Command Language for Single-User, Multi-Robot Swarm Control

Abraham Shultz

February 15, 2018

Abstract

Command and control systems designed for a single operator to operate a single robot do not scale to control of swarms [Wang et al., 2009]. User interfaces that require the user to attend to each robot overwhelm the controller when the number of robots increases beyond 12 or 13 for UAVs and 3-9 for UGVs [Wang et al., 2009]. As robot swarms increase beyond these bounds, the control system must shift to controlling the swarm as a single entity. However, the form this interface should take to permit easy and understandable control of the swarm is largely undefined.

Previous work in HRI shows that multi-touch interfaces allow a scalable and direct mapping between the desires of the user and sequences of commands to the swarm [Micire et al., 2009b]. By defining a mapping from user interface gestures to individual programs loaded on each robot, we can allow an individual to control arbitrarily large, heterogeneous swarms. This thesis presents an interface which extends previous work on multitouch interfaces for small swarms 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 possible failure of individual swarm robots.

The contributions of this thesis are a new swarm hardware platform, software to support it, and a user interface which converts user commands into programs for each robot in the swarm. The new swarm platform combines a wifi-enabled microcontroller with commodity mobility platforms sourced from children's toys to allow large swarms to be built at a low cost. Because the design does not bind the implementation to a specific mobility platform, heterogeneous swarms can be constructed, using different toys. The hardware is also adaptable to new toys as older ones become unavailable, and could be used with custom-designed or 3-D printed mobility platforms. In order to maintain the low cost, sensors for the swarm robots are simulated with a top down camera. The use of simulation allows individual swarm members to be constructed cheaply in the present, while leaving the path clear in the future to push computation to the individual members as computer hardware becomes cheaper and smaller.

1 Intro/Research Statement

1.1 Problem 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. Distributed control systems do not have the single point of failure that centralized models have. In order to create reliable and useful swarm robotic systems, users must be able to specify a desired end state of the system that the swarm can converge to 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 to use by non-programming end users [Palmer et al., 2005b]. The motivating examples of uses for swarms are task oriented, such as sending swarm robots into disaster zones to search for survivors. While first responders are frequently trained in many specialized skills to perform their jobs, adding programming swarm robots to their training would be impractical. Even if first responders could be expected to develop software in a disaster zone, the situation frequently develops faster than the build/test cycles of software development can react. 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. Another possible example of an application for swarm robotics is cleaning, whether in a user's house, or in a more hazardous contaminated area.

In either case, it is desirable for the user to be able to release the robots into the region with minimal oversight, and rely on the robots to discover what areas need cleaning, and to allocate robots to clean them. Modern household cleaning robots, such as the Roomba, are restricted to floors, and even specific types of flooring, but a heterogeneous cleaning swarm might include robots for cleaning windows and dusting surfaces as well. Currently, this kind of whole-house cleaning does not require programming the cleaning tools in advance, and any product which does require such a task is unlikely to succeed in the market.

In order to remain useful in real-world applications, this work makes certain assumptions about the robots and the swarm. Networking between the robots is expected to be unreliable. Individual robots have limited transmission power over radio or infrared light links. Network links are also frequently attenuated by distance or intervening objects. As a result, when robots spread out into an area, some robots may not be directly reachable from others. Any software that purports to control a swarm under these conditions cannot rely on perfect connectivity for its operation.

Robots' perception of their environment is frequently limited. Objects can block the sensing field of sensors. Even without obstructions, most sensors have a limited effective range. Because the real world is dynamic, robots can

make only limited assumptions about what they cannot sense, and can only sense a limited area. Algorithms to control robots must primarily make use of locally sensed information, and secondarily make use of information received in communications from other robots.

Because of the limited range of their sensing and networking, the localization of the robots may also be unreliable. GPS provides a global coordinate system in which robots can easily localize, but GPS signal is absent indoors, under dense tree cover, and in many other areas where it would be desirable to use swarm robots. The GPS signal is also weak, and so can easily be jammed by ambient RF interference or by malicious actors.

Robots can also fail. The harsh conditions of the WTC rubble pile destroyed many of the robots sent in [Micire and Murphy, 2002]. In order to remain robust in the face of failures, algorithms should not be developed to depend on the perfect functioning of any individual robot. Rather, the behavior should emerge from the interaction of multiple, interchangeable robots, and tolerate the loss of individual robots up to some limit. In a situation where many or all of the robots attempting to complete a task are destroyed or disabled, it would be unreasonable to expect the task to be completed, but ideally, the bounds on how many robots are required to complete the task should be knowable.

Cooperating systems can be broken down into two major classes, those that intentionally cooperate, and those that cooperate in the manner of ants or other swarm insects, without explicit agreements and planning [Parker, 1998]. The swarm-like approaches generally assume that the robots are homogeneous and all have the same control software. For the work described in this proposal, it may be that neither of these assumptions hold. The proposed hardware enables the creation of highly heterogeneous swarms, including variation in scale of the robots involved. Because the proposed control software generation scheme is intended to potentially result in different software for each robot, even robots with the same hardware may have different control programs.

1.2 Hypothesis

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. This transition point will be apparent because of a change in the gesture set that the user uses 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 use 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, the 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 the user intent to interact with individual robots as opposed to interacting with the swarm as a whole.

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 et al., 2012]. Scaling the representation of the robots up, relative to the map, will make the robot representations overlap unrealistically and obscure the map. Instead, we propose 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 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 et al., 2014]. Altering how the user interface displays the location of the robots in the swarm will affect the transition point. More specifically, a display which 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 different gestures.

Further, it is hypothesized that it is feasible to convert user commands into programs for each robot which will converge 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.

2 Related Work

2.1 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 isn't kept constant, the robots will crowd each other, and so 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 also multiplied by the number of robots in the swarm.

The robots used in most swarm work are of a sufficiently small size that many of them can fit in a room. In addition to budgetary constraints, interaction with an environment built for humans places an upper bound the scale of the individual swarm members. For example, typical indoor doorways are around thirty inches wide, so a robot would have to be less than thirty inches 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 a result of these bounds, swarm robots are mostly between 1cm^3 and 0.3m^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, then, is to put all of the same parts as non-swarm mobile robots: a mobility platform, a power supply, a processor, some sensors, and a communication system, 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.

