ABSTRACT

AAROM: AN AUTONOMOUS SYSTEM BASED ON SWARM INTELLIGENCE AND ANT FORAGING TECHNIQUES

by

Kristin Eicher-Elmore

Master of Science in Computer Science

California State University, Chico

Spring 2008

Recently, researchers have placed much emphasis on developing simple robotic systems that can operate collectively in an intelligent way. The benefits of such a system are obvious and important to certain applications. A swarm intelligence system is by definition made up of several relatively unsophisticated agents that work together to create an intelligent solution. Being relatively unsophisticated means that these agents are inexpensive. This means many can be deployed into a hazardous and unpredictable environment. This paper describes a swarm intelligence system referred to as AAROM, or the Artificial Ant Route Optimization Model, that is designed to search for the shortest path to a goal and relay the location of a target to the other robots. The shortest path is discovered by optimization techniques that develop at runtime. AAROM is primarily meant as a prototype model for a search and rescue application.

CHAPTER 1

INTRODUCTION

Background

Swarm intelligence has been described as a colony of simple agents that work cooperatively to solve apparently difficult tasks [1]. When agents act in this way, and this solution appears dynamically it is called *emergent behavior*. When more than one agent acts together to find this solution it is called *collective behavior*. These two concepts are indispensable to swarm intelligence robotics systems. The idea behind swarm intelligence is to have a group of agents that have the ability to cooperate and to generate new behaviors due to reactions to their environment. Two researchers are considered as doing the seminal work in swarm intelligence. These computer scientists are Eric Bonabeau, and Morico Dorigo. There research will be discussed further in chapter two and chapter three.

There are many benefits to using multiple simple agents that are exciting the robotics research community at this time [2], [3], [4], [5], [6], [7], [8]. This period in robotics research is a very rich one where the development of swarm intelligence robotics systems can be very rewarding in many different applications. According to Steele et al., “Multi-robot systems are particularly useful in tasks that require searching large areas such as planetary science exploration, urban search and rescue, or landmine remediation [9].” There are other uses as well, such as, “... telephone switching, network routing, data

categorizing, and shortest-path optimizations[10].” The applications use certain features of biology to expand and enhance the algorithms employed. But, in all cases the idea of using more than one agent to find an optimal solution is standard. In fact, the more agents used the better the solution. Even though each application may not model biology exactly, the fact remains that in order to utilize the beauty and simplicity of a true swarm, all research in the field originates from a knowledge of biology first. A very useful swarm to study is an ant swarm because of its ability to find the shortest path to a goal. This project, named the Artificial Ant Route Optimization Model (AAROM), involves a swarm that has been modeled on ant foraging techniques.

Stygmergy

In the case of scouting or foraging for food in the ant world, scouts find the shortest path to a food source using stigmergy. Stigmergy is in essence merely the indirect stimuli that is sometimes known as pheromone communication, or the use of scent droppings to find a food source [3], [2]. Thus, AAROM works on the concept of stigmergy or the indirect clues left by a scout. Also, AAROM is an experiment in finding the shortest path to a goal. In the ant world, the first ant to reach the food and return to the nest will have found the shortest path. This ant leaves a scent trail from the food to the nest on returning for other ants to follow. Another ant follows this trail since it is fresh and it will have the most attractive scent. As this ant follows the trail, it will leave more pheromones and more ants take this route. Since the original ant found the food first, then it is more likely that it found the shortest path since the other ants did not return as fast as the first scout. This is a fine example of simple agents like ants using something as

seemingly innocuous as a dropping of scent to solve a problem that may be complex such as finding the shortest path to a goal [12], [2], [10].

Purpose

AAROM is an illustration of what a system based upon ant foraging can do in the field of search and rescue. The ability of a system to find the shortest path to a goal could be used to find a victim and contact other agents. Path details could be then sent to other agents enabling them to swarm to the scene bringing help such as water, food and oxygen or other forms of aid. A system such as this is valuable because of its ability to not only communicate successfully with other agents using indirect clues, but it also employs an algorithm that successfully finds and stores directions to a goal in a highly efficient and successful way. The way in which the project goes about achieving these goals is in compliance with one of the main goals of swarm intelligence. Basically this goal is to keep things simple.

Pheromone Communication

Using indirect clues works very well towards keeping AAROM simple. Although AAROM uses wireless radio frequency (RF) communication to create a “virtual” pheromone trail, the source of this communication can still be considered indirect and simple. This is because the project uses an indirect planningtechnique utilizing the communication of an action sequence route used to find the goal, instead of an actual plan or complete path. This can be thought of as the robot sensing its way through hints like an ant in nature would use the sense of chemical information. This is opposed to the idea of direct communication that in the natural world would be a verbal

or a visual direction to the goal. The use of these hints is different than it would be to communicate an actual step by step direction that would appear more direct in its literal definition of an exact path or trail. The robot, like an ant, interprets the hints of how it should act in its directions, instead of moving in the exact same way as the scout did when it left the nest. The benefits of using these kinds of indirect clues become apparent when one considers using a robot that doesn't require a lot of memory to store a map or a complete path to the source. Keeping this element simple means more memory for other abilities and thus increases the cost-effectiveness of a system.

Search Algorithm

The search algorithm itself that discovers the optimal (shortest) route results in another kind of benefit, efficiency. The search algorithm is designed in a way that discourages any robot from doubling back and failing to make progress in a particular quadrant. It establishes a boundary on the amount of steps are made in a particular area of a block before it moves on to another one. In this way, the robot makes efficient forward progress to a goal. The search environment for this project is an artificially controlled 3-D maze, which will be described in detail in chapter three. However, steps could be made to adapt this process to an actual terrain, using internally represented search grid. The project stands as an illustration of how a group of robots can be sent out to find a target and communicate with a swarm to come out with aid for the victim. This model shows how simple agents can be used inexpensively to produce one of the primary goals of swarm intelligence, which is emergent behavior. So one might think, what is another

benefit of simplicity? One of the best factors for a system that might be working in tough conditions is cost-effectiveness.

Cost Effectiveness of Hardware

"Inexpensive" is the key word when one refers to the benefits of Swarm intelligence methods. The cost-effectiveness of hardware is so important in multi-agent systems that this is one of the biggest reasons it is being considered now in the search and rescue, surveillance and land mine remediation where robot losses could be great [13], [8]. Robots used in swarm intelligence systems are kept simple so their hardware is inexpensive. Not only can robots be easily broken or disabled in a search and rescue venture, but also a good system requires a lot of agents to be effective in large areas. Thus, robots can be replaced or fixed when the cost of the hardware, such as microcontrollers, sensors and memory are low. Also, this thrift allows for the easy creation of huge swarms for large and treacherous areas. It is these large areas that need a great number of scouts and aid robots to make up for the size of the area and the inevitability of robots getting stuck or broken. AAROM only requires five robots because of the small and uncomplicated choice of terrain and the necessity for limiting the scope and cost of the project. Also, techniques are used to keep the scouts and other robots from getting stuck. Although the project is meant as a model for a bigger autonomous system operating in real terrain, it still successfully demonstrates the ideals of simplicity and cost-effectiveness. This lies mainly in the fact that the robots and the sensors being used are simple and inexpensive.

Cost Effectiveness of Software

Cost-effectiveness of the hardware is one aspect of being inexpensive, however, this flows into other considerations as well. The simplicity of AAROM creates a cost-effectiveness related to time rather than just money and memory space. A system using precise and simple algorithms and simple forms of communication that is timely will create a robot that is able to compute and react quickly. This kind of cost-effectiveness is extremely important when that goal needs to be found quickly as possible for the fastest kind of help. The routing algorithm used by AAROM keeps communication simple and cost-effective because of an efficient use of memory. Having a plan of every movement made on the way to the goal is less cost effective then the method used by AAROM because it uses more memory and takes more time to access the route data. Also, communication time itself is kept at a minimum. Smaller packets of information mean quicker communication meaning optimized response time. Each of these aspects of the routing and communication algorithms used by AAROM demonstrate the simplicity and cost-effectiveness characteristic of swarm intelligence ideals by keeping the use of memory space and time at a minimum.

Search and Rescue

Creating a model for a possible swarm intelligence search and rescue system became the motivation for this project while investigating ant algorithms and contemplating the benefits such a system could provide [4], [8], [14]. Police and fire departments, and the military, are examples of organizations that would find a system that could employ a quick, efficient and adaptable search technique using several agents

very useful. Victims of natural disasters, victims of war and fire, and crime victims or other injured people need to be found quickly and efficiently. From a humane and economic standpoint, it makes more sense to deploy a large group of small and unobtrusive robots to do the search and rescue tasks in a hazardous environment then it does to send limited large-scale expensive and obtrusive equipment and/or fragile human lives.

Small, inexpensive, yet durable robots would be very useful for search and rescue and should not impede other rescuers at the scene. Also a system such as AAROM has inherent value since it could be deployed and left alone. This is because it is a completely autonomous system, meaning that it can work in a dynamic real-time environment without the direction of a human operator. This would mean that a handler might be able to take other swarms to different areas to create the best search sweep possible. The next chapter will present some of the research and uses of swarm intelligence robotic systems being studied in the current decade.

Glossary of Terms

- Collective Behavior: When agents work together to solve a problem using the same algorithm [15], [16], [17], [1], [2], [8].
- DCF Protocol: It is an acronym that stands for distributed coordination function. A mandatory protocol that makes up one of the two forms of medium access that makes up the MAC Layer. It requires that stations that are attempting to send frames must wait until there is no other station transmitting [18].

- Emergent Phenomena: The process where a community of agents dynamically develop behaviors as they work collectively to solve a task [15], [16], [17], [1], [2], [7].
- Mass Recruitment: The process where ants are directed towards a food source by the use of pheromone trails [16], [3], [1], [2].
- NAV: It is an acronym that stands for network allocation vector. This is a measurement kept at each station that represents the amount of time the previous frame that was sent needs to send its frame. The NAV must be equal to zero for a station to send a frame [18].
- PCF protocol: It is an acronym that stands for point coordination function. An optional protocol that makes up another part of the MAC Layer besides the mandatory DCF. This protocol controls communications by requiring the access point to poll the station when it is free to receive a frame [18].
- Pheromones: The chemical scent used by ants to communicate with each other [19], [16], [3], [1], [2], [8].
- Reactive Behavior: The reaction of an agent towards an outside stimulus, such as light [15], [20], [1], [2].
- Stigmergy: Indirect communication as opposed to a visual or auditory one between two agents. For example, ants use scent to communicate with each other [19], [16], [3], [1], [2], [8].
- WLAN: It is an acronym that stands for wireless local-area network. It is a type of network that uses high- frequency radio waves rather than wires to communicate between network nodes [18].

CHAPTER II

LITERATURE REVIEW

A Live Swarm: Ants

Dorigo et. al. experimented with live ants to observe how they scouted using stigmergy or indirect communication. The use of pheromone to mark a trail in the process of scouting is called mass recruitment [2]. The experiment involves a bridge from a nest of Linepithema humile ants to a food source. The bridge has two branches that converge from the main path and come back to it a later point close to the food source. One path is long, and the other is shorter. The experiment itself did not take pheromone evaporation or decay into account because the procedure only lasted for about eight minutes on the longer path and four minutes on the shorter path which is not enough time for any noticeable pheromone evaporation [2].

Mass Recruitment

After some brief initial fluctuations, the shorter bridge was most often chosen. This happens because the first ants coming back to the nest have already used the shorter bridge more than once, leaving the nest and returning after finding the food source. Therefore, immediately after they have returned there is already more pheromone on the branch that they had taken. The ants left at the nest will then follow the trail that brought the first ants home. Therefore, according to a very simple process called mass recruitment, the process of following chemical trails to find a food source, results in

calculating a solution to a potentially complex problem like finding the shortest path to a goal [2]. It is through this collective process of the first ants to find the food that the rest of the colony benefits with a short path to a food source.

Pheromone Characteristics

As the experiments with real ants have shown [2], sometimes a few ants will choose the longer trail and drop enough pheromone to cause other ants to become "trapped" into a sub-optimal path. This is unless an individual strays from the path and finds another route, which isn't the norm but can happen. Also, pheromone scents, depending upon species and conditions, can last for hours or for months. An ant colony might work on the same site using the same trail for quite some time. But, in the ant world this works when considering that switching a popular trail to enhance optimality may cause other problems for the colony, such as defense. Once the food is depleted, the ants will respond by ceasing to release any more pheromone as they return to the nest. This results in the path's attractiveness dwindling as the pheromone scent fades. The ants will then stop using it altogether. This describes how a real ant colony forages for food [12], [2].

AAROM described in this paper mimics such a search for a short path. In AAROM , two agents or "scouts" find this best path and send the pheromone value that communicates the path information to the other ants in the group.

An Autonomous Swarm Application

Work in the field of swarm intelligence is ongoing and new. Spanning many different aspects of swarm intelligence research is deemed important and powerful.

"Swarm intelligence provides us with a powerful new paradigm for building fully distributed de-centralised systems in which overall system functionality emerges from the interaction of individual agents with each other and with their environment [4]." The importance of this research is coupled with the fact that the environment in which the agents work is highly unpredictable and even dangerous. In these unstable situations, emergent behaviors are essential to reacting in a dynamic environment. This is the case in a dangerous and unpredictable situation like agents face when they are deployed to find a chemical spill. Bruemmer et. al has created an experimental system in which they utilized the emergent behavior idea [17]. This idea involves a close relationship between environmental changes and reaction.

