidt. Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots. In *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, pages 72–75, 2006. 2.2

J. McLurkin, A. J. Lynch, S. Rixner, T. W. Barr, A. Chou, K. Foster, and S. Bilstein. A low-cost multi-robot system for research, teaching, and outreach. In *Distributed Autonomous Robotic Systems*, pages 597–609. Springer, 2013. 2.1.1

J. D. McLurkin. *Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots*. PhD thesis, Massachusetts Institute of Technology, 2004. 2.3.4, 8.4

D. McNeill. So you think gestures are nonverbal? *Psychological review*, 92(3):350, 1985. 5.1

A. M. Mehta, J. DelPreto, K. W. Wong, S. Hamill, H. Kress-Gazit, and D. Rus. Robot creation from functional specifications. In *Robotics Research*, pages 631–648. Springer, 2018. 2.3.7

M. Micire, M. Desai, A. Courtemanche, K. M. Tsui, and H. A. Yanco. Analysis of natural gestures for controlling robot teams on multi-touch tabletop surfaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-733-2. doi: 10.1145/1731903.1731912. URL `http://doi.acm.org/10.1145/1731903.1731912`. 2.2, 2.2.2, 2.5, 4

M. Micire, M. Desai, J. L. Drury, E. McCann, A. Norton, K. M. Tsui, and H. A. Yanco. Design and validation of two-handed multi-touch tabletop controllers for robot teleoperation. In *Proceedings of the 16th international conference on Intelligent user interfaces*, pages 145–154. ACM, 2011. 6.1

M. J. Micire. *MULTI-TOUCH INTERACTION FOR ROBOT COMMAND AND CONTROL*. PhD thesis, Citeseer, 2010. 4.4, 4.4.1, 4.4.3, 5.3, 6, 6.7, 9.3

A. G. Millard, R. Joyce, J. A. Hilder, C. Fleşeriu, L. Newbrook, W. Li, L. J. McDaid, and D. M. Halliday. The pi-puck extension board: a raspberry pi interface for the e-puck robot platform. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 741–748. IEEE, 2017. 1

F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009. 7.1

L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi. Team-level programming of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 177–190. ACM, 2014. 2.3.6

M. A. Nacenta, Y. Kamber, Y. Qiang, and P. O. Kristensson. Memorability of pre-designed and user-defined gesture sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1099–1108. ACM, 2013. 4

S. Nagavalli. Algorithms for timing and sequencing behaviors in robotic swarms. 2018. 2.4

J. Nagi, A. Giusti, L. M. Gambardella, and G. A. Di Caro. Human-swarm interaction using spatial gestures. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3834–3841. IEEE, 2014a. 2.2.4

J. Nagi, A. Giusti, F. Nagi, L. M. Gambardella, and G. A. Di Caro. Online feature extraction for the incremental learning of gestures in human-swarm interaction. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3331–3338. IEEE, 2014b. 2.2.4

R. Nagpal. A catalog of biologically-inspired primitives for engineering self-organization. In *Engineering Self-Organising Systems*, pages 53–62. Springer, 2004. 2.3.4, 8.4

R. Nagpal and M. Mamei. Engineering amorphous computing systems. In *Methodologies and Software Engineering for Agent Systems*, pages 303–320. Springer, 2004. 2.3.1

N. Napp and E. Klavins. A compositional framework for programming stochastically interacting robots. *The International Journal of Robotics Research*, 30(6):713–729, 2011. 2.3.4, 9.3

J. Ng and T. Bräunl. Performance comparison of bug navigation algorithms. *Journal of Intelligent and Robotic Systems*, 50(1):73–84, 2007. 7.5, 7.6

S. Nunnally, P. Walker, A. Kolling, N. Chakraborty, M. Lewis, K. Sycara, and M. Goodrich. Human influence of robotic swarms with bandwidth and localization issues. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 333–338. IEEE, 2012. 2.2.4

D. R. Olsen and M. A. Goodrich. Metrics for evaluating human-robot interactions. In *Proceedings of PERMIS*, volume 2003, page 4, 2003. 2.2

D. R. Olsen, Jr. and S. B. Wood. Fan-out: Measuring human control of multiple robots. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 231–238, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. doi: 10.1145/985692.985722. URL `http://doi.acm.org/10.1145/985692.985722`. 2.2

E. Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, May 2011. 3.2

E. Olson, J. Strom, R. Goeddel, R. Morton, P. Ranganathan, and A. Richardson. Exploration and mapping with autonomous robot teams. *Communications of the ACM*, 56(3), March 2013. URL `http://doi.acm.org/10.1145/2428556.2428574`. 2.1.2

A. Özdemir, M. Gauci, and R. Gross. Shepherding with robots that do not compute. In *Artificial Life Conference Proceedings 14*, pages 332–339. MIT Press, 2017. 2.3.5

R. OGrady, M. Birattari, and M. Dorigo. Swarmanoid, the movie. *AAAI-11 Video Proceedings*, 2011. 3.1.4

D. W. Palmer, M. Kirschenbaum, and L. Seiter. Emergence-oriented programming. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 2, pages 1441–1448. IEEE, 2005a. 2.3.5

D. W. Palmer, M. Kirschenbaum, L. M. Seiter, J. Shifflet, and P. Kovacina. Behavioral feedback as a catalyst for emergence in multi-agent systems. In *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, pages 1575–1580. IEEE, 2005b. 1.1, 2.3.5, 2.3.5

R. Parasuraman, T. B. Sheridan, and C. D. Wickens. A model for types and levels of human interaction with automation. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(3):286–297, 2000. 2.2, 2.1

R. Parasuraman, S. Galster, P. Squire, H. Furukawa, and C. Miller. A flexible delegation-type interface enhances system performance in human supervision of multiple robots: Empirical studies with roboflag. *IEEE Transactions on systems, man, and cybernetics-part A: Systems and Humans*, 35(4):481–493, 2005. 2.2.4