2.1.1 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 et al., 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 et al., 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.

Alice, by Caprari *et al.* [Caprari et al., 1998] 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. The processor used in Alice is relatively underpowered compared to modern processors at the same price point and power consumption. Alice robots are no longer available for purchase. The AmIR robot was similar to Alice in size and capability, but with a more modern processor [Arvin et al., 2009]. There is no evidence that AmIR was ever widely

available.

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 li-po batteries. Unfortunately, Jasmine units cost about 100 Euro ($111 USD) each when they were available, and they are no longer available for purchase. The plans and information required to reproduce Jasmine units are available for free at Swarmrobot.org. Assembling a Jasmine robot is not beyond the reach of competent electronics hobbyists, but it does require some unusual build processes, such as grinding down the cases of certain electronic parts and filling holes in the PCB with solder to prevent light leaks. The chassis of Jasmine is also a custom mechanical assembly, rather than a commercially available product.

InsBot was a small robot, measuring 41mm x 30mm x 19mm, that was designed to interact with cockroaches [Colot et al., 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. InsBots were never commercially available, and each required approximately 6 hours of work to assemble by hand. However, the construction process appears to have been relatively straightforward.

Even when they are commercially available, most existing swarm robots are too expensive to build a large swarm. The Epuck from EFPL is approximately 800 Swiss francs ($810 USD) per unit, so the cost of maintaining a large swarm can become daunting quickly. The high price of the Epuck is a result of its extensive suite of sensors, including a camera and 360° IR range sensor and communication system. Epucks also require a fairly smooth operating surface. The Epuck uses differential drive, and allows the front or back edge of the robot to serve as a skid. Due to the relatively sharp corner of the lower edges of the robot, the Epuck can become stuck on 2-3mm high obstacles.

The r-one research robot is cheaper than the Epuck, at approximately $220 USD per unit [McLurkin et al., 2013]. The developers of the r-one position it as a more-featureful and less expensive alternative to the Epuck ($810, cannot sense neighbors without additional hardware), Parallax's Scribbler ($198, minimal sensors), the iRobot Create ($220, requires additional hardware to be programmable), the K-team Khepera III ($2000), and the Pololu 3pi ($99, minimal sensors). The main advantage in sensing that the r-one has over these other platforms is neighbor sensors and ground truth position sensing, both of which are implemented on the r-one using infrared. The design of the r-one is open source, but it does not appear to be commercially available as of this writing.

The Harvard Kilobots are a more recent entry to inexpensive swarms, and have been produced in large quantities [Rubenstein et al., 2014]. Kilobots contain about $15 worth of parts, but a 10-pack sells for 1100 Swiss francs, or about $112 (US) 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.

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 RC car. At the time of this writing, the car used in COTSBots is moderately expensive for a toy car, although quite cheap for a research robot, costing a little over $100USD per unit. COTSBots use TinyOS, a modular and event-driven framework for developing node software [Levis et al., 2005]. TinyOS is written in a dialect of C called nesC. The motor and mote boards communicate using a messaging layer. The motor driver board is not commercially available, but can be custom-built by board fabrication companies, without the researcher having to assemble it by hand. The second COTSBots is larger, and will be discussed in the next section.

2.1.2 Room-sized Swarms

One potential problem with extremely small swarms is that while the robots may scale down, the scale of obstacles they have to traverse may not scale with them. As previously mentioned, the Kilobots require a smooth surface, and the Epuck can be stopped by obstacles no more than a few millimetres tall. 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. 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 et al., 2010]. The size and complexity of the MarXbots, as well as their powerful computer, renders the individual robots quite expensive.

Swarmanoid extends the interlinking mechanism of MarXbot to a heterogeneous swarm with three different types of robots [Dorigo 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.

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 et al., 2007, Tammet et al., 2008, Olson et al., 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.

2.2 UI 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 [Micire et al., 2009b]. 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 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 et al., 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 with drawing gestures, and paths with fluid strokes, rather than e.g. selection of vertices bounding an area. The use of multi-touch interaction is desirable because one-at-a-time selection doesn't 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.

One approach to getting feedback from a swarm was the development of the Swarmish sound and light system [McLurkin et al., 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. As a result, most of the interactions were remote. 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 can vary in instrument, pitch, duration, and volume, in addition to 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 you can look at 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 et al., 2003]. 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 et al., 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 10 being a fully automatic system with no humans involved, and level 1 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. At level 1, 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

autonomy, neglect time increases. The fifth level of the autonomy scale is a sort of operation by consent model, where the computer chooses a route and executes it if the human permits it. The ninth level is the inverse of the fifth. Rather than asking the user for consent to act, the robot acts and informs the human only in exceptional cases. 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 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 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 et al., 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. 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 et al., 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 allows the user to handle novel exceptions. Knowledge gives the user an understanding of how the system works, and can apply that understanding 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 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, as in the case of Swarmish and the robot-as-pixel UI designs. 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 bots 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 et al., 2004]. For example, if a robot can send video and range information, the video and range 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 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 et al., 2009a]. 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 et al., 2009].

2.3 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.

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 that can be codified into a compiler. As a consequence, input in the command language defines a program which is compiled and loaded onto the robots. Another possibility is the composition of preexisting behaviors that each satisfy part of the user's desired behavior. 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.

2.3.1 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 et al., 2000]. The motivation for AC is that while it may be possibly 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, and if AC promises to get useful work out of smart dust, then it also has some applicability to larger swarm robots.

There are several languages intended to program amorphous computers. "Proto" is a language for a continuous plane spatial computer [Correll et al., 2009]. Because the devices are distributed over a plane, the difficulty in communicating between any two devices is a function of the distance between them, much as with RF or other radiative communications. In Proto, the behavior of regions of space is described by the programmer, and the description is transformed into local actions for the network of devices. 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. Swarm robots are mobile, so some swarm algorithms can be implemented as a description of constraints on the robot's state, such as "the robot must have communications links to no more than 2 and no less than 1 other robots", and a command to move randomly unless the constraint is satisfied. Within a bounded environment, such an algorithm can be shown to converge to satisfy its constraints [Correll et al., 2009].