Very little processing occurs when a cockroach senses a subtle change in lighting and it moves. Likewise, our robots do not make deliberative, high-level decisions about the task, but react to their environment using fast, responsive behaviors that are domain independent and robust because they do not rely on sophisticated internal processing. We believe that our approach is especially useful for rapid response type missions where little is known about the environment and/or there is insufficient time to custom tailor a robotic solution [17]. Reactive behavior that is created to respond quickly is very important in hazardous conditions.

The Use of Human Operators and Autonomy

One of the challenges Bruemmer's research team faced was finding a balance between having a human operator and autonomy. They have found a medium ground with their system AgentCDR [17]. This system has a parent robot, sergeant robots, and private robots. The parent is led by remote control into a target building by a human

operator. The parent robot's purpose is merely to transport the swarm into the target building. From there the sergeant robots are commanded by RF communication from a human operator. The sergeants are specialized for communication. Their role is to tell the privates what to do. Although a human operator communicates to the sergeant what the private's duties should be, the control used is not completely automated. In this system the control is still influenced by the private robots' emergent behavior resulting from its dynamic responses to light and sound. The agent is given an internal critic that is adjusting the agent's sensitivity factor to light and sound. Some randomness is added to the level of sensitivity to the environment so that emergent behavior can develop in a dynamic way. The agent's can therefore react to their environment in a proper way while cleaning up chemical spills.

Partial Autonomy vs. Autonomy

Experiments showed that the AgentCDR system was successful with the properly trained human operator. Finding and cleaning spills is a hazardous and delicate job. Thus, it requires the help of a human operator to aid in the clean up. Breummer's research is a good example of a level of autonomy with swarm intelligence and human control that, when working together, can find and deploy dangerous spills. However, in other applications requiring a swarm, a high level of human intervention may not be desirable. This is true especially in the case of surveillance or even to a greater extent, search and rescue. Regarding search and rescue systems, it would be desirable for a handler to place a team in an area and be able to leave the agents there to do their job so that another team can be deployed. In these applications, a more autonomous swarm intelligence system like the one presented in AAROM is more desirable.

A System that Models Three Ant Behaviors

A different kind of control layering is seen with De Wolf et al.'s swarm intelligence system [3]. With their system, one of three behaviors is scouting for food. Two sub-behaviors would be “searching for food” and “laying the trail to the nest.” The sub-behaviors will change when the environment makes it impossible to perform the method. Sub-behaviors are chosen according to a threshold value. Some of these behaviors will have lower thresholds and these are chosen more. The threshold values all change dynamically while a behavior is being performed. The behaviors are also chosen from a set of different actions according to the stimulus values of each agent. Once the stimulus reaches a certain level then an action is performed. The dynamic quality of the reactions of these agents is based upon the changing threshold values. Learning occurs when an agent chooses a behavior based on the success it had with performing the action previously. This research team concentrated on ant behavior for the benefit of different applications and had very realistic results [3].

The Three Behaviors

The dynamic task allocation model of De Wolf et al. was based upon three major behaviors and four sub-behaviors. This system was a successful attempt at modeling ant behavior. The goal of the De Wolf model was to mimic three behaviors: scouting, nursing and foraging. Foraging is the only behavior described in AAROM research. Therefore the scope of De Wolf's research was much broader than AAROM in terms of the amount of possible ant optimization techniques since it included not only foraging, but nursing as well. The focus of AAROM was not to primarily model ant

behavior in general but to demonstrate one way for a swarm to find and swarm to a target. However, the idea of creating a dynamic program where behavior is emergent is basically the same goal.

Investigating Applications using Blanket and Sweep Coverage

The military [20] has been looking at swarms very intensely in its plan to use it for a wide variety of applications involving dangerous environments. The use of large numbers of small robots (with varying degrees of 'large' and 'small') has been seriously proposed for a wide variety of applications, including intelligent land mine deployment, behind-the-lines military communications relaying, warehouse security sentry, ship hull cleaning, warehouse material handling, lunar base construction, gathering oceanographic data, planetary surface exploration, and aircraft carrier deck foreign object debris (FOD) disposal [20].

The objective of the military research is to use swarm intelligence to achieve blanket, barrier, and sweep coverage. Blanket coverage involves the static placement of detection agents that will maximize the discovery of targets in the coverage area [20]. Barrier coverage is also a static arrangement but seeks to prevent penetration of a target into the coverage area [20]. And sweep coverage is the movement of agents whose goal is to keep the target out of the coverage zone [20]. The challenge in the military's research is to program the agents to stay a certain distance from each other and maintain a tight formation while avoiding obstacles. These goals are achieved through sensor information and communication for each robot to understand the position and movements of the others.

When the goal of a robotic swarm system is to cover a large area in a horizontal fashion, the objectives are somewhat different. As is the case with this military research, the positioning of the agents becomes primary. The positioning is done in a highly horizontal fashion with each agent a certain distance apart maintained through sensor information. AAROM does not aim to concentrate on methods of sweep coverage. However, AAROM research is based upon emergent behavior and the stigmergent communication that characterizes the research of swarm intelligence in general.

A Swarm Intelligence Solution using Pheromone Information

Another use of swarm intelligence can be found in a new system developed by Asama et al. [21]. This system, termed the Intelligent Data Carrier or IDC, is used for corporate transportation and uses pheromone information to find the optimal path to the agent's destination. If this path is inaccessible, the agent will take an alternate one and store the information it discovers about this route on memory nodes mounted on the halls of the building using RF wireless communication. If a robot encounters a situation where the desired path chosen by pheromone information is unavailable then it will access the memory of the nearest node to it to find the next most efficient path.

Mapped vs. Unknown Environments

Corporate transport robots have a very static environment where structures can be mapped and planned for in a multi-agent system. Research in other domains does not have the luxury of this kind of certainty in the features of the environment it will encounter. Therefore, the use of memory nodes to store path information will work in known locations. However, in uncertain environments the use of this amount of memory

will not be available. The storage of information on known environments and features will not be efficient or even possible for simple, small swarm agents. In systems like the one presented in this paper, only pheromone information is used as the indicator of which path an agent should take. Similarities between the IDC, or Intelligent Data Carrier, and AAROM do exist, however. In both the IDC and AAROM path directions represent pheromone trails and are stored in a memory module and later communicated to other agents. Also, in both systems the pheromone values do not decay.

Wireless Communications

Due to the benefits of autonomous agents with the ability to cooperate and act autonomously, wireless communications become an important issue [22]. It is especially important to make sure that these communications are accurate. It is through accuracy that quick communications can be maintained for fast reaction times. Also, because of the unpredictable terrains that a swarm might be introduced to, the communications between agents might be obstructed by walls or environmental static. It is therefore necessary to ensure accurate communications with certain safeguards and protocols.

A Communications System for Areas with Obstructions

One of the most difficult challenges in multi-agent communication is maintaining wireless RF signals. Distance, obstructions, and noise all contribute to a difficult situation. One solution proposed by Nguyen et al. [22], is a caravan of robotic RF links that will connect a human controller with the main robot. Moving links enable a flexibility that dropping communication bricks (another solution) fail to achieve. The communication link monitors the radio signal of the agent behind it. When the signal

strength drops below a certain level, the agent stops and becomes a stationary radio link. The result is like a trail of bread crumbs that drop themselves in a trail.

Another level of functionality involves the node sensing that the communication agent is no longer needed in its station because of strong signals behind and in front of it in line. The communication link agent will then move to be delegated to another position where it can be more useful. This method is very valuable in environments where thick concrete walls such as bunkers make communication difficult and when high bandwidth transmissions and frequency necessary for video feeds are required [22]. Lower frequency transmissions are actually more successful at penetrating walls, but when video feeds are required this is not an option.

The dynamic nature of Nguyen et al.'s communication scheme [22] makes it a very attractive solution for the military. It becomes almost necessary in situations where high bandwidth communication is required. However, the simple and uncomplicated communication scheme of the wireless RF communication in AAROM makes this kind of complication unnecessary. Indirect communication relies on only very simple cues so low frequency RF communication is all that is required. The transceivers used in AAROM can penetrate walls up to a distance of 200 feet because of its low frequency nature. Swarm systems based upon simplistic communication needs are efficient and practical since the ability to maintain communication is much less complicated than high bandwidth and high frequency communication.

The Mac Layer of WAN Communications

When dealing with any kind of network communication, protocol becomes an important factor in making data flow seamlessly and effectively. The main operating

control that coordinates transmissions and receipt of wireless communications between the radio network cards of a wireless computer and its access point is called the MAC layer. MAC layer is an acronym for medium access control. Coordination between these two radio mediums is essential since a radio channel is shared between each of them. The physical layer used by the MAC layer are the standards such as 802.11b or 802.11a that perform the duties of carrier sensing, transmission, and receiving of 802.11 frames.

The DCF and PCF Protocols

Before transmission of any 802.11 frame can occur, a station, which might be a radio network card or an access point, must gain access to the medium or the radio channel that all must share. The 802.11 standard has two different protocols that it uses: the distributed coordination functions (DCF) and the point coordination function (PCF) [18]. DCF involves competing stations that send frames when there are no other stations transmitting. If a station is already using the channel, then a competing station will wait until the channel is free. Before accessing a medium the MAC Layer will check the value of its NAV or network allocation vector. The NAV is a counter value representing the time the previous frame requires for transmission. The value of the NAV must be zero before the next frame can be sent. Before this next frame is transmitted, the time to send this frame is calculated and used to set its NAV. When this is complete, the channel is reserved for this station.

One of the most important components of the DCF is a random back-off timer that a station uses if it discovers that the channel is busy. If the channel is being used then the medium will wait for a random period of time before it attempts to use the channel

again. This process ensures that several mediums will not try to transmit at the same time and jam and confuse transmissions on the radio channel.

With radio LANs, a station cannot listen for collisions while sending transmissions because they are unable to receive transmissions while it transmits. Therefore, an acknowledgment is required by receiving stations to communicate to a transmitter if no errors in the packet were detected. If the sending station does not receive a positive acknowledgment after a specific amount of time, it will assume that a collision or RF noise was encountered and it will send the frame again. Thus, DCF is a mandatory protocol for wireless communications [18].

PCF is an optional protocol. According to this protocol, the access points will poll stations one by one from a table during a period when there are no competing mediums waiting for the channel (contention free period). Stations cannot transmit data unless the access point polls them first to see if there are frames to be sent. PCF is intended to be used by the 802.11 standard alternately with DCF during contention free periods if DCF is enabled. Currently, this protocol is not being used by wireless NICs or access points [18].

A Protocol for AAROM

Since the radio communication used by AAROM is similar to wireless local-area network communication (WLAN), in that only one channel is available for the communication of agents, a protocol was designed for AAROM that has some similarities to the MAC Layer scheme. This protocol was ultimately removed from the system, but since this protocol is mentioned later in chapter four it will still be described here. The protocol is based loosely on the DCF and PCF protocols. Agents in AAROM

that wish to transmit some data listen before transmitting a packet to see if the channel is being used. It will not transmit if it receives any data and waits a random amount of time before listening again. However, if it finds that the channel is free, it will transmit its packet. In this way, no two agents will transmit at the same time. When data is being sent, the sender waits for a standard amount of time for an acknowledgment from the receiver that the data was received error free. If the acknowledgment is not received in this time, then the data is sent again just as with the DCF protocol. AAROM's protocol differs from the DCF protocol in that it does not use a NAV value to determine when to send the next packet. Instead, it listens for any transmissions before sending to the receiving agent. The protocol for AAROM is different than the PCF in that the agent will not transmit unless it does not hear any transmissions on the channel, instead of waiting for a poll value to confirm transmission.

The MAC Layer protocol is successful at managing the WLAN that uses the 802.11 standard. However, in AAROM the length of a packet is always the same. Thus, a NAV is unnecessary since the time required for a transmission will always be constant. This fact makes the protocol originally developed for this multi-agent system a lot less complex than the WLAN using the 802.11 standard. What it means for AAROM is less cost in time, memory use, and programming complexity. The command that controlled the receiving of transmissions sent the flow of the program to a subroutine that indicated that no data was received after a standard amount of time (constant time that it takes for a packet to be sent). This feature enabled the system to determine when no data was being sent at a particular moment of time and would allow transmitting agents to know when the channel was free, making it easy to determine proper transmission timing. Although

this method was very successful in keeping the communications between robots from jamming, some necessary changes in the hardware made it impossible to use. This change will be discussed further in chapter four.

Summary

This completes the discussion of related swarm intelligence research being done in the field of artificial intelligence presently. Although each system varies, from AAROM they share the main characteristics of swarm intelligence. The use of a great number of simple yet competent robots is the norm for each. The next chapter discusses how ants find food. The simple algorithm that is the foundation for this process and how AAROM uses its ideas is examined. Additionally, the way in which emergent behaviors in the ant world develop is compared and contrasted to the agents in AAROM . Finally, an overview of the platform and hardware used to create AAROM is illustrated in chapter three.