M. Patil, T. Abukhalil, S. Patel, and T. Sobh. Ub robot swarmdesign, implementation, and power management. In *Control and Automation (ICCA), 2016 12th IEEE International Conference on*, pages 577–582. IEEE, 2016. 3.4

D. Payton, R. Estkowski, and M. Howard. Compound behaviors in pheromone robotics. *Robotics and Autonomous Systems*, 44(3):229–240, 2003. 2.3.2

D. W. Payton, M. J. Daily, B. Hoff, M. D. Howard, and C. L. Lee. Pheromone robotics. In *Intelligent Systems and Smart Manufacturing*, pages 67–75. International Society for Optics and Photonics, 2001. 2.3.2

R. Pelrine, A. Wong-Foy, B. McCoy, D. Holeman, R. Mahoney, G. Myers, J. Herson, and T. Low. Diamagnetically levitated robots: An approach to massively parallel robotic systems with unusual motion properties. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 739–744. IEEE, 2012. 1.2.3, 2.1.1

J. Penders, L. Alboul, U. Witkowski, A. Naghsh, J. Saez-Pons, S. Herbrechtsmeier, and M. El-Habbal. A robot swarm assisting a human fire-fighter. *Advanced Robotics*, 25(1-2):93–117, 2011. 2.2.5

A. Perlis. Epigrams in programming, 1982. URL `http://www.cs.yale.edu/homes/perlis-alan/quotes.html`. 8.2

D. Pickem, M. Lee, and M. Egerstedt. The gritsbot in its natural habitat-a multi-robot testbed. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4062–4067. IEEE, 2015. 2.1.1, 3.3.1

C. Pinciroli, V. Trianni, R. OGrady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, et al. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6(4):271–295, 2012. 7.2

C. Pinciroli, A. Lee-Brown, and G. Beltrame. Buzz: An extensible programming language for self-organizing heterogeneous robot swarms. *CoRR*, abs/1507.05946, 2015. URL `http://arxiv.org/abs/1507.05946`. 2.3.6

C. Pinciroli, A. Lee-Brown, and G. Beltrame. A tuple space for data sharing in robot swarms. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 287–294. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016. 2.3.2

D. Pinelle, N. Wong, and T. Stach. Heuristic evaluation for games: usability principles for video game design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1453–1462. ACM, 2008. 6.1

J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3299–3304. IEEE, 2017. 2.1.1

J. Price, D. Laird, and I. Raptis. Spider bots: A low cost platform for testing and validating cooperative control algorithms. In *ASME 2014 Dynamic Systems and Control Conference*, pages V001T14A005–V001T14A005. American Society of Mechanical Engineers, 2014. 3.4

M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009. 4.1

M. Quinn. A comparison of approaches to the evolution of homogeneous multi-robot teams. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 128–135. IEEE, 2001a. 2.3.5

M. Quinn. Evolving communication without dedicated communication channels. In *Advances in Artificial Life*, pages 357–366. Springer, 2001b. 2.3.5

M. Quinn, L. Smith, G. Mayley, and P. Husbands. Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 361(1811):2321–2343, 2003. 2.3.5

M. Rahimi, S. Gibb, Y. Shen, and H. M. La. A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing. *arXiv preprint arXiv:1809.08337*, 2018. 7.5.5

T. S. Ray. An approach to the synthesis of life. 1991. 2.3.5

C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM Siggraph Computer Graphics*, volume 21, pages 25–34. ACM, 1987. 2.2.4

B. Ricks, C. W. Nielsen, M. Goodrich, et al. Ecological displays for robot interaction: A new perspective. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2855–2860. IEEE, 2004. 2.2

M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975, 2014a. 1.2.1, 2.1.1, 3.1, 3.3.1

M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014b. ISSN 0036-8075. doi: 10.1126/science.1254295. URL `http://science.sciencemag.org/content/345/6198/795`. 7.1

J. Seyfried, M. Szymanski, N. Bender, R. Estana, M. Thiel, and H. Wörn. The i-swarm project: Intelligent small world autonomous robots for micro-manipulation. In *Swarm Robotics*, pages 70–83. Springer, 2005. 2.1.1

J. Shah, J. Wiken, B. Williams, and C. Breazeal. Improved human-robot team performance using chaski, a human-inspired plan execution system. In *Proceedings of the 6th international conference on Human-robot interaction*, pages 29–36. ACM, 2011. 2.5

T. Soule and R. B. Heckendorn. Cotsbots: computationally powerful, low-cost robots for computer science curriculums. *Journal of Computing Sciences in Colleges*, 27(1):180–187, 2011. 2.1.1, 2.1.2

R. Spica and P. R. Giordano. Active decentralized scale estimation for bearing-based localization. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5084–5091. IEEE, 2016. 7.1

D. St-Onge, V. S. Varadharajan, G. Li, I. Svogor, and G. Beltrame. ROS and buzz: consensus-based behaviors for heterogeneous teams. *CoRR*, abs/1710.08843, 2017. URL `http://arxiv.org/abs/1710.08843`. 10

D. St-Onge, C. Pinciroli, and G. Beltrame. Circle formation with computation-free robots shows emergent behavioural structure. 2018. 2.3.5

K. J. Stewart. Trust transfer on the world wide web. *Organization Science*, 14(1): 5–17, 2003. 2.4

Strategic Capabilities Office. Perdix fact sheet, 2015. URL `https://www.defense.gov/Portals/1/Documents/pubs/Perdix%20Fact%20Sheet.pdf`. 2.2.4

K. Sugawara, N. Correll, and D. Reishus. Object transportation by granular convection using swarm robots. In *Distributed autonomous robotic systems*, pages 135–147. Springer, 2014. 7.5.5

D. J. Sumpter and M. Beekman. From nonlinearity to optimality: pheromone trail foraging by ants. *Animal behaviour*, 66(2):273–280, 2003. 2.3.2