Proto also has considerable appeal as a programming language for swarm control development because of the layering in its structure. Proto is designed to map from behavior of regions at the global level to programs for discrete points at the level of individual devices [Beal and Bachrach, 2006]. 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.

Origami Shape Language (OSL) uses the abstraction of a foldable sheet to form shapes, inspired by both origami and the folding of epithelial cells during the development of biological organisms [Nagpal and Mamei, 2004, Nagpal, 2001]. Regions and edges on the sheet can be defined by propagation of morphogens, and folds along the edges result in the development of the final form. Because of the use of morphogens and local communication between the agents on the sheet, there is no need for a global controller to dictate the development of the final form. Further, because the high-level description of the desired form does not involve abstractions of the underlying modules, OSL could operate on interlocking modular robots, actuated flexible materials, swarms, or other kinds of computational media. In fact, the flexible sheet could be assumed to be virtual, and the resulting motions of the sheet could be translated into motor commands to configure swarm robots into specific arrangements in space.

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.

2.3.2 Pheromone Approaches

The abstractions of GPL bear a very strong relation to pheromone robotics. Pheromone robotics refers to a metaphor for 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 is 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 which are detected by chemical sensors (for an exception, see [Hayes et al., 2001]). Each pheromone can have properties such as diffusion and evaporation rates that result in the pheromone spreading in space or gradually disappearing. In addition to its properties, the pheromones may have other characteristics which robots can sense. For example, a robot may emit a pheromone which 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 pheremone gradients [Payton et al., 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 signalling and associated behaviors, controlling the swarm to perform a task with these primitives is still done by hand [Payton et al., 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 which the robots can refer to when needed. 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. 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 et al., 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. The system under discussion was developed to have the task implied by the behaviors available for the agents, rather than generating the program from a higher-level specification, such as the form of the structure to be built.

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 et al., 2014].

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. 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. 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.

This vector field interface does not directly map to programs on the robots. Instead, the central computer maintains the vector field representation and commands the individual robots. Vector fields also do not allow the assignment of tasks to robots, but allows the user to directly control the motion of the robots. In order to convert from a task-based user interface to a vector field representation, the task would have to be converted into a series of changes to the field. 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 et al., 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.

2.3.3 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. The first five are common in amorphous computing as well, but the last three are not. 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, compostional approaches have been proposed in control-theoretic terms as well as pheremone-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 pheremone robotics [Belta et al., 2007].

The pheremone based approaches to swarm programming are sufficient for relatively complex behaviors. Quorum sensing is used to detect whether the local agent count is sufficient for a task. By detecting the presence of sufficient robots to perform a task, the robots can allocate themselves to tasks in a just-in-time manner, rather than being pre-allocated when the task is designed. Decentralizing the selection of robots, in turn, may be more robust against failures of individual robots, as it uses the robots that are in the right place at the right time, rather than waiting for specially assigned robots. However, under sufficiently bad conditions, a sufficient quorum may never arrive, deadlocking the task. In combination with domino timing, where completion of each phase triggers the next, locking at any step could then deadlock the entire process unless another mechanism detects and corrects it.

Individual robots can cooperate without communication to push an object to a beacon based on simple behaviors [Chen et al., 2015]. Each robot has two simple behaviors. If the view of the beacon is blocked, and the robot is next to an object, the robot pushes on the object. If the robots can see the beacon, it wanders and avoids obstacles. The sum of the two behaviors results in a net pushing force on the side of the object opposite the beacon, which moves the object to the beacon. There do exist certain pathological shapes which the system cannot move towards the beacon, but it is demonstrated to work for all convex shapes.

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 primitive 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 one, 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.

2.3.4 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 of programs for swarm robots consists of iterating between composing sets of primitives and observing 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 et al., 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 useless 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 9 classes [Harriott et al., 2014].

- Human attributes - Interaction, trust, intervention frequency
- Task performance - Ability to accomplish task, speed, accuracy, cost
- Timing - Command diffusion lag, behavior convergence
- Status - Battery life, number of functioning members, stragglers
- Leadership - Interaction between special members of the swarm and others
- Decisions - Action selection, likelihood that the correct action is chosen
- Communication - Speed, range, network efficiency
- Micromovements - Relative motion of individual swarm elements
- 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 hasn't moved any units to the goal yet.

Ideally, it would be possible to recognize and evaluate performance on subproblems. 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 et al., 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 et al., 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 which 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 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, attacking together, etc. The resulting controllers only controls 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.

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 early version of the evolved aggregation controllers allowed the evolved motion strategy to acquire a high fitness by spinning in place, and so the fitness function required modification to prevent robots from simply spinning. 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.

2.3.5 Domain-Specific Languages for Swarm Robotics

Proto and other programming languages for amorphous computers provide abstractions for computation performed on homogeneous spatially-distributed computing nodes, but do not generally support motion of nodes within the space. Other versions of tuple-space based amorphous computation include motion of the agents, but do not explicitly support heterogeneity [Viroli et al., 2012].

The Voltron programming language provides what its authors describe as "team-level programming" for autonomous drones [Mottola et al., 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 et al., 2011]. The framework allows the composition of behaviors described at the level of individual robots. Rather than each robot performing series 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 et al., 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 which 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 et al., 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 neighbours of the robot executing the program, which usually consists of all robots within communication range. Using the neighbours abstraction, the robot can, for instance, query all its neighbours about the value of a sensor precept at their location, in order to build a local map of the intensity of the sensed quantity.

3 Proposed work

For the purposes of this work, the tasks that users will be asked to complete consist largely of directing the motion of the swarm. While a task is generally defined as a unit of work to be accomplished, motion within space is not usually considered a task. Instead, motion within a space is usually performed as part of a task. For example, searching an area requires moving over the area to be searched. Specialized tasks will likely require specialized interfaces to provide the information that is specific to that task. To attempt to elicit specifications for such interfaces, users will be requested to think aloud while they are using a proposed interface. It is hoped that users will describe how they would invoke the more specialized components of the desired interface.