CHAPTER III

METHODOLOGY

AAROM Environment, Platform and Hardware

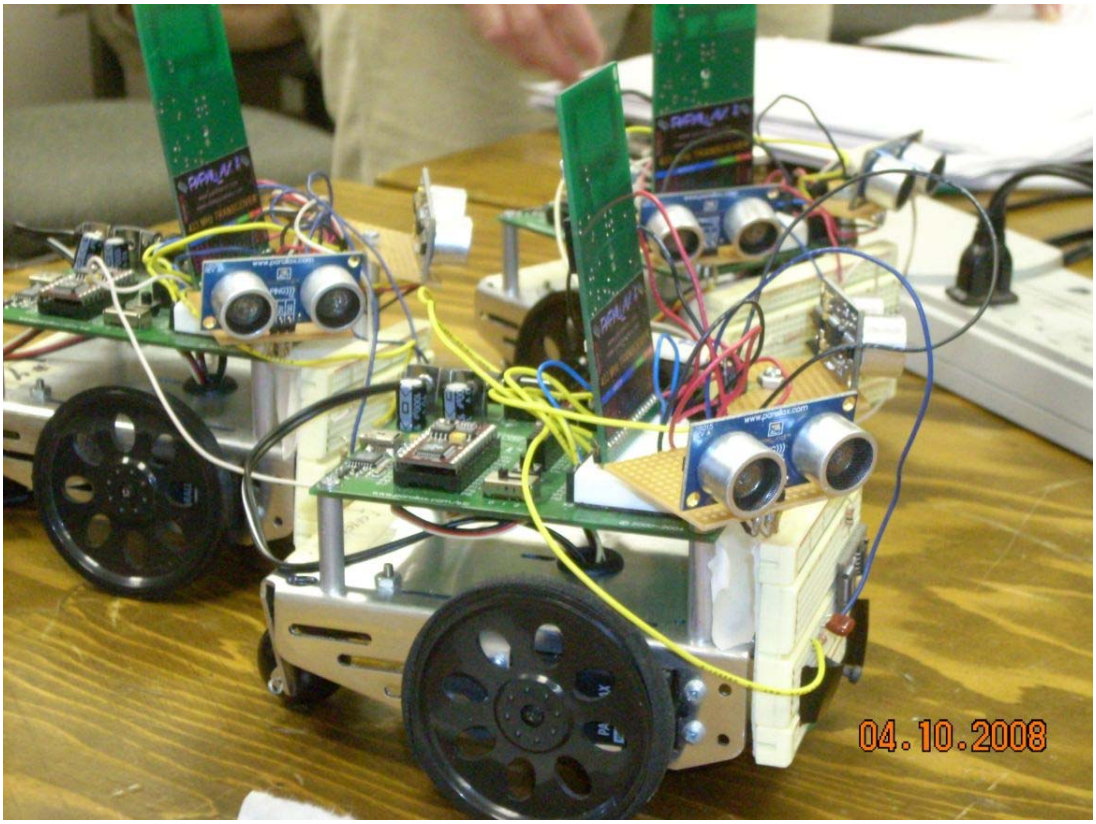

Fig. 1. AAROM agents.

AAROM Environment

The environment for AAROM, a maze, does have some specifics. But, beyond these requirements the walls can be put in any arrangement. One of the most

important specifications is that the walls must be at least six inches high. This is so the ultra-sonic ping sensors will be able to sense them as an obstacle and will not see over them leaving the agent stuck against an obstruction. The height of an agent in AAROM is about five inches tall with the sensors mounted on top. The ground surface of AAROM environment is marked with colored zones. These zones help the agents make forward progress in the maze. They can be arranged in any way, but must be at least approximately seven inches long and the width of the corridors of the maze. The last requirement is the width of the corridors of the walls. The corridors should be a maximum of approximately two feet wide. This specification comes from an idiosyncrasy of the algorithm that results in an AAROM agent becoming confused and perhaps doubling back because it loses track of the fact that it has been in this particular quadrant before. This problem will be explained in further detail in chapter four.

Microcontroller

The robots used in AAROM are Parallax™[1] Boe-Bots®[2]. They use a BS2pe microcontroller. These chips have an instruction speed of 6000 instructions per second. They also have a memory that holds 32K of memory total, 16K which is reserved for the program only. The rest of this memory space can be written to and read from for algorithm uses. The microcontrollers were purchased primarily for their relatively large memory capacity. All of the solution was written in a language called PBasic®[3]. The editor used is software created specially for the Parallax™ Basic Stamp®[4] chips [23].

[1] Parallax is a trademark of Parallax, Inc.
[2] Boe-bot is a registered trademark of Parallax, Inc.
[3] Pbasic is a registered trademark of Parallax, Inc.
[4] Basic Stamp is a registered trademark of Parallax, Inc.

Transceiver

The hardware used for communication in AAROM is the Parallax™ 433 Mhz Transceiver. The module is able to communicate in switch or serial mode depending upon the way it is wired to the robot [24]. In AAROM , it is hooked up for serial communication. Figure 2 illustrates the way in which the transceiver is hooked up.

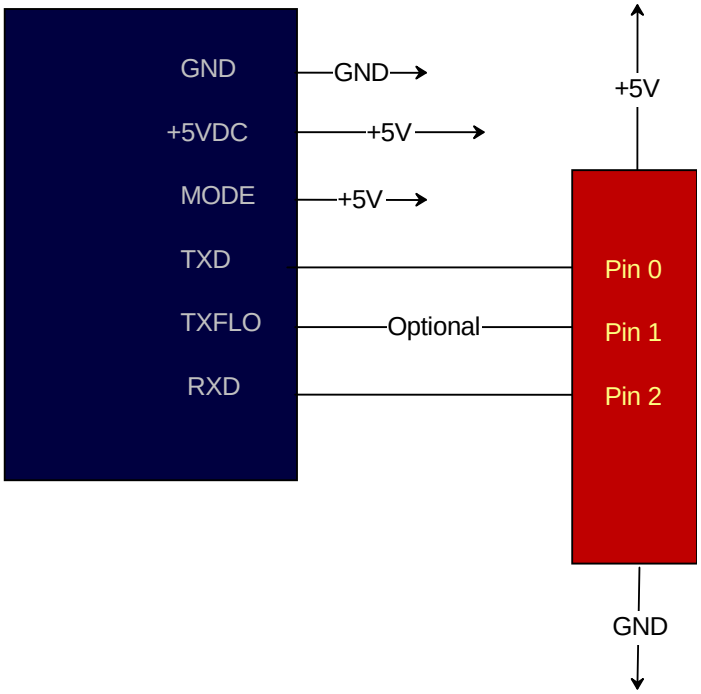

Fig. 2. Transceiver hooked up to microcontroller.

As shown in Fig. 2, it is hooked up to power and ground. The mode pin is hooked up to power, or five volts to indicate that it will be operating in serial rather than switch mode. The next pin, TXD indicates the pin in which outgoing data will flow or the transmit pin. The serial communication will be outgoing at 9600 baud, and operating at +5 volt and zero volt logic operation levels. Sending data involves using standard serial

protocol, at 9600 baud with eight data bits, no parity and one stop bit. The next pin ensures the maximum efficient throughput of data and is called the TXFLO pin. The RXD pin takes care of any 9600 baud incoming data coming from another transceiver. It also uses standard serial protocol [24].

The Basic Stamp® microcontroller mounted on the robots is able to perform serial byte pacing and flow control handshaking in one instruction. However, additional precautions must be made in the solution to allow for noise and obstructions. The use of a checksum procedure is necessary and used in AAROM to ensure that the data that is sent by a transmitting robot is the same as that of the receiving one.

Ultrasonic Ping Sensors

The Boe-Bots® are also equipped with ultrasonic ping sensors. These sensors provide accurate measurements from about two cm to two meters. They work by transmitting a 40 Khz ultrasonic burst. They then sense when the pulse returns after it bounces off an object. The host or the basic stamp is alerted when the ultrasonic burst is sent, then it is contacted again when the burst returns. The width of the pulse defined by the time the burst left until it returns is calculated by the Basic Stamp® to find the distance to the object detected [25].

The Boe-Bots® have two of these ultrasonic sensors. When they are placed on the front of the Boe-Bot® facing straightforward, it became obvious that they saw straightforward and not to the sides at all. Therefore, the sides of the Boe-Bot® were left unprotected by any sight protection, leaving the possibility of the wheel or side of the bot to get stuck on obstacles (see Fig. 3).

Fig. 3. Field of vision for ultra-sonic ping sensors facing forward.

By turning the sensors out slightly, they can still see straight and they give more protection to the sides of the Boe-Bot® as well (see Fig. 4).

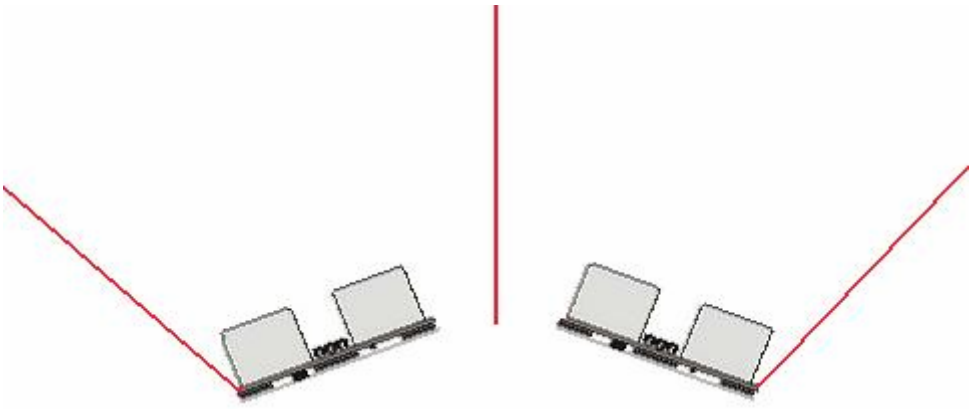

Fig. 4. Field of vision for ultra-sonic ping sensors facing outward.

Photo-Resistors

There is another form of vision on the Boe-Bots®. These are called photo-resistors and they work by sensing the amount of light that reaches their surfaces. They can therefore be used to sense the amount of light that is being reflected by a surface.

This measurement can be used to determine if this surface is light or dark. Photo-resistors can also be used to sense the direction of a light source such as a light source.

The photo-resistors have a surface that is covered with cadmium sulfide. When light hits this surface an electrical resistance is built up that is inversely proportional to the level of light in the environment. For example, darkness causes more resistance and light causes less resistance. The circuit of the photo-resistor includes a capacitor. This capacitor is charged by a five volt burst from the pin's high signal. The I/O pin then will sense the decay of the voltage as the photo-resistor resists the electrical charge coming from the capacitor. The amount of time is calculated by the basic stamp that it takes for the voltage that the I/O senses to decay below 1.4 volts. If the photo-resistor has a strong resistance due to low lighting conditions, then the capacitor takes longer to discharge. If the lighting conditions are high, then the capacitor loses its charge very rapidly. It is in this way that the lighting conditions sensed by the photo-resistor is computed [26].

There are two photo-resistors on the Boe-Bots®. One is mounted on the front, and the other is mounted on the bottom. The one mounted on the front of the bot is used to sense a light source (the food source). The photo-resistor on the bottom is used to determine the floor color. In order for the photo-resistor on the bottom to work efficiently, an LED is added. The extra light makes up for the darkness of the photo-resistor being mounted on the bottom of the bot. This makes it possible to tell if the Boe-Bot® is on a white block or a black block in the maze the ants traverse [26].

Circuit Diagram

Figure 5 shows a circuit diagram of the hardware that makes up AAROM . Each pin number represents the I/O pins that supply the power to the hardware in the form of a five volt pulse, or that wait passively for input from the hardware. The two circuits at the top represent the photo-resistor circuits. The second photo-resistor from the top has an LED attached to the circuit (see Fig. 5).

Next, a description of how the method in AAROM uses the ant algorithm of finding the shortest path.