R. Suzuki, J. Kato, M. D. Gross, and T. Yeh. Reactile: Programming swarm user interfaces through direct physical manipulation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 199. ACM, 2018. 2.2.4

S. P. Swinnen and N. Wenderoth. Two hands, one brain: cognitive neuroscience of bimanual skill. *Trends in cognitive sciences*, 8(1):18–25, 2004. 6.3

T. Tammet, J. Vain, A. Puusepp, E. Reilent, and A. Kuusik. Rfid-based communications for a self-organising robot swarm. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO'08. Second IEEE International Conference on*, pages 45–54. IEEE, 2008. 2.1.2

A. Tapus and R. Siegwart. Incremental robot mapping with fingerprints of places. In *IROS*, volume 1, pages 2429–2434, 2005. 7.6

K. Taylor and S. M. LaValle. I-bug: An intensity-based bug algorithm. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3981–3986. IEEE, 2009. 7.3, 7.6

P. Thomas and R. Macredie. Games and the design of humancomputer interfaces. *Educational and Training Technology International*, 31(2):134–142, 1994. doi: 10.1080/0954730940310208. URL `https://doi.org/10.1080/0954730940310208`. 6.1

M. Tomita and M. Yamamoto. A sensor based navigation algorithm of a mobile robot with moving obstacles in its workspace assuring convergence property. *Memoirs of the Faculty of Engineering, Kyushu University*, 69(2), 2009. 7.5

E. Tuci, M. H. M. Alkilabi, and O. Akanyeti. Cooperative object transport in multi-robot systems: A review of the state-of-the-art. *Frontiers in Robotics and AI*, 5:59, 2018. 7.5.5

A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 237(641):37–72, 1952. 2.3.1, 2.3.2

A. Van Dam. Post-wimp user interfaces. *Communications of the ACM*, 40(2):63–67, 1997. 2.2.3

D. Vanacken, A. Demeure, K. Luyten, and K. Coninx. Ghosts in the interface: Meta-user interface visualizations as guides for multi-touch interaction. In *Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on*, pages 81–84. IEEE, 2008. 4

C. Vasile, A. Pavel, and C. Buiu. Integrating human swarm interaction in a distributed robotic control system. In *Automation science and engineering (CASE), 2011 IEEE conference on*, pages 743–748. IEEE, 2011. 2.2.4

K. J. Vicente. Ecological interface design: Progress and challenges. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 44(1):62–78, 2002. 2.2

K. J. Vicente and J. Rasmussen. Ecological interface design: Theoretical foundations. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(4):589–606, 1992. 2.2

M. Viroli, D. Pianini, and J. Beal. Linda in space-time: an adaptive coordination model for mobile ad-hoc environments. In *International Conference on Coordination Languages and Models*, pages 212–229. Springer, 2012. 2.3.6

P. Walker, S. Nunnally, M. Lewis, A. Kolling, N. Chakraborty, and K. Sycara. Neglect benevolence in human control of swarms in the presence of latency. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 3009–3014. IEEE, 2012. 2.2.4

H. Wang, M. Lewis, P. Velagapudi, P. Scerri, and K. Sycara. How search and its subtasks scale in n robots. In *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*, HRI '09, pages 141–148, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-404-1. doi: 10.1145/1514095.1514122. URL `http://doi.acm.org/10.1145/1514095.1514122`. 1, 2.2

T. Wareham and A. Vardy. Viable algorithmic options for designing reactive robot swarms. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 13 (1):5, 2018. 2.3.5

J. Wawerla, G. S. Sukhatme, and M. J. Mataric. Collective construction with multiple robots. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2696–2701. IEEE, 2002. 2.3.2

J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758, 2014. 2.3.2

E. L. Wiener and R. E. Curry. Flight-deck automation: Promises and problems. *Ergonomics*, 23(10):995–1011, 1980. 2.2

D. A. Wilder. Perception of groups, size of opposition, and social influence. *Journal of Experimental Social Psychology*, 13(3):253–268, 1977. 2.4

D. A. Wilder. Perceiving persons as a group: Effects on attributions of causality and beliefs. *Social Psychology*, pages 13–23, 1978. 2.4

S. Wilson, R. Gameros, M. Sheely, M. Lin, K. Dover, R. Gevorkyan, M. Haberland, A. Bertozzi, and S. Berman. Pheeno, a versatile swarm robotic research and education platform. *IEEE Robotics and Automation Letters*, 1(2):884–891, 2016. 2.1.2

A. F. Winfield, W. Liu, J. Nembrini, and A. Martinoli. Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence*, 2(2):241–266, 2008. 8.3

J. O. Wobbrock, M. R. Morris, and A. D. Wilson. User-defined gestures for surface computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1083–1092. ACM, 2009. 2.2.2, 4, 4.1

H. A. Yanco, J. L. Drury, and J. Scholtz. Beyond usability evaluation: Analysis of human-robot interaction at a major robotics competition. *Human–Computer Interaction*, 19(1-2):117–149, 2004. 2.2

J. Yao, T. Fernando, and H. Wang. A multi-touch natural user interface framework. In *Systems and Informatics (ICSAI), 2012 International Conference on*, pages 499–504. IEEE, 2012. 2.2.2

Appendices

Appendix A

Coding Definitions for User Gestures

General Points

Code the time that the user ends the gesture (takes their finger off the screen), to at least 0.1 second, 0.01 second is better.

Code all interactions with the screen. Some users draw on the screen with their finger while explaining their commands, or describing the reaction they would expect from the system. These should be coded because

- The system, being a dumb computer, can't determine the user's intentions
- Everyone coding the same things helps with inter-coder reliability

Use the "example" flag when coding these actions. The example flag is also used for if the user repeats their command while describing it, but not for the first time they do it.

- Do not code non-contact examples, which is to say when the user is discussing their thinking and not touching the screen, even if they are repeating an action that they did while touching the screen. Only count examples that touch the screen.