3.1 Compilation of user commands into robot programs

In order to create a mapping from commands, as issued by the users, to a set of individual robot programs for multi-robot command and control, the gesture set for the commands must first be specified. Multitouch gesture sets as a command language for controlling robots have been developed by an empirical process with naive users [Micire et al., 2009a]. These gestures frequently consisted of sequences of gestures that roughly fit the linguistic structure of a sentence, with the first gesture indicating the subject of the sentence, and the next gesture indicating a verb and possibly an object. As a consequence, the gestures form a sort of language, and commands are sentences in the gesture language. A user might select a group of robots by circling them. These robots would be the subject of the "sentence". Then the user might draw a path or tap a spot on a map, indicating that the robots should "go here". Taken all together, the sentence could be read as "these robots, follow this line to this location", with the pronouns disambiguated by the locations of the command gesture on the screen.

One way to define such an interface would be to select a fixed set of gestures, and then train the users to use those gestures when they interact with the system. However, if the system is not one that the users use frequently, they will forget the training. Since the advent of multitouch interfaces for smartphones and the trackpads of some laptops, many users already have some prior experience with

multitouch gesture controls in everyday life. Multitouch gestures can also be imitations of the way the users would expect to interact with material objects. For example, zooming out of a map view by pinching two fingers together imitates the distant points of the map becoming closer together, indicating outward zoom, and spreading the fingers imitates stretching a smaller region to cover the screen for an inward zoom. From these "naturalistic" expectations and daily use, users already have some idea of how a multitouch user interface can work. If the interface fulfills these intuitions, the users will find it easier to learn to use.

For the purposes of this research, an intuitive gesture language is one that is freely chosen by the majority of users. In [Micire et al., 2009a], for each available command, one or two gestures were used by 60% or more of the users. These gestures are the intuitive gestures for issuing the command. It is possible that there is no intuitive gesture for a given task. If no two users use the same gesture for the same commands, or, more generally, there is very poor inter-user similarity for the gestures chosen to issue a command, then there is not an intuitive gesture for that command.

In order to determine if the intuitive gestures change with the size of the swarm, tests will be conducted with varying swarm sizes performing the same tasks. The swarm used for these tests will be large, with the lower bound on its size being significantly larger than the number of fingers a user could potentially gesture with. In order to have a less tongue-in-cheek definition of "large", the scale required for a swarm to be considered "large" will be determined empirically. It is expected that there exists a transition point for the number of members in a swarm where users will stop interacting with the UI representation of the members of the swarm as individuals, and attempt to interact with the representations as groups or collections.

A large swarm is, then, a swarm with a number of members above the point at which such a transition occurs. 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, by depicting the group in different ways [Manning et al., 2015]. 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]. Other approaches include 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. 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. 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.

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.

3.2 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 interfacing with the robot's sensors, if any, and controlling the motors of the robot. The structure of the software framework is such that as available processing power on each individual 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 [Olson, 2011] on top of it, so that the central computer can localize them within the arena. 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, since the central computer knows the location, adding a range and bearing sensor that allows each robot to detect the distance and angle of the nearby robots is simple to implement. This functionality is available in hardware on Epucks 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. Because the system has an omniscient-view camera, other objects in the robot arena can also be tracked. The system currently treats blue tape on the floor of the arena as obstacles.

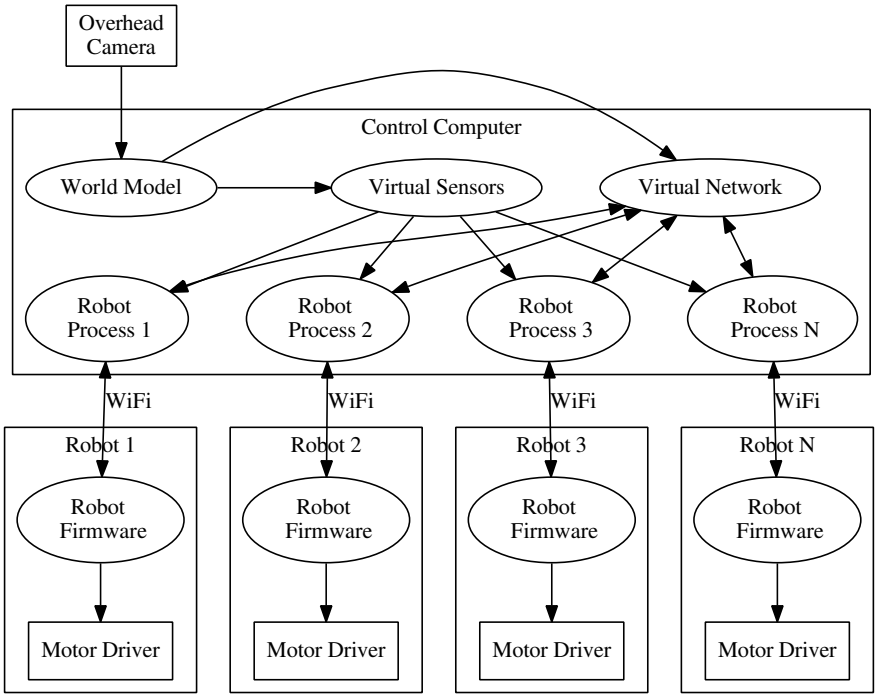

Figure 1: Overview of the planned framework. Rectangular nodes are hardware, oval nodes are software.

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, 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 with 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.

Similarly, 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, or the location of other robots, if the experiment calls for such a lack of information. However, the central computer can use the location information to create "virtual sensors" for each robot. The sensor precepts from virtual sensors would be simulated, but their magnitude or direction is be based on the location of a real robot relative to a real object in the experiment area. For example, collision avoidance between two robots can be implemented by a virtual sensor on each robot that indicates the direction and heading of the nearest 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. 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.

3.3 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 Hokuyo 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.

3.4 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.