Project Algorithm

The main algorithm operating AAROM has been based on techniques used by ants to solve the shortest path problem. As mentioned in chapter two, this algorithm is based on the natural foraging behavior of ants. The search method used by AAROM can be defined as a depth first search using a set of action sequences that describe a route to the goal. It is this dynamic action set that is sent as a virtual pheromone guide to the other ant agents. In the course of finding food, the first scout to find the food is most likely to have found a route to the food source that could be described as a local optima to the shortest path problem. If one were to merely expect an agent to venture forth into an environment full of obstacles and find a goal quickly, then that would be a bad assumption. One could merely let the robot move around aimlessly until it found the food, but then this would really take too long. If a solution is going to truly find a quick route to the food then it must be reasonable. What is ideal, of course, is to try and force the agent to progress towards a solution as much as possible so the food is found in a

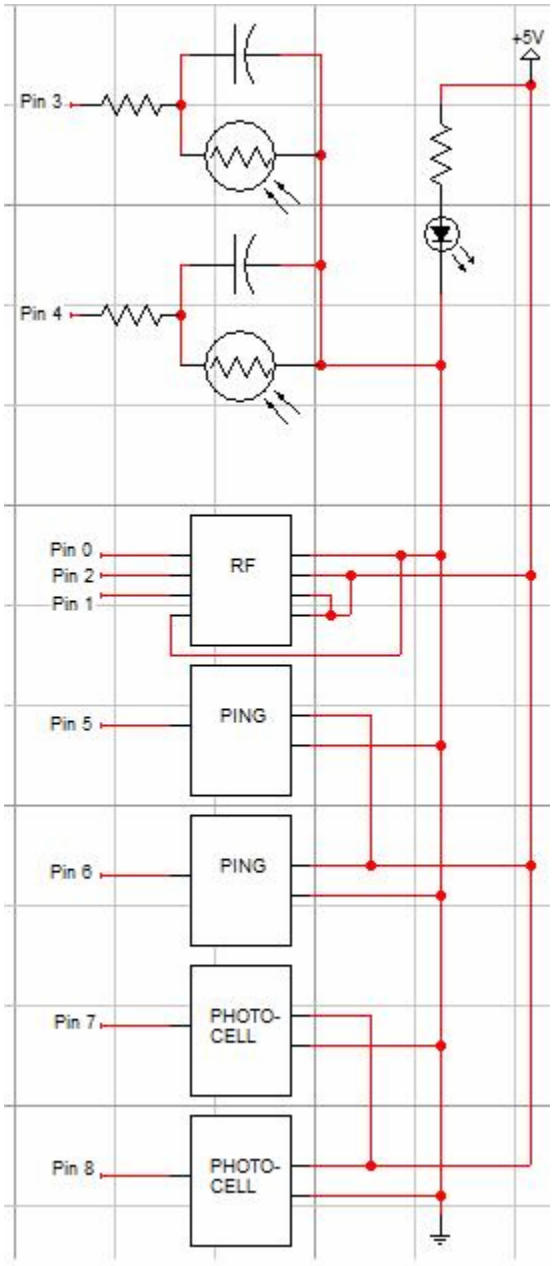

Fig. 5. Circuit diagram.

timely manner. AAROM employs techniques to encourage agent progress while foraging for food. There are four main features of this algorithm that ensure a good solution. These

are the use of zones, the use of a dynamic action sequence, a constant survey for the food, and adequate obstacle avoidance.

The Use of Zones

Throughout the search environment of the robots, a black and white ground is used that create a zoning grid. The use of zones in AAROM serves two purposes: one is to create an awareness that the robot is progressing, and the other is to force the change of action methods. An agent will gain the knowledge that it is progressing when it reaches the border of two zones. Once it sees that it is leaving one zone and entering another one, the robot will end up storing the action method that facilitated the advancement. At this point it is assumed that the robot is one zone closer to the goal. The action method data becomes the guide that will help the follower ant make its own headway when it comes to this part of the environment. Thus, the data put in memory by the scout will be used as pheromone markers to define a trail to the food. The zones are a very important factor in the process of the ants becoming aware that no one area is searched over and over again. In this way, a gradual forward advancement is made towards the goal. Another purpose of the zone differences is to force a change in the action set. Every time a zone is completed, the action mode is changed to the default action of forward search. Changing action sequences periodically is important for finding a path to the goal. If one action is not working, then another might. Thus, it is important to keep the methods changing. The different colored zones are one way of maintaining a fresh approach to the search for the goal. But, this method is not the only way in which the action sequences change.

The Use of a Dynamic Action Set

Action methods may also change according to the value of a variable that measures how many steps the robot makes. These steps do not count pivots, these can be thought of as merely the shifting of the robot on its wheels to face a new direction. The step counter only measures how many forward steps the robot makes. These steps could be in any direction. The default action method is a forward search. After a certain amount of steps of not discovering the food, this method is considered inadequate and a new action method is attempted. The second action method is a right wall hug search. This action will follow a safe distance along the right wall. If the robot has been using this method without ultimate success for too long, then a left wall hug search is performed. Hugging the left wall is the purpose of the left wall hug action. If the left wall hug action is not working, then the default method of searching forward is again attempted. It is through this constant effort to find a route to the next zone that the robot is encouraged to make a forward progression to the next one, and to another sequence of the dynamic action set. As mentioned earlier in this chapter, the algorithm is a depth first search algorithm using a dynamic action set as the nodes it traverses. The action method will be forward for as long as possible before reaching a limit defined by a global counter, and then after this action will attempt the next one in the set.

The Search for Food

What also promotes an eventual discovery of the goal is a thorough search for the food that is represented by a light source. Sensing of the light source happens between each and every step forward. What this means is that there should be little chance for an oversight of the food. The robot should not travel right by the goal when it is on a

constant search for it. When it sees that it is close to the light, it will go on a search for it that is separate than the action set search mode. In this special loop, the robot will perform its own specialized searches if it senses the closeness of the light. It will react to the strength of the light and react accordingly. If the robot feels that it is right up to the source then it will stop and tell the others how it got there. If the robot feels that it was close and somehow loses it again, it will perform a sweep for the light so that it senses its closeness again. If it fails after a few tries to get all the way up to the light, then it will give up and return to the main search loop. What is important about this algorithm is that not only is it called upon every time the robot makes a step forward, but it will also try and forge a path to the light if it senses its presence at all. It is this diligence to find the food source at every step that creates a shorter path to the goal than if the agents were looking for the goal at a longer interval. Without this kind of sensitivity the path would become a long one indeed.

The Obstacle Avoidance Algorithm

Lastly, an efficient obstacle avoidance algorithm keeps the journey to the goal as short as possible. If a robot becomes confused in a corner, then it seeks the best possible direction to get out of its trouble. The goal of this algorithm is not to just try any direction to release itself from its trap, but to find the most likely to get it out. The robot will only pivot a few times back and forth before it makes a sweep to scout out the farthest direction from the closest wall to the stuck robot in which both sensors see a clear path. This means that the robot will not be stuck for long, and the goal can be found in a timely manner.

Algorithm Summary

Each of these algorithms work in cooperation to achieve the main goal of AAROM algorithm, that of finding a locally optimal solution to the shortest path problem. An expensive path would be one where a robot has spent a long time trying to find the goal. The factors that might make the solution a long and timely one are circling back to the beginning, taking a long route following along every obstacle until the goal is found by using one search method, failing to find the light or passing by it, and becoming stuck in a corner for too long. The first problem is solved by the use of black and white zones so that the robot can take note that it is making progress and using this information to change the action method to a forward one from the action sequence so that the robot does not go over the same ground over and over again. The robot will not choose a path that might be longer than need be by hugging the same wall or structure all along one half of the environment so that a long meandering path is made. Instead, the action methods used are constantly changing depending upon the amount of steps made. Also, the light will be found promptly due to the interruption of the robots progression through the maze, and a thorough search for the food when the robot is near it. And, of course, any search will take too long if the robot becomes stuck in a corner without an efficient way to find a quick solution to the problem. As a result of all of these different methods, a route is found using an artificial agent that might achieve a solution from a local optimum. Real ants have their own safeguards making them the natural winners at the foraging game. Next, the natural world will be compared to the artificial ants in AAROM in regards to emergent behavior.

Nature vs. Boe-Bot

Because of reactions to the stimuli in their environment, AAROM agents develop behaviors dynamically during the run-time of a system as a result. These dynamic behaviors can be thought of as the ultimate imitation of biology. In biology, a living creature will change and adapt its behavior in response to an outside stimulus. These changes or adaptations usually result in a better choice for the creature depending upon environmental factors. Even simple creatures like ants will adapt to their environment. It is these reactions, also called reactive behavior, that give the illusion of more complexity to a simple creature like an ant. Since it is also the goal to make a simple and limited agent like a Boe-Bot® choose an optimal action depending upon the environment, producing emergent behavior and the illusion of thought is a very important goal.

Reactive Behavior and Pheromone Trails

One way in which an ant colony might define the concept of reactive behavior is in the way they react to the pheromone chemicals laid out on a trail. The odor causes a reaction in the foraging ants. They will either follow the trail, or they will ignore it due to its fading quality. Thus, we see new behaviors emerging when the ants that are using this trail start to notice a dwindling food supply and stop using as much pheromone. Thus, a behavior that kept ants using the trail is ceased because of a food supply that is not as rich as it was before [12] [2]. This causes most of the decay to occur in relation to a chemical trail. However another factor is simply the time it takes for the chemical to fade. An active trail is constantly being reinforced with scent, so time is not as an important factor as the popularity of the trail [2].

Optimization and Pheromone Decay

Artificial ants will actually thrive on the pheromone information fading at a faster rate than it would in a real ant colony. Optimization in a dynamic artificial system benefits from a pheromone decay that occurs at a very high rate [2]. This kind of fast decay of the pheromone works well in systems that are working to solve something like the traveling salesman problem [2]. In AAROM , there really is no decay of the pheromone as such. This is one way in which this artificial swarm differs from a real one. The reason for this difference is simple. Since the agents do not travel back and forth, there is no reason for the pheromone to fade. However, since each ant is sensing the food source itself, along with following the cues, sometimes more optimal results will develop than the pheromone provides. This may occur when an ant is following the route defined by the action sequence that describes the first scout path. The artificial ant does not follow these instructions blindly. Instead it uses them as a guide. All along the way the ant will still be sensing for food. It will be checking its sensors and “sniffing” for the light source while it follows the general directions. In other words, each agent is given the opportunity to develop emerging behaviors. Instead of following one formula, these agents react to the environment on their own, using the pheromone information only as a guide. Because there is a continuous sensing for the food source, the follower ant might veer off from the guide it was given and get to the food faster than the original scout’s trail. This is because the section of the algorithm that senses and points the robot towards the light also has a function for moving straight towards it that is separate from the function where the follower merely follows the pheromone guide. This possibility for

adaptive behavior lends the appearance of learning. The agent is following the instructions for the best path, yet at the same time the algorithm allows for adaptation.

Straying from the Trail

The process of the follower ant continually sensing for the source instead of just following the trail exactly as it is described is actually similar to what happens in nature. Real ants do not follow the pheromone scent exactly as it is all the time. Sometimes an ant that is following the trail will randomly go off on its own and end up finding a better food source or a better route [2]. In the ant world, these anomalies are extremely important in keeping a constant and efficient trail to food. Another reason for leaving the direct path that pheromones mark, is finding an obstacle in the path. In these cases, the ant does not give up and go home. Instead, one can witness dynamic behaviors developing from these situations. The real ant can still roughly smell the pheromone scent and it finds a way to get back onto the path [2]. In the same way, the agents in AAROM will roughly sense the direction of the source if it is close enough, and will dodge any obstruction that might be in the way as it moves closer to the goal. However, the agent will still follow the general guidelines of the route information. The agents will do all of these things at the same time, just like a real ant would. The ability for the artificial ant to adapt and produce new behaviors makes this a good imitation of biological ants.

Optimization from the Follower Ant as well as the Scout

When an AAROM follower finds a more direct source than the scout, or it finds a way around an obstacle that the scout did not encounter, these are what would be thought of as acceptable situations. An acceptable situation is not always possible though,

of course. Consider a case where a change in the way the follower ant follows the path information may cause one individual to get lost. This really should not happen, but it is possible. For instance, differences in the way the servo-motors of a robot work may cause it to go off course. In this case, there is an adaption in behavior that tries to ensure that even this individual will get to the goal. Once the follower ant has followed the pheromone information to no avail, then it will change into a scout. It is this change from the agent's behavior as a follower that will blaze a new trail to the goal. Of course, this is how a real ant would behave if there were some reason that it simply could not follow the pheromone trail to a food source. In a situation like this, an ant would merely look for another route to the food source or find another one altogether. In AAROM, there would be nothing keeping a follower turned scout from finding a second food source. Thus, it is clear that even in the worst-case scenario where a follower becomes lost, an adaptive behavior provides a fresh opportunity for further optimization. This is very similar to how real ants would behave.

Changing Action Methods

The agents also keep track of how long they use a particular action method so that they do not become stuck on one possible movement technique and never reach the goal. This is an important consideration if the robot becomes stuck, or if it just becomes lost from differences in individual sensors, servo-motors, and just from possible changing conditions. When it is determined that the robot has used this technique for too long then it will "change its mind" and attempt another action from the action sequence. This factor is another example of adaptive behavior at work. The change in action methods attempts

to maintain an overall movement forward that is akin to learning from experience that something isn't working and trying another method.

Ant Movement from the Nest

Although AAROM is an attempt at using the simplicity and beauty of an ant algorithm, mainly seeking the shortest path to a goal, there are ways in which it departs from pure biology. When a scout finds a food source then the occupants of the nest will go back and forth gathering food and bringing back to the nest. In AAROM , the scouts work on finding the goal and they remain there. Their objective then is to get the rest of the agents there. Once the rest have reached this goal the program ends. There is also no need for a pheromone value that decays because as soon as the goal is reached there is no more going back and forth as there is in a real life ant swarm or even in an artificial ant solution for a network. For the scope of this solution, which was merely to imitate the real workings of foraging ants with their use of the shortest path algorithm, there was no need to continue the movement. In a real case scenario using a system based upon this method, there might be a need for the agents to go back to a starting point and return to the target. In this case, an optimization that might come about on the way back to the target could be remembered and transmitted to the rest of the swarm for future reference.