Lasso and Box Select are essentially specific forms of Drag, but the users frequently clarify these actions by saying things like "I'd take these robots" or "Select some robots...", or by mentioning their inspiration for the action, such as Real Time Strategy (RTS) games or selecting multiple files on the desktop. Users are generally consistent, so if a user has used Box Select or Lasso previously with a clear statement of their intent, future Box Selections or Lassos can be coded as such without requiring the user to say what they're doing.

Generally, if the user does the same action with both hands, and the actions have the same object, such as dragging the robots to one point with both hands, it's a single action, but coded as 2-handed. If the actions have different objects, such as dragging one group of robots with one hand and a different group with the other hand, it's two drag actions, although they may have very similar or the same ending time.

Gestures

Drag - User places their finger or fingers down and moves them to a different location while touching the screen. The drag ends when the user lifts their finger, so drawing a circle around robots and then to another location is a single drag action, not a lasso followed by a drag.

- If two or more drags are performed at the same time on the same object, code it as one instance of two-handed drag, not two instances of drag.

- If two or more drags are performed at the same time, but have different objects, code it as two drags with different objects.
- If the user is drawing a specific form, use the "draw" flag of the drag code, and describe what they drew.
- If a user lifts their finger while drawing something in multiple parts, such as writing out words, or drawing arrowheads on lines, code each time they lift their finger as a separate drag.
- A person dragging the side of their finger is an "other", not a "drag"

UI - User interacts with the screen while describing a UI element such as a menu, drop-down box, on-screen joystick, or similar. Also used for description of a sequence of events, such as the user saying "I would pull up a menu" or "I would type in a command".

- Do not code the user describing something that they would like to have in the UI unless they are describing how they would use it to issue commands for the current task.
- If the user taps for a menu, or drags a menu down, code it as UI, not as a tap or drag.
- Code UI actions at the end of the action, not at the end of the user description.
- If the user taps the same button many times, code each time (don't combine them)

Tap - User places finger on screen and lifts it immediately, without moving it a significant distance. Taps longer than one second are "Holds", as defined below.

Double-tap - User taps twice with both taps falling within an inch of each other and within one second, and without describing tapping on something else, such as tapping to bring up a menu and then tapping something on that menu.

- Taps for actions such as drawing a dotted line should be coded as individual taps. Double-taps are coded as taps with the -c (for "count") flag set to 2.

Triple-tap - As with double-tap, but with three taps.

- Anything beyond three taps should be coded as individual taps, but obviously with close-together time codes. Quadruple-taps and beyond are also comparatively rare.

- Triple-taps are coded as taps with the -c (for "count") flag set to 3.

Hold - User places one finger on the screen and leaves it there without moving for more than one second. Code the time at the end of the hold, when they lift their finger.

- Holds are coded as taps with the -h (for "hold") flag.

Pinch - User places two fingers on the screen, resulting in two points of contact, and moves them towards each other in a line.

- The Pinch code has flags for specifying the number of fingers and hands used, please code them appropriately.

- Multiple pinches at the same time on the same object should be coded as a single pinch instance. For a case where a user makes a pinch gesture with both hands at once, code it as a single pinch, with two hands and four fingers.

- Multiple pinches at the same time on different objects should be coded as separate pinches, with the -o/object flag describing which things were pinched.

Reverse pinch - User places two fingers on the screen, resulting in two points of contact, and moves them away from each other in a line.

- The Pinch code has flags for specifying the number of fingers and hands used, please code them appropriately.

Lasso - User touches on or near the robots and moves their finger in a closed shape (usually a circle or oval) around or over some set of the robots. Immediately precedes issuing some other command to the selected group.

- If it is unclear whether the user intended to perform this action as a selection, code it as a drag.

Box select - User touches on or near the robots and drags their finger diagonally across the robots in a straight line, then lifts their finger. Immediately precedes issuing some other command to the selected group.

- If it is unclear whether the user intended to perform this action as a selection, code it as a drag.

Voice command - Statements addressed to the robots, such as "Robots, go to area A", or statements such as "I would tell the robots to form a square". Code the time of voice commands at the end of the user's full sentence.

- If the user says something like "Robots, do X and then do Y", that's one voice command, don't break it into two commands at the "and then".

Other - Anything not listed above, but intended by the user as a command for the robot. This code has a description field in the coding program, please use it to describe the action.

- Gestures over the screen, without contact, and that don't match the any of the other commands should generally be coded as "Other", but only if they are clearly intended as a command to the robot, not e.g. pointing at something on the screen or indicating the screen itself.