3.5 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,

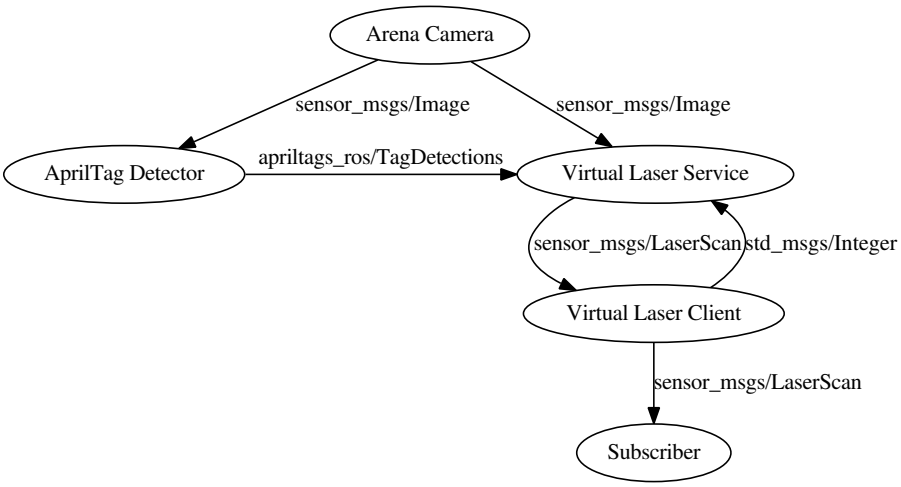

Figure 2: Data flow in the virtual laser service

the virtual network can provide full connectivity, range-limited mesh networking, directional beacons, or 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 to which it was transmitted may depend on the distance between the sender and receiver, or signals that pass through a virtual wall may have a reduced virtual signal strength and range.

3.6 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 is to allow 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.

The DRV8830 driver is a voltage-controlled motor driver. It accepts a single-

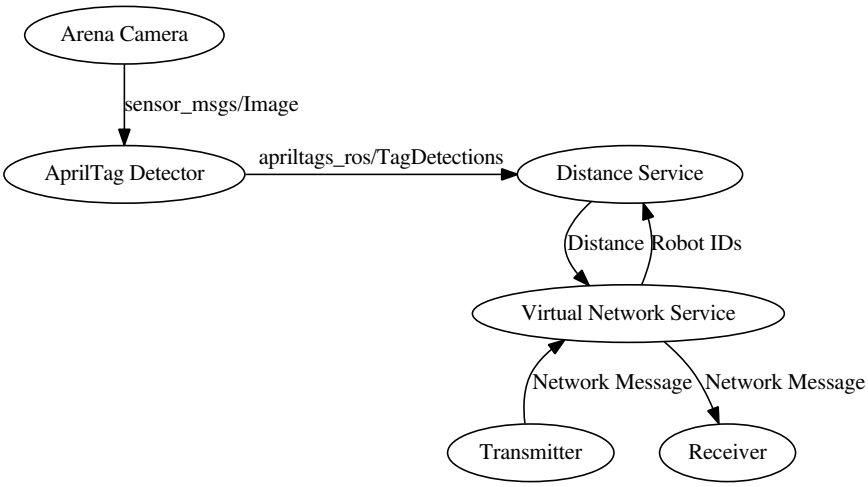

Figure 3: Data flow in the virtual network. The virtual network service can take the distance between the transmitting robot and the receiving robot into account when determining if the message is delivered.

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 1: Truth table for DRV8830 drive direction bits. Coast allows the motor to turn freely. Brake connects the motor leads, which results in braking using the motor's back-EMF

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 1.

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.

4 Timeline

4.1 September

Gestures - The experiment to collect the gesture data set will be performed in September.

Software - A preliminary gesture recognition system will be developed, although it may have to be amended to reflect the gestures collected in the experiment. Preliminary versions of the gesture compiler and the infrastructure for running the resulting programs on the robots are written, but they will be refined and extended.

4.2 October

Gestures - The experiment to collect the gesture data set will be completed, and analysis of the gesture data set will be done. The results will be written up and published if possible. At this point, it should be clear that there either is or is not a transition point where the user changes the gestures they use for a bigger swarm. If there is no transition, then it would seem to indicate that the same UI can be used for individual robots and swarms.

Writing - Assuming data set for gestures is useful for a paper, it will be written up and published. The resulting write up, as well as expansion on experimental methodology will be added to my thesis.

Software - The gesture recognizer will be extended with the gestures collected from the experiment. Automation will be put in place to have the pipeline from gestures to program deployment on the robots occur without user intervention.

4.3 November

As of this writing, the deadline to submit a Declaration of Intent to Graduate form for the fall is unknown (https://www.uml.edu/Registrar/Calendars/2017-fall-grad.aspx lists it as "TBD"), but it is likely in early December. It should be submitted sometime this month.

Software - Finishing and cleanup work will be done on gesture compiler. It may be possible to connect the UI from the experiment to the overhead camera for the swarm, resulting in a complete overhead view interface for real robots.

Writing - Notes on gesture compiler design and development will be added to thesis, and possibly developed towards a paper. Thesis defense presentation will be written.

4.4 December

Defend Thesis

Print copies of the thesis for submission to the library.

As of this writing, the deadline to submit the thesis/dissertation to the library is unknown, but it is likely around the last day of fall semester classes, Dec 13th.

5 Expected Contributions

5.1 Multitouch Gesture set for Swarm Control

This work will attempt to determine if there is an intuitive multitouch gesture set for a single user to command a large swarm of robots. For the experiment, there are 5 cases: 1 robot, 10 robots, 100 robots, 1000 robots, and cloud (unknown robot count). In all the cases except the cloud case, the robots are represented as individual robots on the screen of the interaction device. In the cloud case, the robots are presented as a cloud covering the area that the robots are present in, but without precise representation of each individual robot.

There are 18 tasks. Each user is assigned to one case, and then performs all the tasks using that case. Having each user work on one case is intended to prevent the user from being influenced by their memory of what they did in a different case, and so creating a consistency that is not a product of the requirements of the task and case. Some of the tasks do not make sense with the cloud case, such as interactions with a single robot, so users who are assigned to the cloud case will not view the tasks that are impossible in that case. Pilot runs of the study indicate that it can be completed in one hour.