Methodology Summary

AAROM attempts to find not only the first option to the food source, but attempts to find a local optimum. Although it is the first solution that is taken as the shortest route, it is nonetheless important to make each scout as skilled as possible in finding the best solution to this problem. One of the best ways to achieve success in

finding the shortest path is to give the agents the possibility of adaptive behavior. These agents will react to their surrounding conditions to find the most efficient way to get to the food source in the shortest time. These reactions cause a dynamic change in the behavior of the agent that are geared to finding a better solution. These reactions are what define most of the behaviors in a true swarm. Thus, it was deemed important to keep this factor as true to life as possible. For it is with these changes at run-time that optimal choices are eventually found, and that the full power of an artificial swarm based upon biology is realized.

Keeping with one of the most important features of swarm intelligence, simplicity, these reactive behaviors are made possible by some very useful but cheap and uncomplicated hardware. Each of the sensors on each robot is relatively inexpensive, yet sensitive enough for the kind of simple reactions required to make this algorithm dynamic. The ultrasonic sensors give the robot enough time to react before hitting an obstacle. Also, the photo-resistors give the robot enough sensitivity to the light source that they can sense it from about seven inches away. This gives the agent enough time to move itself towards it without missing it altogether or losing it. The photo-resistor on the ground gives the robot the ability to see if it is making forward progress in the environment. The LED that is mounted near it gives it the necessary sensitivity to see the differences in ground color. After the food source is found, the transceiver that is mounted on each robot enables the communication needed to let every robot know how to get to the food. This information is broadcast autonomously by the scout to the followers. Each of these pieces of hardware along with the microcontroller that controls them, creates a robot that can react as an insect might when reacting to its environment.

This chapter concentrated on describing the shortest path algorithm and how AAROM models it. In the next chapter, an exact description of how the algorithm is implemented is examined. Also, the observations made while watching the system operate are discussed. In the process of creating AAROM, some of the limitations from hardware necessitated a change in the implementation. These changes will also be discussed in chapter four.

CHAPTER IV

IMPLEMENTATION DESCRIPTION, OBSERVATIONS AND RESULTS

Project Implementation Description

The implementation can be divided into a controller, a logic module and the program's low level operations. The basic job of the controller is to provide a means of creating a loop. By defining these calls in the loops into two different subsets, one can observe the different goals of the agents in AAROM . One of these goals is to act as a scout to provide a means of finding a food source and communicating this to the other agents still at the nest. For simplicities sake, the agents awaiting instructions are called followers. The other goal is for the follower ants to receive an instruction with the action sequence used to find the food. This plan acts as a virtual pheromone trail. In keeping with swarm intelligence simplicity, each agent has a copy of the same algorithm, with the exception of a flag that indicates the initial state of the particular robot as scout or follower. Thus, the controller sends the agent on a particular logic path determined by its initial state. These two different logic paths have been illustrated in a logic diagram in an attempt to make the implementation easy to describe. The scout logic is presented in one general diagram (Fig. 6), and then the more complex functions are displayed on additional diagrams.

Scout Search Loop

Fig. 6 shows the general search loop taken by the scouts as they look for food.

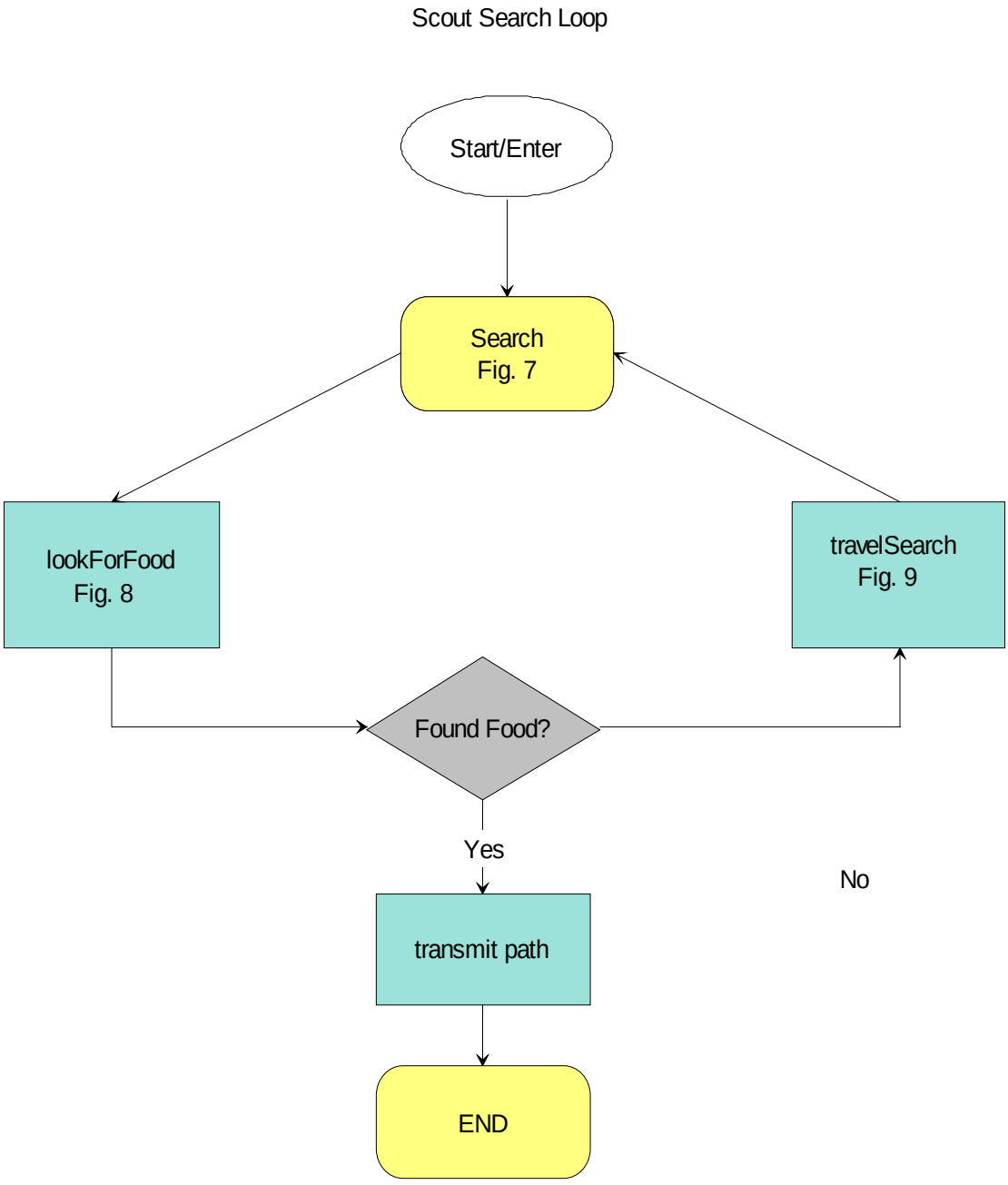

Fig. 6. General scout search loop.

Figure 6 shows the functions that the scout will visit and the order in which it will complete them. This is essentially the controller for the agents with the scout flag initialized. The scout will first go to the *Search function*, then the *lookForFood* function. If the scout has found the food, then the controller loop is completed and the search is terminated. If the scout has not yet found the food, then the *travelSearch* function is implemented and then the *Search* function is entered. The only stipulation for the completion of the program after the food is found, is to communicate the location of the food to the follower ants.

Search Function

The *Search* function is important in itself because it makes a call to the low level sensor functions and gives them an initial value. Figure 7 provides a representation of the *Search* function.

In the *Search* function the ground sensor is read first. This photoresistor sensor measures the ground value. It will either be dark or light. A higher reading will indicate that the robot is traveling over a white section on the path. Alternatively, a lower reading suggests a black section. A count is made of how many times the ground sensor registers a white or black floor each time a pass is made through the *Search* function. If a consecutive reading of the same color is measured fifteen times in a row, then it is assumed that the floor has changed from what it was initially and the current ground color has changed. A step counter called the global counter is reset to zero, so that a new action method will occur in the new block to be searched. The default action method is forward. There should be little chance of trying the same method of searching over and over in one block when the steps are monitored on a global variable, and reset every time

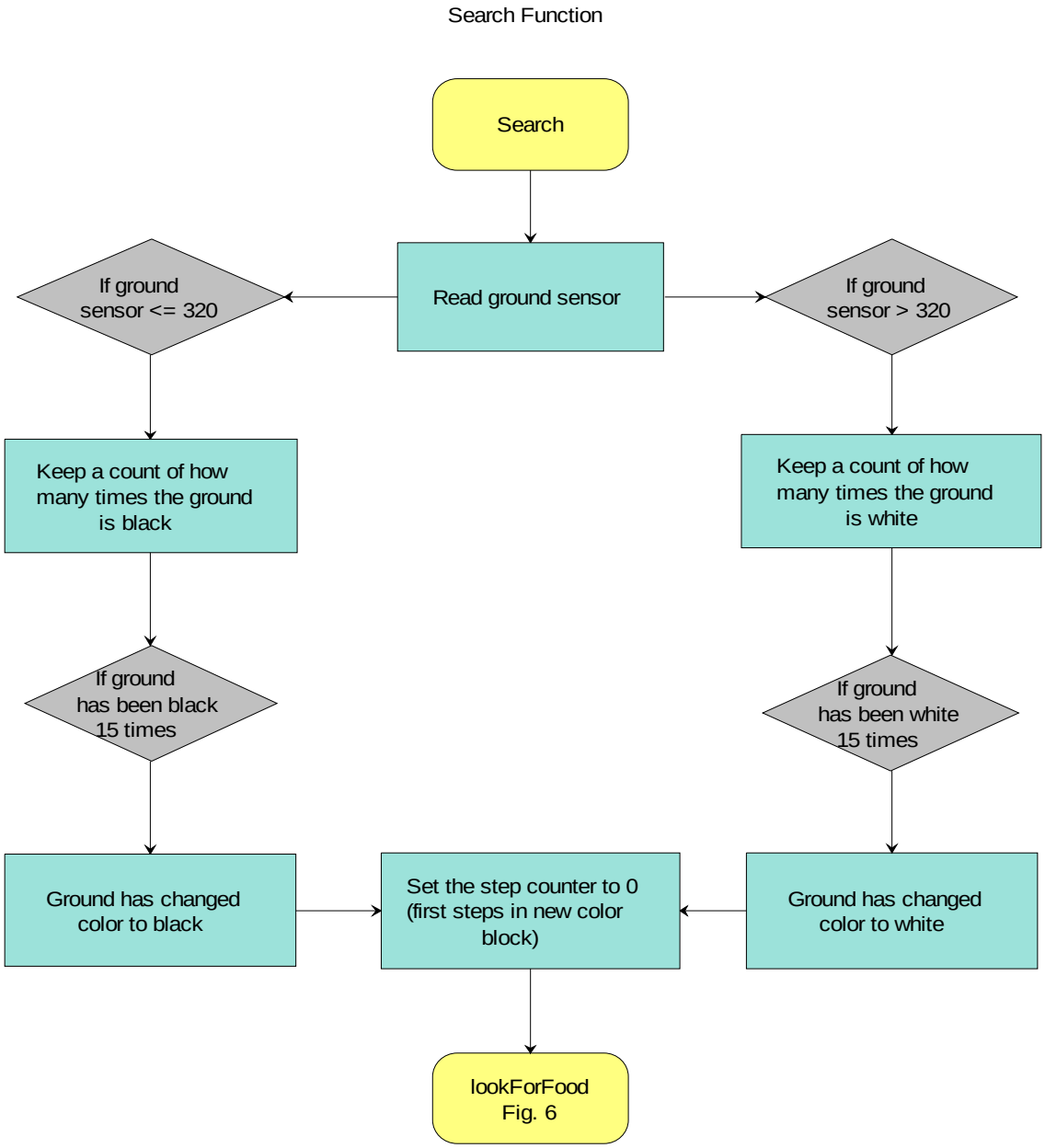

Fig. 7. Search function.

a new zone is entered. This is because once the variable reaches a certain count, the action method changes whether it is going forward or performing a right wall hug search or a left wall hug search. By changing the method of movement, an efficient progression towards the food source is attempted.

lookForFood Function

After the ground check is made, it is up to the robot and its top photoresistor sensor to sense the presence of any food in the *lookForFood* function that is pictured in Figure 8.

The *lookForFood* function concentrates on trying to go directly to the food source if the robot can sense that it is near. The food is symbolized by a light source. Therefore, a photo resistor sensor mounted on the top of the robot is used to sense the food. Once the reading of the sensor is taken, the value represents how close to the food source the robot is. The closeness of the robot to the light source is signified by the value of thirty. If the measurement of the photo resistor is below thirty, then there is sufficient light to assume the robot's close proximity to the food source. In this case, the flow of the program goes on. If the light measurement is above thirty, then the robot must be too far from the food source and a return is made to the function *travelSearch* that works to move the robot closer to the goal. In the case where the robot is found to be in the general area of the food source, another reading of the food sensor is done. If the reading is less than or equal to one, then the scout has found the food and the other ants are alerted using the function *sendMessage*.