Appendix B

All Task Slides

1 Robot Case

Figure B.1: One robot: Move to A

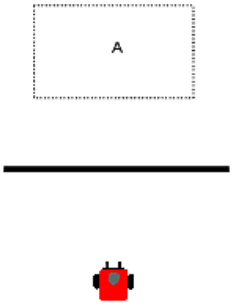


Figure B.2: One robot: Move to A with wall

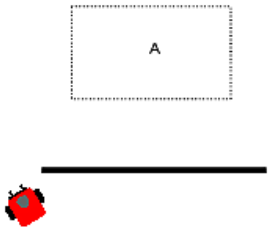


Figure B.3: One robot: Stop the robot

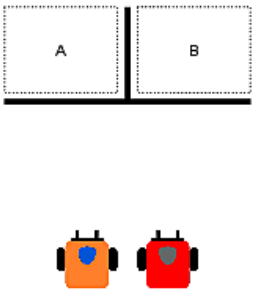


Figure B.4: One robot: Orange to B, Red to A

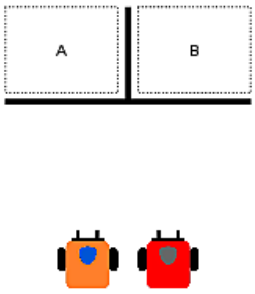


Figure B.5: One robot: Orange to A, Red to B

Figure B.6: One robot: Divide group

Figure B.7: One robot: Move the crate to A

Figure B.8: One robot: Mark defective robot

Figure B.9: One robot: Remove defective robot

Figure B.10: One robot: Patrol the screen border

Figure B.11: One robot: Patrol area A

10 Robot Case

Figure B.12: Ten robots: Move to A

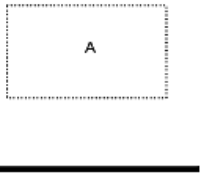


Figure B.13: Ten robots: Move to A with wall

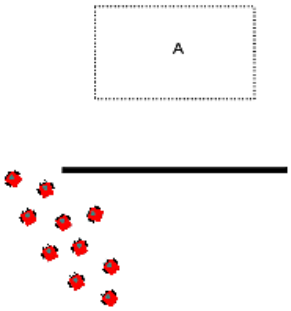


Figure B.14: Ten robots: Stop the robots

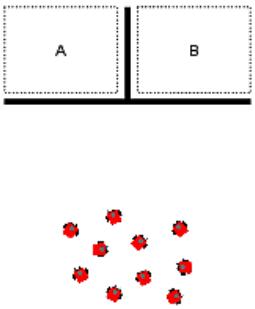


Figure B.15: Ten robots: Divide around obstacle

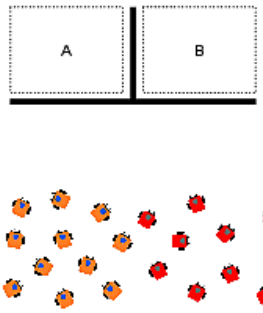


Figure B.16: Ten robots: Orange to B, Red to A

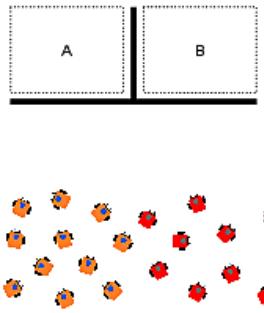


Figure B.17: Ten robots: Orange to A, Red to B

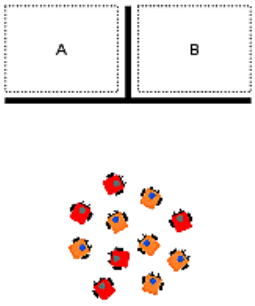


Figure B.18: Ten robots: Orange to A, Red to B

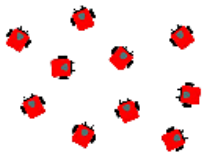


Figure B.19: Ten robots: Divide group

Figure B.20: Ten robots: Combine groups

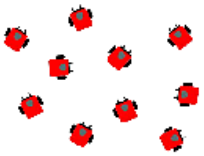


Figure B.21: Ten robots: Form a line

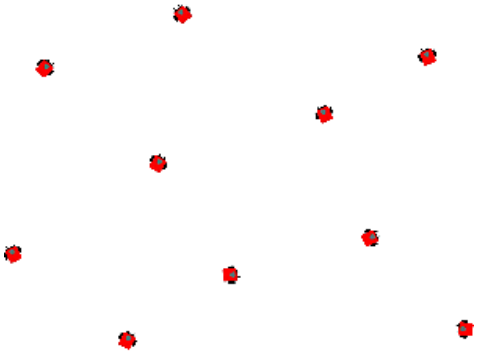


Figure B.22: Ten robots: Form a square

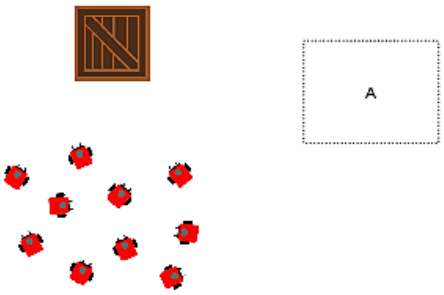


Figure B.23: Ten robots: Move the crate to area A

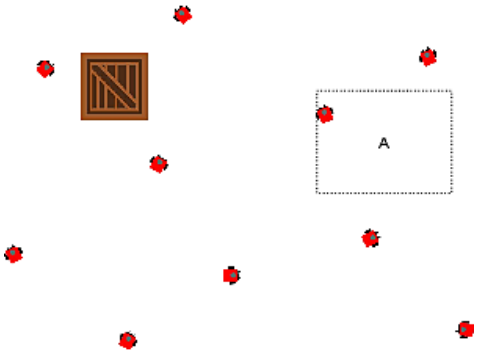


Figure B.24: Ten robots: Move the crate to area A

Figure B.25: Ten robots: Mark the defective robot

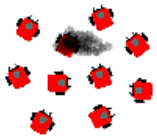