The data set will be analyzed and compared to existing work on multitouch gestures for command and control. The analysis will also attempt to determine if the gesture set shows a transition point between many and few robots. The influence of the presentation of the interface on the gesture set will be examined. The gesture set will also be analyzed to determine if the size of the swarm has any effect on the gestures used, or on neglect of individual robots by the user.

5.2 Compilation of User Gestures into Robot Programs

The automatic conversion from a user-specified task into a set of command programs to be distributed to the swarm robots is still an open question. One recent approach uses a human-in-the-loop multitouch interface to allow a human to guide a swarm by drawing a bounding prism that the swarm attempts to remain within [Ayanian et al., 2014]. As the bounding prism moves, the swarm moves with it, with the individual robots performing obstacle avoidance. However, this work assumes that the individual swarm units can localize themselves, and that there is constant availability of communications between all swarm members and the central controller. For a number of reasons, these assumptions frequently fail to hold, and so a more robust system can be designed by assuming that localization and communication are difficult. This work will attempt to create an automated process by which user-specified behaviors of the swarm as a whole can be converted into programs that run on individual robots. The behavior of

the individual robots under this control should converge to the user-specified behavior without further communication from the central server.

5.3 Inexpensive Swarm Hardware

The previously described 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.

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 is 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 are constructed by developing a Commercial Off-The-Shelf (COTS) modular control hardware platform that can be attached to children's toys. Modified toys are an adequate substitute for custom mechanical assemblies, and permit easy experimentation with heterogeneous swarms. The use of children's toys as mobility platforms may also avoid the sensitivity to the work surface exhibited by the Kilobots and, to a lesser extent, the Epucks. 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 et al., 2014, Bergbreiter and Pister, 2003]. However, unlike the Spider-Bots and COTSBots, this work does 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

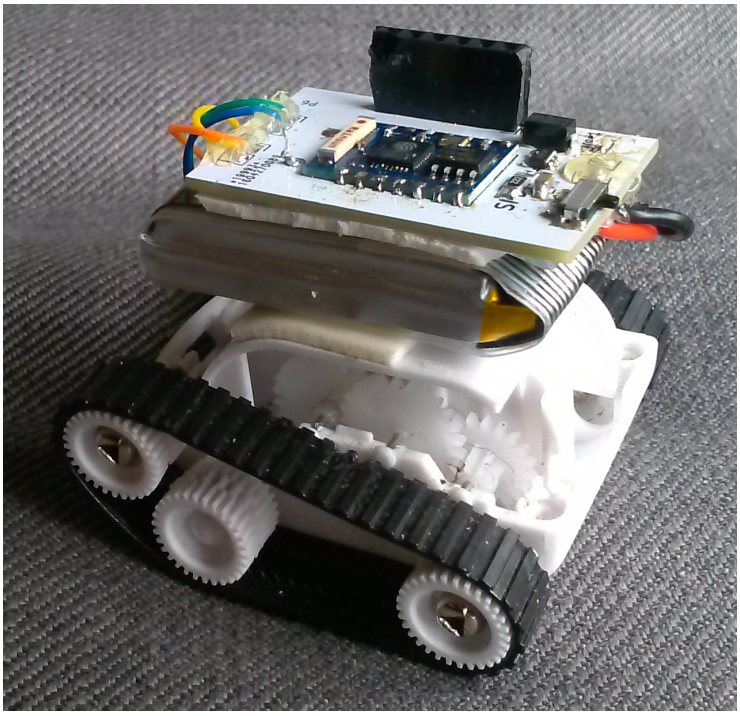

Figure 4: A tank-drive toy with a 3.7V lithium polymer battery and a control board mounted to it.

Part	Price	Quantity	Subtotal
Mobile Toy	14-20	1	20
Battery	2.54	1	2.54
Main PCB	2.75	1	2.75
ESP8266 Module	2-7	1	7.0
DRV8830	2.30	2	4.60
MIC5319-3.3YD5	1.36	1	1.36
MCP73831	0.58	1	0.58
Total			27.83-38.83

Table 2: All prices are for single quantities of new parts. It may be possible to get bulk discounts, especially on the toys and ESP-8226 modules. The costs of the resistors and capacitors has been left off, as they cost fractions of a penny each.

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 software framework in development to support the robots is based on ROS, and so allows modular replacement of the control algorithms used to convert desired motion of the robot into drive signals for the motors.

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

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

Table 3: Current draw for Mabuchi-branded motors.

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.

5.4 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 are as shown in table 3.

These are somewhat large brushed motors. For smaller toys, coreless motors are more common. The values in table 4 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.

5.5 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, which is an infrared proximity sensor with

Motor number	No Load Current	Stall Current (measured)
Hexbug brand mini spider	0.03A	0.13A*
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 4: 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 motors by holding the toy still does not prevent the motor from turning

a 20mm 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.

References

[Abelson et al., 2000] Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight Jr, T. F., Nagpal, R., Rauch, E., Sussman, G. J., and Weiss, R. (2000). Amorphous computing. *Communications of the ACM*, 43(5):74–82.

[Arvin et al., 2009] Arvin, F., Samsudin, K., and Ramli, A. R. (2009). Development of a miniature robot for swarm robotic application. *sensors*, 1793:8163.

[Ashley-Rollman et al., 2007] Ashley-Rollman, M. P., Goldstein, S. C., Lee, P., Mowry, T. C., and Pillai, P. (2007). Meld: A declarative approach to programming ensembles. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2794–2800. IEEE.

[Ayanian et al., 2014] Ayanian, N., Spielberg, A., Arbesfeld, M., Strauss, J., and Rus, D. (2014). Controlling a team of robots with a single input. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1755–1762. IEEE.

[Bahgeçi and Sahin, 2005] Bahgeçi, E. and Sahin, E. (2005). 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.

[Beal and Bachrach, 2006] Beal, J. and Bachrach, J. (2006). Infrastructure for engineered emergence on sensor/actuator networks. *Intelligent Systems, IEEE*, 21(2):10–19.