In *sendMessage* the data that represents the route used by the scout to find the food are used to create a checksum value. This checksum is calculated by using the XOR logic method on the values in the pheromone array. Once a checksum is created, it is sent out with the pheromone data so that it can be compared with a checksum that the receiving robot will create using the message it heard. This method ensures that the message sent by the scout is the same one that the follower will use. But, this

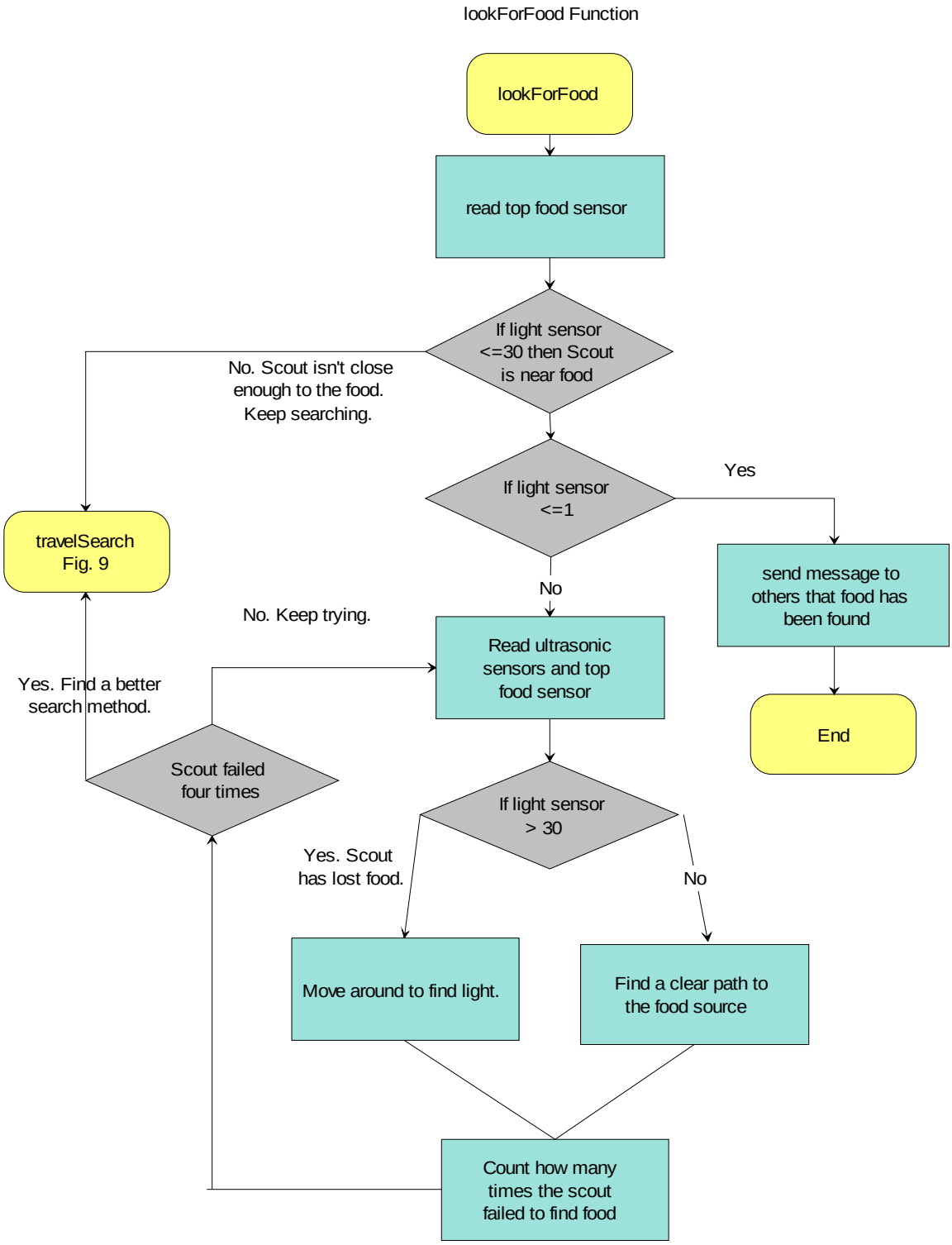

Fig. 8. Lookforfood function.

transmission will only happen if the light value read by the photo-resistor is less than or equal to one.

If the value is above thirty, then it is assumed that the light source has been lost and it needs to be found again. This is achieved by a simple procedure where the robot pivots to the left and right without actually making a movement forward. The reading values at these new positions are taken, and if the light value falls below thirty, then the robot makes sure it is far enough away from obstacles on both sides before it moves forward. If it is above thirty then the same procedure of pivoting is repeated until the source is found again.

As a precaution from the *lookForFood* function being repeated too many times without actually pin-pointing the food, a count is kept of how many times the function is looped through. After four failed attempts of not finding the exact location of the food, then the function enters the *travelSearch* function so that the robot might have a better chance of getting closer to the food. The beauty of this function is that it attempts to find the food in an efficient way so that little time is wasted looking in one spot for the food. If the food is not found then the program progresses to the *travelSearch* function so that some progress towards the goal might be made.

TravelSearch Function

Figure 9 demonstrates the *travelSearch* function. The step counter is a very important component of this program. It helps to determine which action method to use. As shown earlier, the value of the step counter is reinitialized to zero every time the robot enters a new zone *Search* function (see Fig. 7).

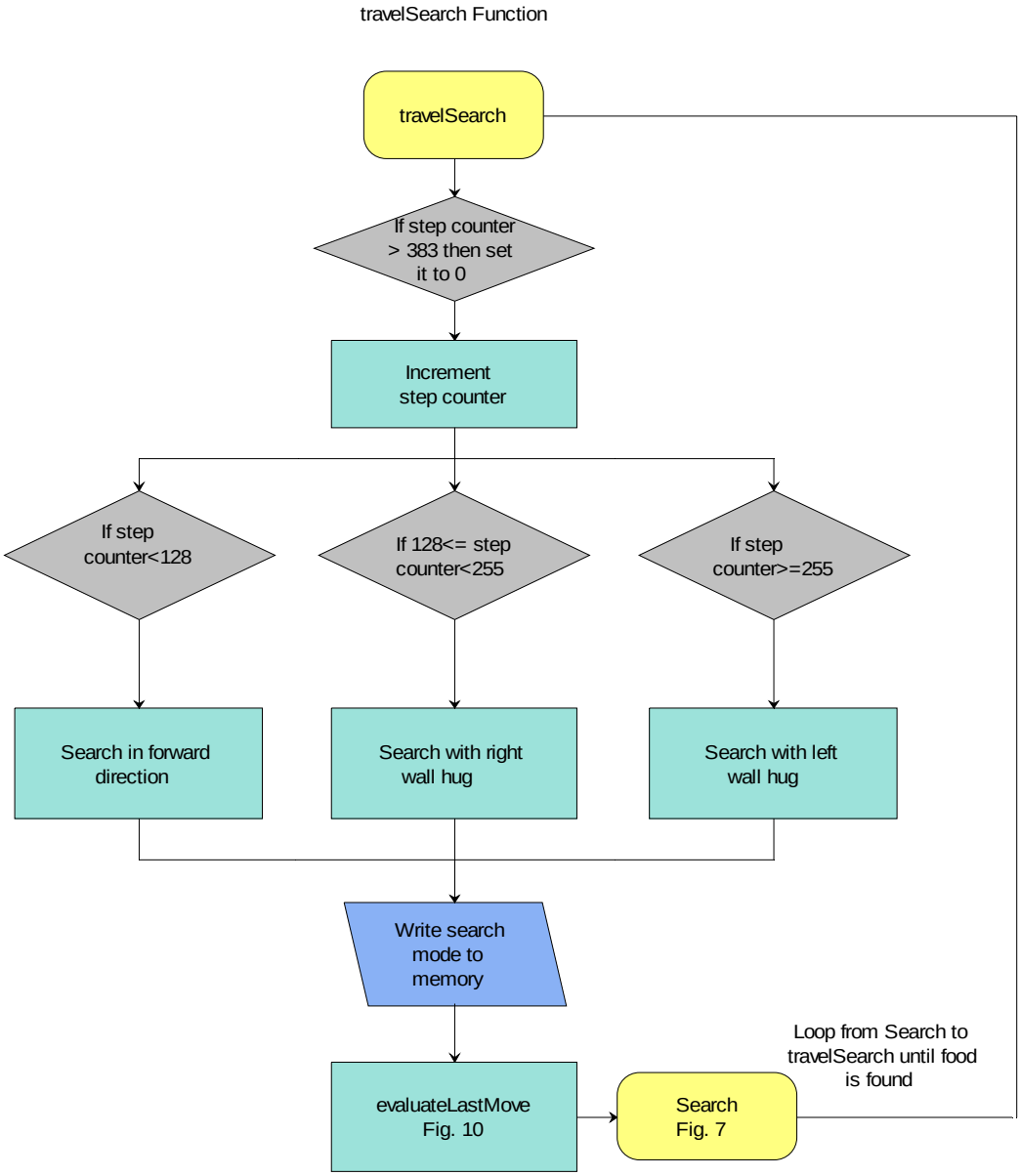

Fig. 9. TravelSearch function.

This facilitates a return to the default action method, *forwardSearch*. The step counter is an indicator of how much ground the robot has covered. At the onset of *travelSearch*, the step indicator is incremented by one. The idea is to keep track of not only how much the robot has moved, but also how much it has used a particular action

method. After the robot has employed the same method for one hundred and twenty-eight steps, the method of action is changed. If it reaches over three hundred and eighty-three steps, marking a complete cycle of using all three action methods in the action set, the step counter is reset to zero. This results in a return to the default action method, *forwardSearch*.

ForwardSearch is a method wherein the robot stays on a straight course between obstacles and involves moving forward. The robot maintains a safe distance between the right and the left walls, and moves in a forward direction. When it nears an obstacle to the right or left, it will pivot away from the obstacle and move forward away from the object. A variable, called *rotateTrys*, keeps track of how many times the robot had to move from the object. This count is kept so that it can be determined if the robot is pivoting left and right in a corner over and over. By recording this count, it will be determined if the robot needs to back up and find a way out of the corner. When the function *travelSearch* has been looped through one hundred and twenty-eight times, then it is assumed that this method is not going to help the robot progress through the maze and find the food. Thus, the next action method in the sequence is performed. This action method is called *hugRightWallSearch*.

In *hugRightWallSearch*, the robot maintains a comfortable distance from the right wall and follows it around in search of the food. If it becomes too far from the right wall it will move closer. If it moves too close to the right wall, it will move away until it is a safe distance again. To make the program more efficient, the action methods are changed if the food hasn't been found. When the step counter has reached two hundred and fifty-five, the mode is changed to *hugLeftWallSearch*. This method is identical to a

hugRightWallSearch but it involves following a safe distance from the left wall on its search through the different zones.

After each different method in the action sequence has been performed, a letter is placed in memory to keep track of which method was used. This information represents the virtual pheromone information that will be saved and communicated to the other ants when the food has been found. The ant merely needs to follow the action methods used to find the food.

EvaluateLastMoveFunction

Next, the *travelSearch* function makes a call to another function called *evaluateLastMove*. This function ensures that the robot does not become stuck and will find an alternate route for it if it is stuck. It is shown in Figure 10.

EvaluateLastMove is a function meant to discover if the robot is stuck. If it is, then it will come up with a new direction for it to go. The variable *rotateTrys* is checked to see if more than eight tries have been made in an attempt to go forward. This variable keeps track of how many times the agent needs to pivot while trying to avoid an obstacle. Depending on which wall the robot saw last, a corresponding function will be called. If it is stuck on the right wall, *findNewHeadingLeft* will be called. Conversely, if it is stuck on the left wall, *findNewHeadingRight* is called. These two functions work to find a new direction for the robot. First, the robot pivots until it is as far from the wall that it is closest to as possible. It then finds a new direction that will be far enough from the wall it is closest to, and finds the first occasion that both ultrasonic sensors see no obstructions. It does this by rotating a set distance, and then returning to the point when the sensors were first both unobstructed. This calculation provides a solution that will be sufficient to