Figure B.26: Ten robots: Remove the defective robot

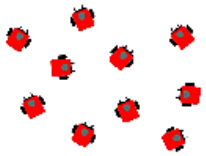


Figure B.27: Ten robots: Patrol the screen border

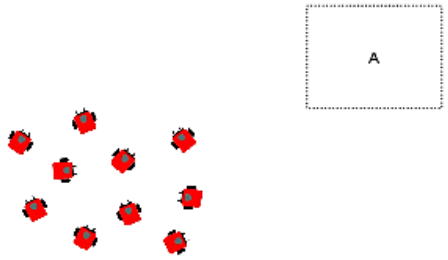


Figure B.28: Ten robots: Patrol area A

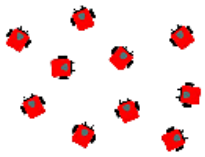


Figure B.29: Ten robots: Disperse over screen

100 Robot Case

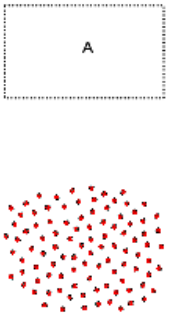


Figure B.30: 100 robots: Move to A

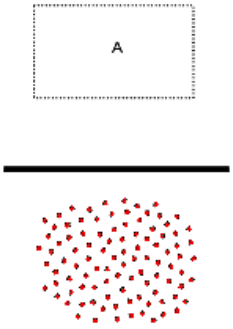


Figure B.31: 100 robots: Move to A with wall

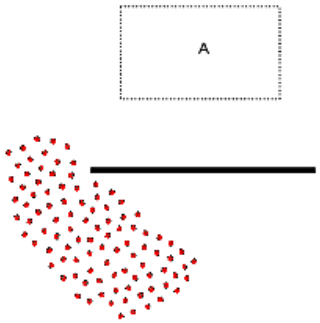


Figure B.32: 100 robots: Stop the robots

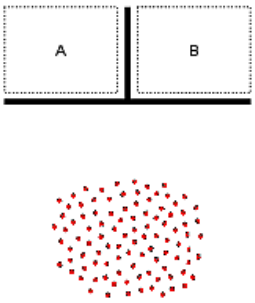


Figure B.33: 100 robots: Divide around obstacle

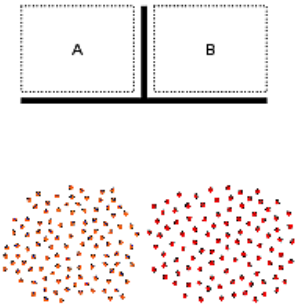


Figure B.34: 100 robots: Orange to B, Red to A

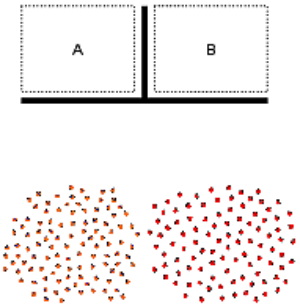


Figure B.35: 100 robots: Orange to A, Red to B

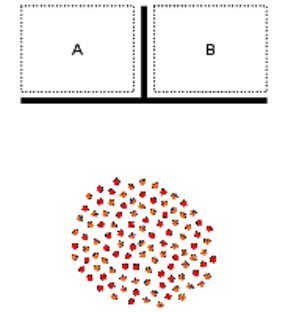


Figure B.36: 100 robots: Orange to A, Red to B

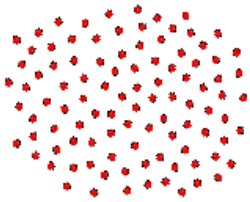


Figure B.37: 100 robots: Divide group

Figure B.38: 100 robots: Combine groups

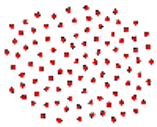


Figure B.39: 100 robots: Form a line

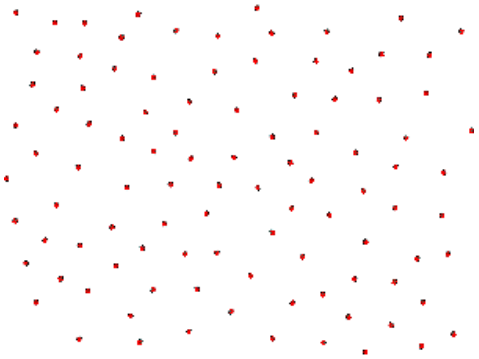


Figure B.40: 100 robots: Form a square

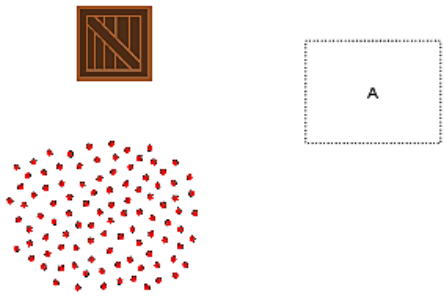


Figure B.41: 100 robots: Move the crate to area A

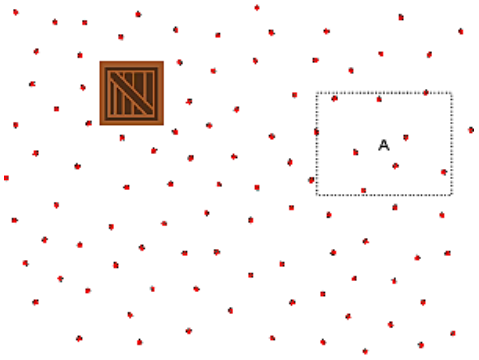


Figure B.42: 100 robots: Move the crate to area A

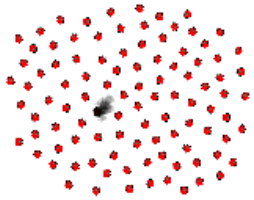


Figure B.43: 100 robots: Mark defective robot

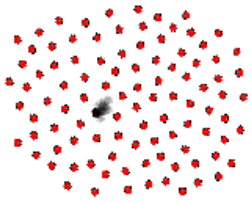


Figure B.44: 100 robots: Remove defective robot

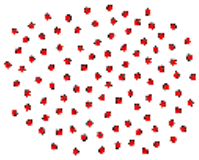


Figure B.45: 100 robots: Patrol the screen border

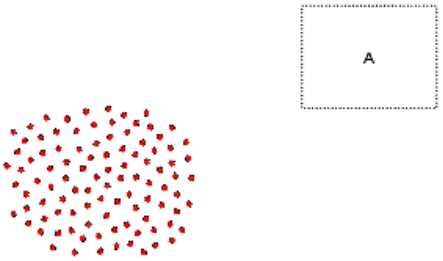


Figure B.46: 100 robots: Patrol area A

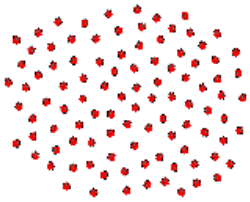


Figure B.47: 100 robots: Disperse over screen