[Belta et al., 2007] Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and Pappas, G. J. (2007). Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):61–70.

[Bergbreiter and Pister, 2003] Bergbreiter, S. and Pister, K. S. (2003). 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.

[Bonani et al., 2010] Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., and Mondada, F. (2010). 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.

[Bowyer, 2000] Bowyer, A. (2000). Automated construction using co-operating biomimetic robots. *University of Bath Department of Mechanical Engineering Technical Report (November 2000)*.

[Caprari et al., 1998] Caprari, G., Balmer, P., Piguet, R., and Siegwart, R. (1998). 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.

[Chen et al., 2015] Chen, J., Gauci, M., Li, W., Kolling, A., and Gros, R. (2015). Occlusion-based cooperative transport with a swarm of miniature mobile robots. *Robotics, IEEE Transactions on*, 31(2):307–321.

[Chen et al., 2011] Chen, J. Y., Barnes, M. J., and Harper-Sciarini, M. (2011). 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.

[Colot et al., 2004] Colot, A., Caprari, G., and Siegwart, R. (2004). 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.

[Coppin and Legras, 2012] Coppin, G. and Legras, F. (2012). Controlling swarms of unmanned vehicles through user-centered commands. In *2012 AAAI Fall Symposium Series*.

[Correll et al., 2009] Correll, N., Bachrach, J., Vickery, D., and Rus, D. (2009). 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.

[Cummings and Mitche, 2008] Cummings, M. L. and Mitche, P. J. (2008). 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.

[Daily et al., 2003] Daily, M., Cho, Y., Martin, K., and Payton, D. (2003). 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. IEEE Computer Society.

[Dantu et al., 2011] Dantu, K., Kate, B., Waterman, J., Bailis, P., and Welsh, M. (2011). Programming micro-aerial vehicle swarms with karma. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 121–134. ACM.

[Dorigo et al., 2013] Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., et al. (2013). Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *Robotics & Automation Magazine, IEEE*, 20(4):60–71.

[Dorigo et al., 2004] Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella, T. H., Baldassarre, G., Nolfi, S., Deneubourg, J.-L., Mondada, F., Floreano, D., et al. (2004). Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2-3):223–245.

[Evans, 2000] Evans, D. (2000). Programming the swarm. *University of Virginia*.

[Floyd et al., 2008] Floyd, S., Pawashe, C., and Sitti, M. (2008). 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.

[Gancet et al., 2010] Gancet, J., Motard, E., Naghsh, A., Roast, C., Arancon, M. M., and Marques, L. (2010). 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.

[Guo et al., 2007] Guo, Y., Hohil, M., and Desai, S. V. (2007). 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.

[Harriott et al., 2014] Harriott, C. E., Seiffert, A. E., Hayes, S. T., and Adams, J. A. (2014). Biologically-inspired human-swarm interaction metrics. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 58, pages 1471–1475. SAGE Publications.

[Hayes et al., 2001] Hayes, A. T., Martinoli, A., and Goodman, R. M. (2001). Swarm robotic odor localization. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 1073–1078. IEEE.

[Hayes et al., 2010] Hayes, S. T., Hooten, E. R., and Adams, J. A. (2010). Multi-touch interaction for tasking robots. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pages 97–98. IEEE Press.

[Jantz et al., 1997] Jantz, S., Doty, K., Bagnell, J., and Zapata, I. (1997). Kinetics of robotics: The development of universal metrics in robotic swarms. In *Florida Conference on Recent Advances in Robotics*. Citeseer.

[Jeanne, 1981] Jeanne, R. L. (1981). Alarm recruitment, attack behavior, and the role of the alarm pheromone in polybia occidentalis (hymenoptera: Vespidae). *Behavioral Ecology and Sociobiology*, 9(2):143–148.

[Kaber and Endsley, 1997] Kaber, D. B. and Endsley, M. R. (1997). 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.

[Kato et al., 2009] Kato, J., Sakamoto, D., Inami, M., and Igarashi, T. (2009). 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. ACM.

[Kernbach, 2011] Kernbach, S. (2011). Swarmrobot. org-open-hardware microrobotic project for large-scale artificial swarms. *arXiv preprint arXiv:1110.5762*.

[Laird et al., 2014] Laird, D., Price, J., and Raptis, I. A. (2014). Spider-bots: A low cost cooperative robotics platform.

[Lee and See, 2004] Lee, J. D. and See, K. A. (2004). Trust in automation: Designing for appropriate reliance. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 46(1):50–80.

[Levis et al., 2005] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al. (2005). Tinyos: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer.

[Lewis et al., 2006] Lewis, M., Polvichai, J., Sycara, K., and Scerri, P. (2006). Scaling-up human control for large uav teams. *Human factors of remotely operated vehicles*, 7:237–250.

[Mamei et al., 2003] Mamei, M., Zambonelli, F., and Leonardi, L. (2003). 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.

[Manning et al., 2015] Manning, M. D., Harriott, C. E., Hayes, S. T., Adams, J. A., and Seiffert, A. E. (2015). 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.

[Mason, 2003] Mason, Z. (2003). Programming with stigmergy: using swarms for construction. *Proceedings of Artificial Life*, 8:371–374.

[McLurkin et al., 2013] McLurkin, J., Lynch, A. J., Rixner, S., Barr, T. W., Chou, A., Foster, K., and Bilstein, S. (2013). A low-cost multi-robot system for research, teaching, and outreach. In *Distributed Autonomous Robotic Systems*, pages 597–609. Springer.

[McLurkin et al., 2006] McLurkin, J., Smith, J., Frankel, J., Sotkowitz, D., Blau, D., and Schmidt, B. (2006). 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.

[McLurkin, 2004] McLurkin, J. D. (2004). *Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots*. PhD thesis, Massachusetts Institute of Technology.

[Micire et al., 2009a] Micire, M., Desai, M., Courtemanche, A., Tsui, K. M., and Yanco, H. A. (2009a). 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. ACM.