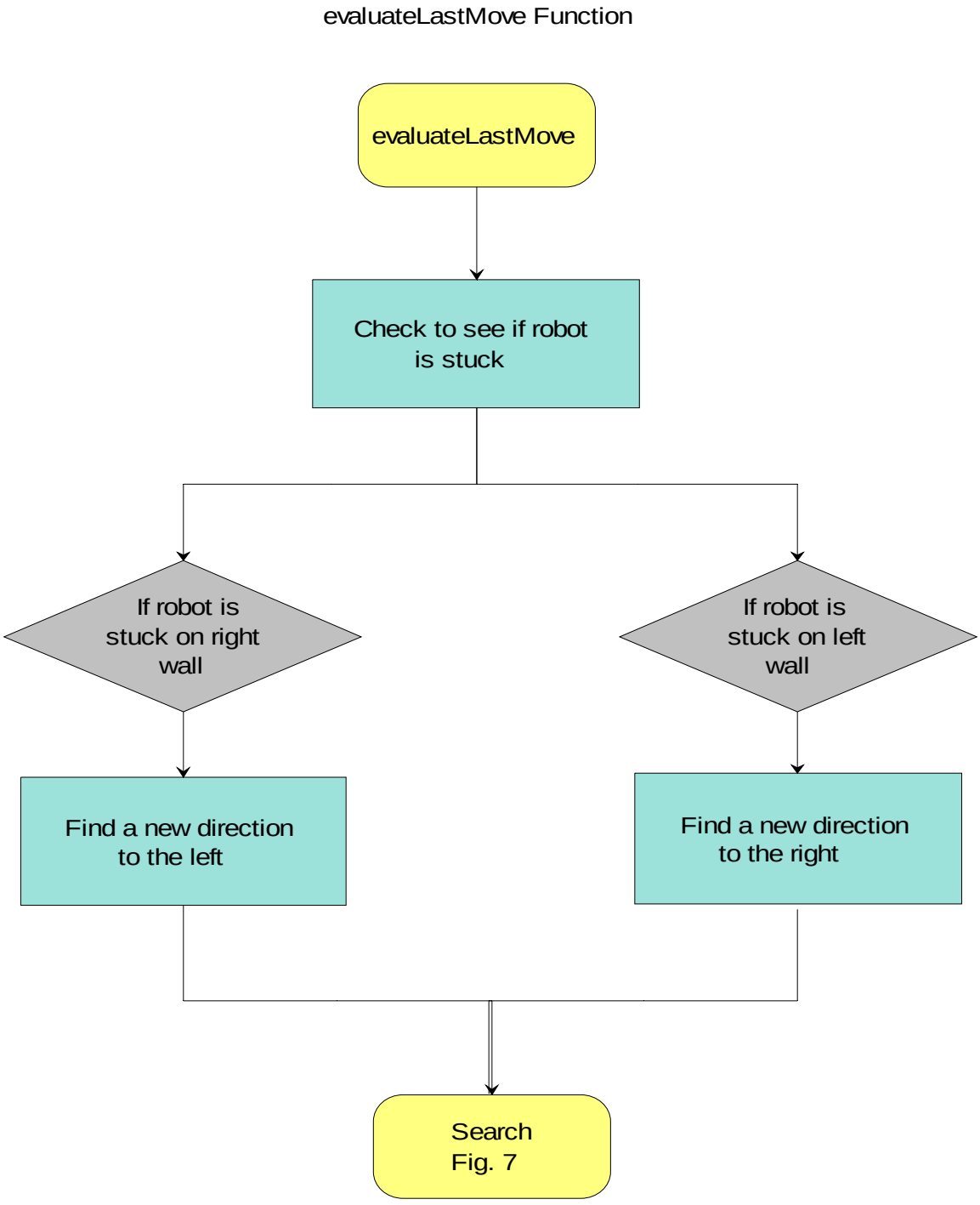

Fig. 10. EvaluateLastMove function.

get the robot out of a corner, but at the same time keep it from going to far away from the original obstruction only to get stuck on another one on the far side.

EvaluateLastMove is then exited and *Search* is visited again. This constitutes the main search loop for the Scout: *Search* is called first, then *lookForFood*, then *travelSearch* and back to *Search,* until the food is found. The scouts have an important job in this program. With the work of the scout agents, the goal is found. They also provide the wireless radio transmission that alerts the other agents of the way in which it found the food. But, in order to have the full benefit of a swarm, the entire group must be at the scene and ready to transmit and receive transmissions, detect obstacles, and detect food.

Follower Ant Search Loop

After the communication is made to the rest of the swarm, it is up to the other follower agents to find the goal. It is the follower search loop that makes this possible (Fig. 11).

The main search loop for the followers uses a *Search* function unlike the scouts. The *Search* function is different for the two types of agents because this determines whether they will go to the function of *travelSearch* or *travelFollow*. The scouts will use *travelSearch,* while the followers use *travelFollow*. But, before the search function is even explored by a follower agent, it must first go to the function *waitForInstructions* (Fig.12).

WaitForInstructions contains a simple loop that listens for a transmission. Once it receives this transmission, then it will go about its own search for the food following the directions given to it by the scout. If no transmission is heard, the

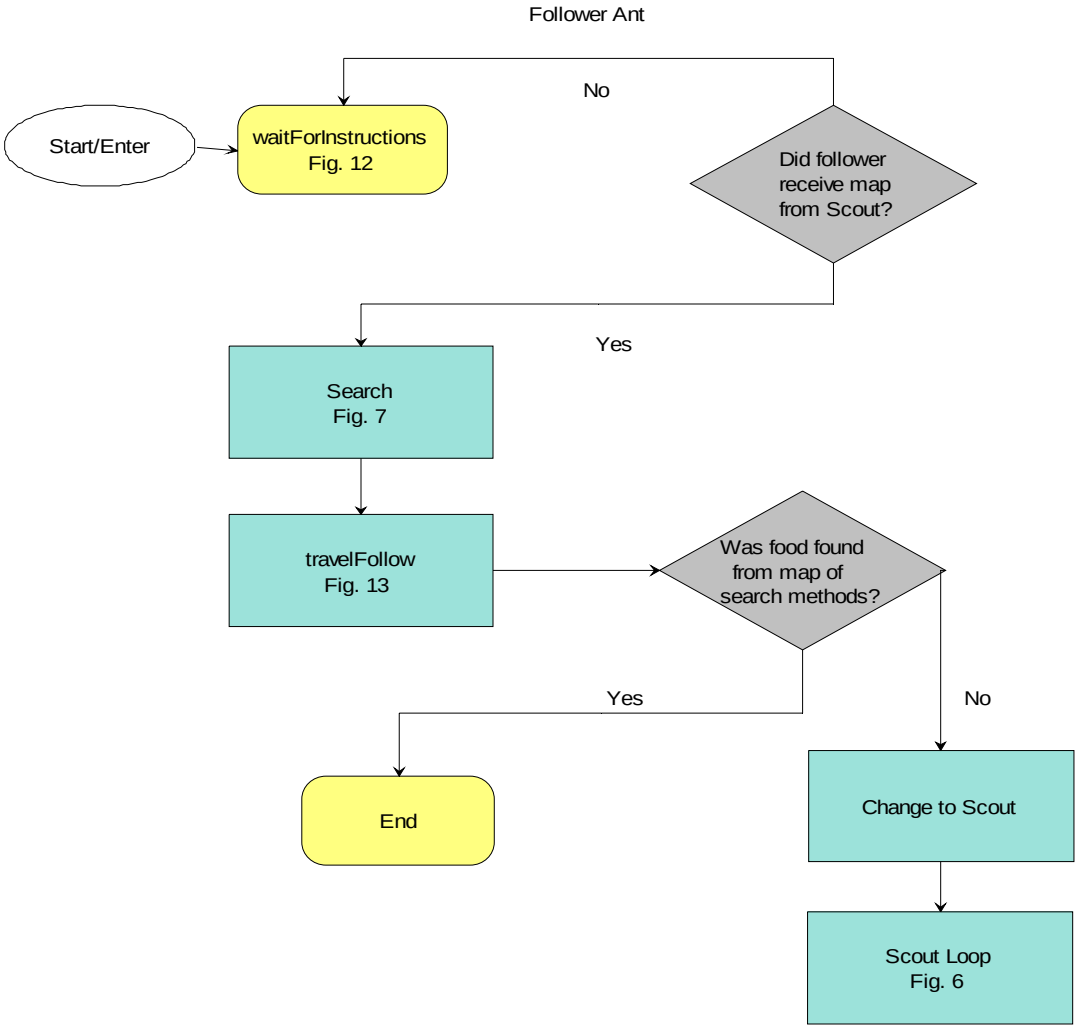

Fig. 11. General follower ant search function.

followers will keep waiting and listening as the scouts continue their search. When the transmissions are received there is an XOR error checking routine that is used to verify that this message is correct. If the checksum calculated on the receiving end matches the checksum that was transmitted by the scout, then the message is considered okay and the follower will leave the *waitForInstructions* loop and start its search using the pheromone information it received.

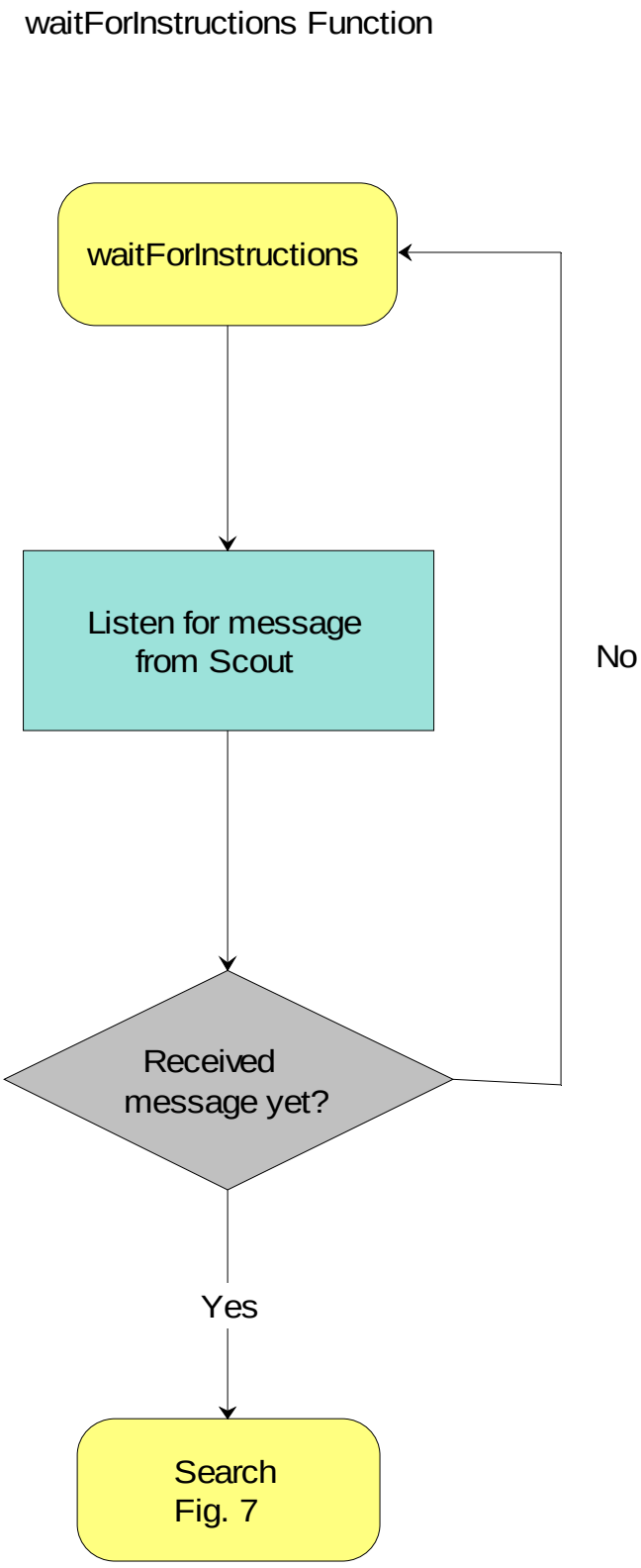

Fig. 12. WaitForInstructions function.

TravelFollow Function

Once the follower gets its transmission, the *Search* function is entered. From here the follower will go to the *travelFollow* function (Fig. 13).

The step counter is as important in the design for the follower as it was for the scout. Like the method for the scout, the step counter ensures that one action method is used too much in the search for the food. The maximum amount of steps for a action method is the same as it is for the scouts, one hundred and twenty-eight steps. If the light isn't found in this amount of steps, then it is reset to zero and the next action method in the instructions is read from memory. Otherwise, if the step counter has not reached that point, the memory is read from beginning to the end as the memory pointer is incremented. Once this value is read the corresponding action method is used to find the food. As previously mentioned, only one hundred and twenty eight steps are traversed using each action method. Once this is met then the memory pointer is incremented and the next action method is used. Another way for the next instruction to be read is if the ground color changes indicating a progression into another zone. Whether the next instruction is reached through too many steps or through finding another zone, the odds are the food will be found since the ant is repeating the same steps that the scout made to reach the food.

Once the memory pointer reaches a standard position, and the follower does not find the food, it is assumed that the agent will not find the target using the instructions it received, and it becomes a scout. This transition occurs so that it still has a chance of getting to the goal. After the *travelFollow* function, the *evaluateLastMove* function is called so that the follower does not get stuck in the process of following its instructions.