1000 Robot Case

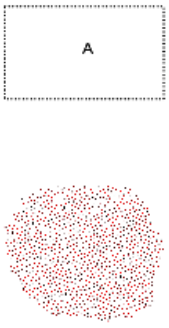


Figure B.48: 1000 robots: Move to A

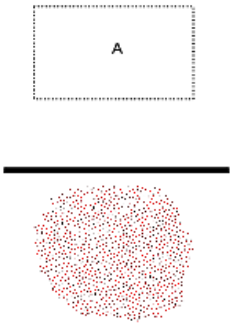


Figure B.49: 1000 robots: Move to A with wall

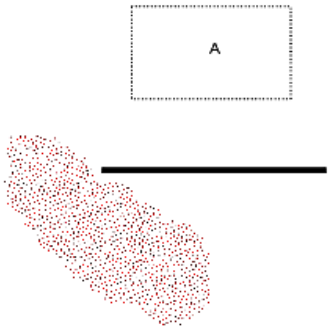


Figure B.50: 1000 robots: Stop the robots

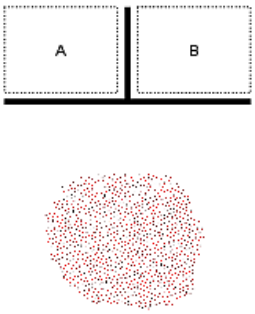


Figure B.51: 1000 robots: Divide around obstacle

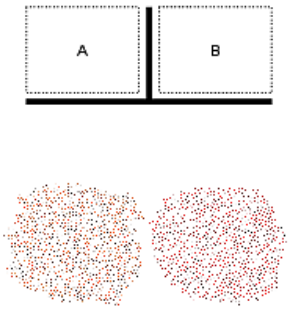


Figure B.52: 1000 robots: Orange to B, Red to A

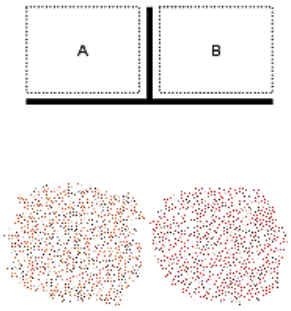


Figure B.53: 1000 robots: Orange to A, Red to B

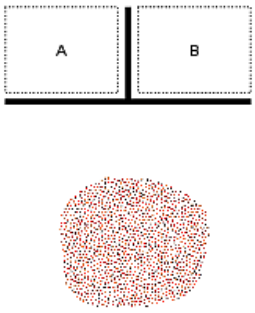


Figure B.54: 1000 robots: Orange to A, Red to B

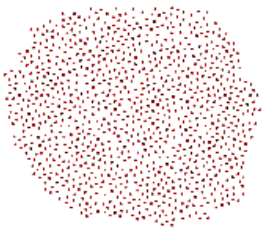


Figure B.55: 1000 robots: Divide group

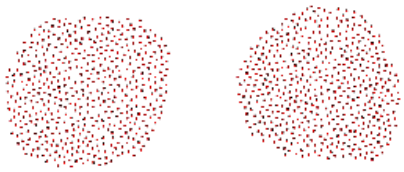


Figure B.56: 1000 robots: Combine groups

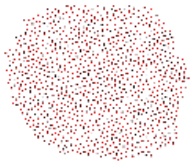


Figure B.57: 1000 robots: Form a line

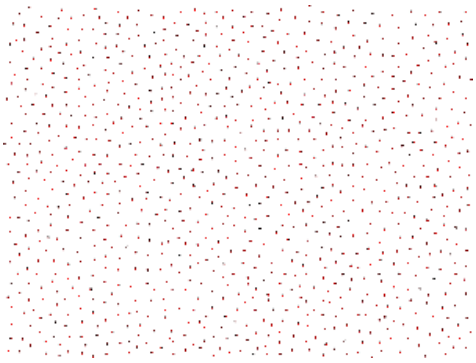


Figure B.58: 1000 robots: Form a square

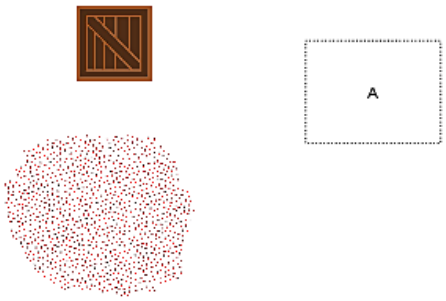


Figure B.59: 1000 robots: Move the crate to area A

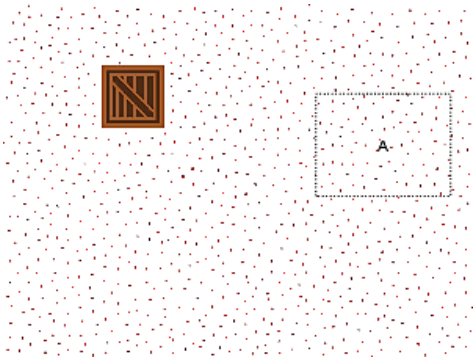


Figure B.60: 1000 robots: Move the crate to area A

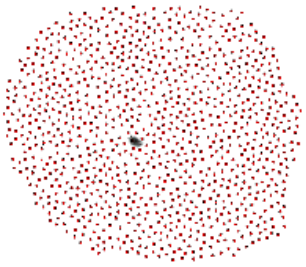


Figure B.61: 1000 robots: Mark defective robot

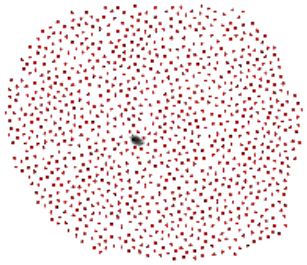


Figure B.62: 1000 robots: Remove defective robot

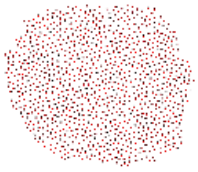


Figure B.63: 1000 robots: Patrol the screen border

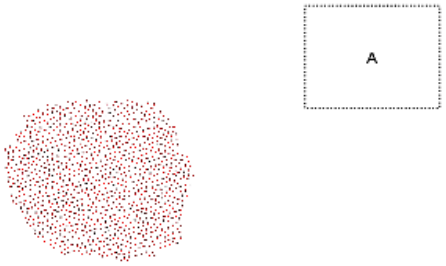


Figure B.64: 1000 robots: Patrol area A

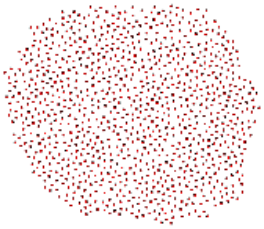


Figure B.65: 1000 robots: Disperse over screen

Unknown Number of Robots Case

Figure B.66: Unknown number of robots: Move to A

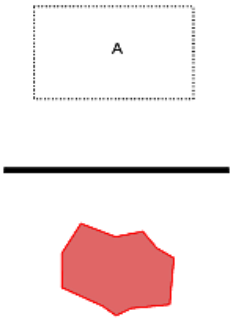


Figure B.67: Unknown number of robots: Move to A with wall

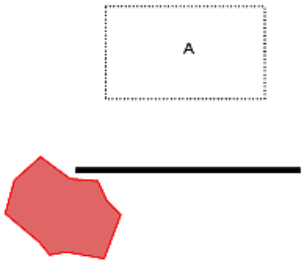