[Micire et al., 2009b] Micire, M., Drury, J. L., Keyes, B., and Yanco, H. A. (2009b). Multi-touch interaction for robot control. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pages 425–428. ACM.

[Micire and Murphy, 2002] Micire, M. and Murphy, R. (2002). Analysis of the robotic-assisted search and rescue response to the world trade center disaster.

[Mottola et al., 2014] Mottola, L., Moretta, M., Whitehouse, K., and Ghezzi, C. (2014). Team-level programming of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 177–190. ACM.

[Nagpal, 2001] Nagpal, R. (2001). *Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics*. PhD thesis, Massachusetts Institute of Technology.

[Nagpal, 2004] Nagpal, R. (2004). A catalog of biologically-inspired primitives for engineering self-organization. In *Engineering Self-Organising Systems*, pages 53–62. Springer.

[Nagpal and Mamei, 2004] Nagpal, R. and Mamei, M. (2004). Engineering amorphous computing systems. In *Methodologies and Software Engineering for Agent Systems*, pages 303–320. Springer.

[Napp and Klavins, 2011] Napp, N. and Klavins, E. (2011). A compositional framework for programming stochastically interacting robots. *The International Journal of Robotics Research*, 30(6):713–729.

[Olsen and Goodrich, 2003] Olsen, D. R. and Goodrich, M. A. (2003). Metrics for evaluating human-robot interactions. In *Proceedings of PERMIS*, volume 2003, page 4.

[Olsen and Wood, 2004] Olsen, Jr., D. R. and Wood, S. B. (2004). 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. ACM.

[Olson, 2011] Olson, E. (2011). AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE.

[Olson et al., 2013] Olson, E., Strom, J., Goeddel, R., Morton, R., Ranganathan, P., and Richardson, A. (2013). Exploration and mapping with autonomous robot teams. *Communications of the ACM*, 56(3).

[Palmer et al., 2005a] Palmer, D. W., Kirschenbaum, M., and Seiter, L. (2005a). Emergence-oriented programming. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 2, pages 1441–1448. IEEE.

[Palmer et al., 2005b] Palmer, D. W., Kirschenbaum, M., Seiter, L. M., Shifflet, J., and Kovacina, P. (2005b). 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.

[Parasuraman et al., 2000] Parasuraman, R., Sheridan, T. B., and Wickens, C. D. (2000). 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.

[Parker, 1998] Parker, L. E. (1998). Alliance: An architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220–240.

[Payton et al., 2003] Payton, D., Estkowski, R., and Howard, M. (2003). Compound behaviors in pheromone robotics. *Robotics and Autonomous Systems*, 44(3):229–240.

[Payton et al., 2001] Payton, D. W., Daily, M. J., Hoff, B., Howard, M. D., and Lee, C. L. (2001). Pheromone robotics. In *Intelligent Systems and Smart Manufacturing*, pages 67–75. International Society for Optics and Photonics.

[Pelrine et al., 2012] Pelrine, R., Wong-Foy, A., McCoy, B., Holeman, D., Mahoney, R., Myers, G., Herson, J., and Low, T. (2012). 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.

[Pinciroli et al., 2015] Pinciroli, C., Lee-Brown, A., and Beltrame, G. (2015). Buzz: An extensible programming language for self-organizing heterogeneous robot swarms. *CoRR*, abs/1507.05946.

[Quinn, 2001a] Quinn, M. (2001a). 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.

[Quinn, 2001b] Quinn, M. (2001b). Evolving communication without dedicated communication channels. In *Advances in Artificial Life*, pages 357–366. Springer.

[Quinn et al., 2003] Quinn, M., Smith, L., Mayley, G., and Husbands, P. (2003). 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.

[Ray, 1991] Ray, T. S. (1991). An approach to the synthesis of life.

[Ricks et al., 2004] Ricks, B., Nielsen, C. W., Goodrich, M., et al. (2004). 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.

[Rubenstein et al., 2014] Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., and Nagpal, R. (2014). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975.

[Seyfried et al., 2005] Seyfried, J., Szymanski, M., Bender, N., Estana, R., Thiel, M., and Wörn, H. (2005). The i-swarm project: Intelligent small world autonomous robots for micro-manipulation. In *Swarm Robotics*, pages 70–83. Springer.

[Soule and Heckendorn, 2011] Soule, T. and Heckendorn, R. B. (2011). Cotsbots: computationally powerful, low-cost robots for computer science curriculums. *Journal of Computing Sciences in Colleges*, 27(1):180–187.

[Sumpter and Beekman, 2003] Sumpter, D. J. and Beekman, M. (2003). From nonlinearity to optimality: pheromone trail foraging by ants. *Animal behaviour*, 66(2):273–280.

[Tammet et al., 2008] Tammet, T., Vain, J., Puusepp, A., Reilent, E., and Kuusik, A. (2008). 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.

[Turing, 1952] Turing, A. M. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 237(641):37–72.

[Vicente, 2002] Vicente, K. J. (2002). Ecological interface design: Progress and challenges. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 44(1):62–78.

[Vicente and Rasmussen, 1992] Vicente, K. J. and Rasmussen, J. (1992). Ecological interface design: Theoretical foundations. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(4):589–606.

[Viroli et al., 2012] Viroli, M., Pianini, D., and Beal, J. (2012). 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.

[Wang et al., 2009] Wang, H., Lewis, M., Velagapudi, P., Scerri, P., and Sycara, K. (2009). 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. ACM.

[Wawerla et al., 2002] Wawerla, J., Sukhatme, G. S., and Mataric, M. J. (2002). Collective construction with multiple robots. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2696–2701. IEEE.

[Werfel et al., 2014] Werfel, J., Petersen, K., and Nagpal, R. (2014). Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758.

[Wiener and Curry, 1980] Wiener, E. L. and Curry, R. E. (1980). Flight-deck automation: Promises and problems. *Ergonomics*, 23(10):995–1011.

[Yanco et al., 2004] Yanco, H. A., Drury, J. L., and Scholtz, J. (2004). Beyond usability evaluation: Analysis of human-robot interaction at a major robotics competition. *Human–Computer Interaction*, 19(1-2):117–149.