Figure B.68: Unknown number of robots: Stop the robots

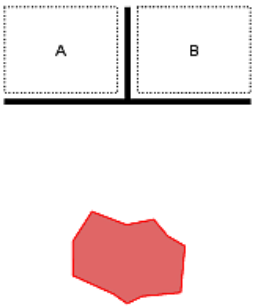


Figure B.69: Unknown number of robots: Divide around obstacle

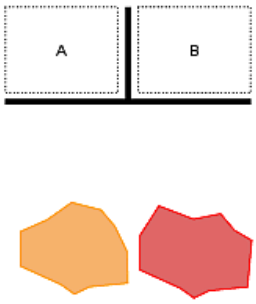


Figure B.70: Unknown number of robots: Orange to B, Red to A

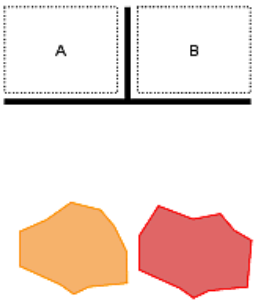


Figure B.71: Unknown number of robots: Orange to A, Red to B

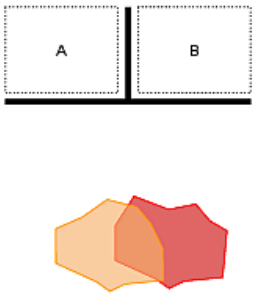


Figure B.72: Unknown number of robots: Orange to A, Red to B

Figure B.73: Unknown number of robots: Divide group

Figure B.74: Unknown number of robots: Combine groups

Figure B.75: Unknown number of robots: Form a line

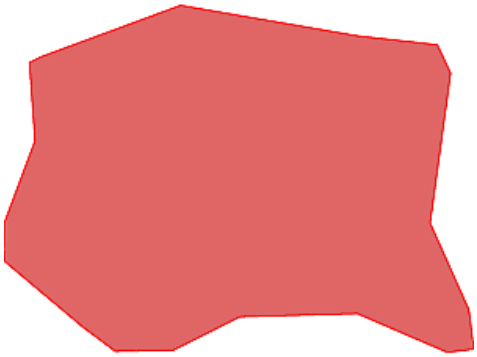


Figure B.76: Unknown number of robots: Form a square

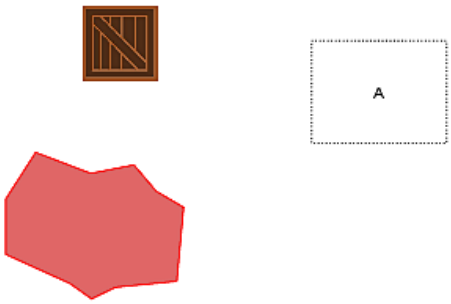


Figure B.77: Unknown number of robots: Move the crate to area A

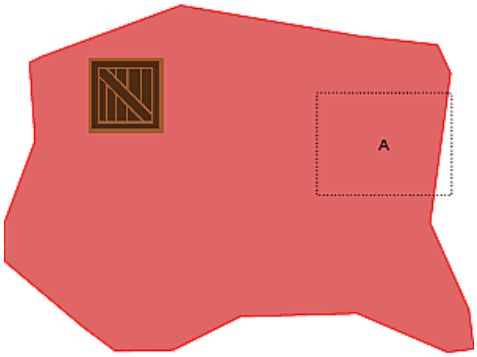


Figure B.78: Unknown number of robots: Move the crate to area A

Figure B.79: Unknown number of robots: Patrol the screen border

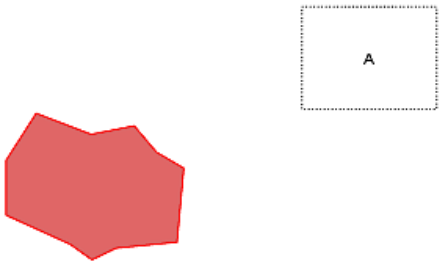


Figure B.80: Unknown number of robots: Patrol area A

Figure B.81: Unknown number of robots: Disperse over the screen area

Biographical Sketch

Abraham M. Shultz received his bachelor's degree in computer science from Worcester Polytechnic Institute in 2004. After graduation, he spent 5 years in industry as a software developer, working for Radiospire, RSA, and EMC^2. He joined the Robotics Laboratory at UMass Lowell in 2010. Abraham received his master's degree in computer science from UMass Lowell in 2016 for work on simulating mouse cortical cultures and integration of cortical cultures with a robot arm and camera to build a cybernetic system. The work described in this thesis is an outgrowth of a personal project to make swarm robotics affordable for home experimenters, a topic of which he is still fond.