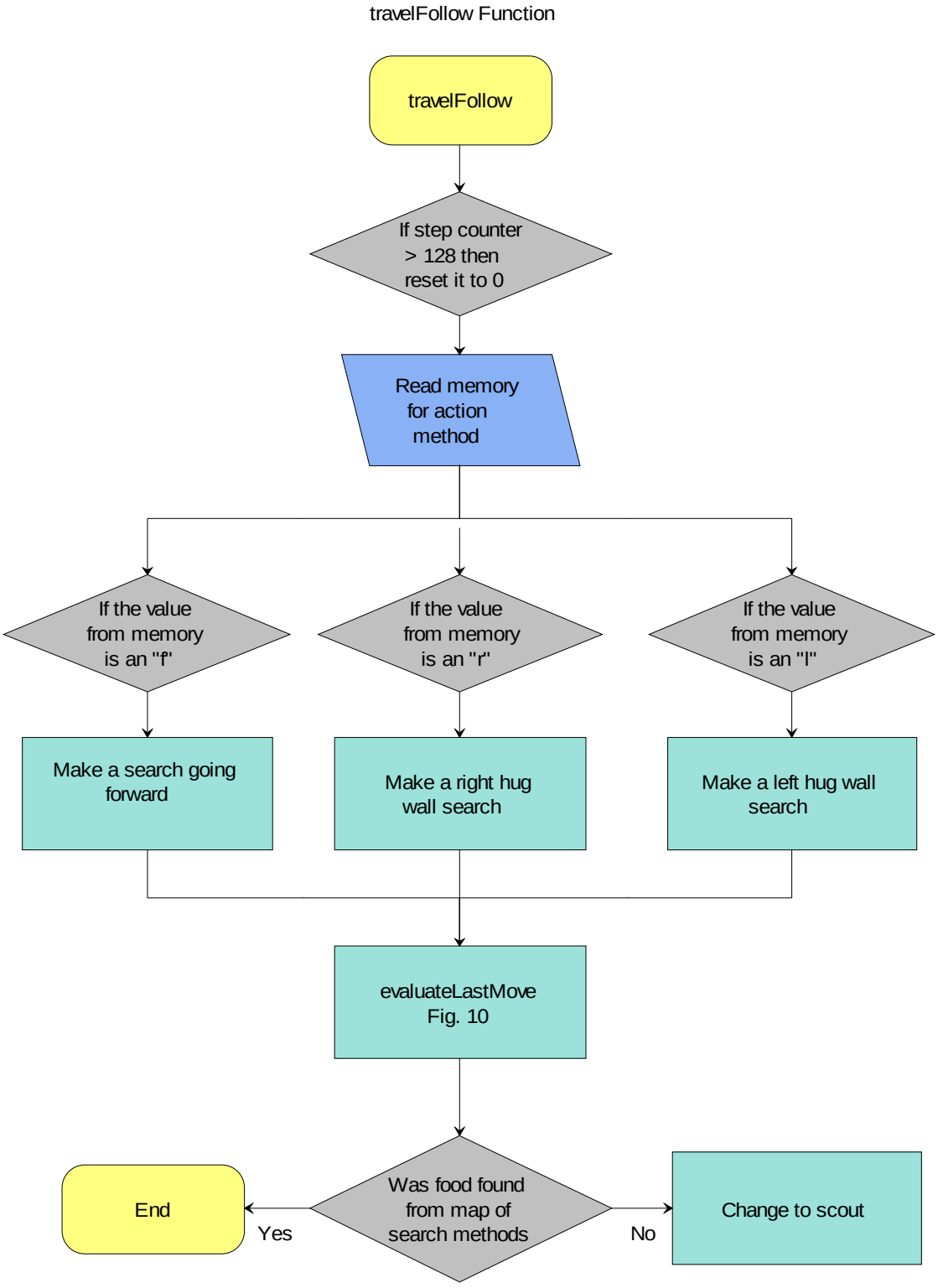

Fig. 13. TravelFollow function.

The *lookForFood* function is also used for the followers. The followers check in with this function during their search loop through *travelFollow* and search functions. This function provides the followers the ability to sense the light for themselves, adjusting for discrepancies so that it is not blindly following a map that may or may not lead them to the goal. It is vital that the followers are able to look for the light source themselves because of discrepancies in the servo-motors of each Boe-Bot®. These differences might result in a pattern that differs from the scout's original directions. When the follower finally finds the light source, it's the program will terminate.

Observations and Results

It has just been discussed how the system is intended to behave. It is common for a program that is run by robots to have glitches in the way it behaves at runtime. Sometimes these run-time glitches are due to the differing behavior of the sensors or the servo-motors that each robot possess. Of course, another possibility is a flaw in the implementation. Because the implementation is flexible enough to allow for slight differences in the sensors and servo-motors, individual hardware idiosyncrasies do not seem to be an issue. There is one glitch that does not seem fixable in software. This will be discussed in later in chapter four. However, on the whole, the agents in the system tend to behave how they are supposed to.

Observations on Obstacle Avoidance

As has been described in chapter three, there has been significant functionality that dealing with not only obstacle avoidance, but getting out of corners and finding new directions. Each of the main search modes is designed to keep the robot a certain distance

from each wall. This keeps the robot from bumping into the walls in between search method functions. The *forwardSearch* function will keep the robot in between the two walls, and the right and left wall hug searches ensure that the robot stays closest to the wall it is to hug. Of course, in each of the functions just described there is functionality to back up the robot if it approaches a wall in front of it. All of these methods intended to keep the robot from running into obstacles work as planned. The ultrasonic sensors are sensitive enough to do a good job at this, and as has been mentioned before, the position of the sensors give it an adequate vision field to sense objects directly in front of the robot and to the side of the agent as well.

Observations on Getting out of Corners

Keeping a robot from hitting a wall successfully is sufficient, but it will not keep a robot from getting stuck in corners without a plan to find a way out. *EvaluateLastMove* addresses the issue of corners (Fig. 10). In obstacle avoidance methods, the robot will either turn to the left or to the right, depending on which sensors see an obstruction. But, if the robot is in a corner, sensing a wall on the left will cause the robot to pivot to the right where the right sensor will see a wall on the left and pivot right. It is clear to see that an unending loop will develop in a situation like this where the robot will never find a way out. There are many ways to solve a situation like this, but the method in this algorithm is a very efficient one. The purpose of this method is to make a sweep search away from the last wall sensed to the first area where both sensors see a clear path. The first area is the one that is used so that an over-correction may be avoided. During runtime, this method works well in getting the robots out of corners. In almost every situation, a new direction is found that gets the robot out of a corner. In a few

situations, it takes a few tries, but in the end the robot always finds a way to get out of tough situations. In the event that a good direction is not found right away, it takes the robot an additional rotation to get out of the situation. These rotations take place in the right direction. Being away from the last obstruction seen. Most cases, the duration of the sweep is sufficient to get the robot out of its dilemma.

Observations on Finding the Light

Another design that works well is that which deals with finding the light, the *lookForFood* function (Fig. 8). Again, the sensors working together with the algorithm bring about adequate results. The success that the photocells have of sensing the light source seems to be supported by the fact that the implementation gives it many opportunities to sense the light. At every step, the photocells are checked to see if they see any light source. If they do sense it at a bright enough intensity, then the photocells are checked again and again as the robot makes a search for the source. If that intensity is bright enough, then the robot has found it. This constant checking for the light source is adequate because the robot finds the light if the source is unblocked. The average distance that the robot finds the food source and stops in front of it is about three inches on average. The robots will usually not run into the light source or passed it by as long as the light source is not blocked.

Observations on XOR Error Checking and Communication

The agents are also successful in communicating to each other. The XOR error checking scheme that AAROM uses will ensure that the message that is received by the follower ants is the same one that is sent by the scouts. If the message is not the same

as the one that is received, then it will go back to listening. The scout will send out another message after two seconds have gone by after the first one in case the ants that did not get the message right the first time. It will repeat this one more time after this. After this third time, the ants will accept the second message and discard any information that does not make sense as it reads it. Since the follower ants have instructions to turn into scouts if the message doesn't make sense, they will still make their way to the goal. This is written in as a safeguard. The communication error checking was more exacting in the original solution, but as will be mentioned in four, much of the security for communications has been sacrificed.

It was noticed that the messages were received correctly by the third time. The follower robots were getting the correct directions to follow, because errors in transmission did not repeat themselves three times in a row. Thus, it was noticed after testing, that the communication and the XOR error checking algorithm was working fine for the purposes of the simple one way communication of AAROM . However, if it was not possible to keep the original error checking method which was more secure. The reason behind the changes in the communication will be explained further later in this chapter.

Issue with Maze Size and Progression

There was one problem found within the *Search* function (Fig. 7). Unfortunately, there seemed to be no way to solve the following problem with the design. This issue arose with the functionality that dealt with using the black and white colored zones to note progression through the maze. The solution was designed to work with any configuration of maze that could be made. The walls of the maze could be in any

configuration as long as there are black and white zones on the ground of this field for the robots to use as a guide. However, during testing it was noticed that the further apart the walls were that a problem arose. Unfortunately, the robots were turning back the way they came and when they noticed that they were approaching a color of ground that was different than the previous one, they would think they were progressing through the maze. There seemed to be no way of solving this issue. When the walls were kept closer together though, the program works just as it is supposed to. The robots notice progression when they are supposed to and generally make a very smooth progression into the next section with no looping around or turning back. In fact, with a narrower maze, the robots will almost most often move directly to the next section of the maze in a very direct way as expected, given the action method of the search technique.

Observations and Results Summary

In conclusion, about eighty percent of the observations made during the testing of AAROM showed positive results. Eighty percent of the behaviors that were observed were not only expected, but were promising. These behaviors were in obstacle avoidance, finding the food source, and in communication. Although the error checking method for the communication is not ideal, it has been found to work sufficiently. The only real problem with the results appears to come from making a maze with wider walls. In this case the robot will become confused. In chapter five, a solution to this issue will be discussed. This was not the only challenge that was encountered while creating AAROM. In the next chapter, the maze and progression problem as well as other issues is discussed.

Project Challenges and Changes

Changes with Communication

Communication Protocol. Originally, the solution was much longer and communication was more complex. There was a lot more communication between robots in the design. This required putting more attention into the way in which the messages were transmitted and received. Essentially, more than one robot might be transmitting and those who were receiving the messages were reading garbled data. The problem was that if two robots were trying to transmit information at the same time, then a message that was being sent out could be interfered with by another message. No amount of error checking would solve this problem. A protocol for communication was needed. Therefore a different approach was created. Now, if a robot wants to transmit some information, it has to listen first prior to transmitting. If it hears any kind of communication, it waits a random period of time before listening again to see if it is safe to transmit. In this way, transmissions will not interfere with each other. This protocol is based on the DCF and PCF layers in the MAC layer used by WAN communication mentioned in chapter two.

Error Checking Changes. In addition to this protocol, there were other parts of the communication solution that needed to be changed as well. In the original version, the XOR error checking scheme was more exact than it is now. Instead of repeating the message three times if the follower ant did not hear it correctly the first time, the follower ant would inform the sender that the message was not correct and to send again. This would ensure that any agent that didn't get the correct message would finally receive it. Without the protocol mentioned above to keep robots from confusing the messages that

were being made, and the need to keep only one robot communicating at a time, the design was changed to repeating the message three times in case the receivers did not receive the correct message.

Hardware Changes. Although the solution worked well, other considerations made it necessary to make the solution a lot less complicated. The design drastically changed once some major hardware alterations were made. When the solution was a lot longer and more complicated, an additional memory board and chip were required. One chip became responsible for the logic and movement of the robot, and the other chip became responsible for communication. There was just not enough space on the BS2pe for logic, movement, and the kind of space needed for error free communication. With the addition of another chip came more issues. One problem that came up was that with the addition of another chip, another battery pack was needed that could hold 5 AA batteries. The standard battery pack holding four batteries did not supply enough power for an additional chip. Before fixing this issue, a power supply was used. With this problem temporarily solved, another one came up that could not be overcome.

The new problem involved chip to chip communication. When working with two chips on one system, it is necessary to create a protocol so that the two chips can cooperate with each other. A protocol was developed to communicate between the two chips. It was then noticed that when the transceiver received some data that needed to be sent to the primary chip, the primary chip would not stop from the flow of its own program to get this information. The implementation failed to create an interrupt. A speculation is that this had to do with the differing speeds of the chips. The primary chip, the BS2pe, runs at 6000 instructions per second and the BS2 runs at 4000 instructions per

second. Many different techniques were attempted to fix this. Finally it was decided that since the robots needed bigger battery packs as well, that maybe simplifying the design would be a better solution. Thus, with a simpler design and less code, two chips would no longer be needed. The secondary boards and chips were then removed from the system, and the transceivers were hooked up to the micro-controller on each agent. When the robots became lighter, another benefit arose in the fact that the robots were lighter and less cumbersome in their movements.

Planning Techniques. Another change in the design that improved the use of memory was changing the way the pheromones stored information. Originally, detailed actions were kept as the pheromone information that was used to mark the shortest path. These actions were basically every movement made by the robot to locate the food source. Using a route defined by action sequences, rather than detailed actions, was found to be more efficient. This means that instead of storing every movement in memory, a series of movements is stored. Discovering this new method produced two benefits. One of the improvements came in the movements of the robots themselves in relation to the pheromone data. Unfortunately, each agent has servo-motors that create differing movements in each robot. This can be solved partly by testing each servo's turn radii and making adjustments in each of their methods, but this isn't practical. The implementation of a route of action techniques gives the follower the ability to make its own reactions to its sensors and the obstacles in the environment so that a better interpretation of the pheromone data is reached. If the follower is relying solely upon the movements that the scout made along the trail, then it would might veer off the path when making movements with its individual servo-motors. With the discovery that a

dynamic set of action sequences could be put in memory instead of action details, the amount of information held in memory is a lot less.

Challenges and Changes Summary

Although the design went through many changes, the previously mentioned changes were the main issues that needed to be addressed. Essentially, the changes were required because of a lack of room for the program in memory. Using the transceivers requires a lot of methods. This became one of the biggest reasons to attempt to put another chip on the robots, and it also became the reason that they needed to be removed. There were issues communicating between the two micro-controllers. Therefore, the communication data was not reaching the micro-controller that controlled the main logic methods and the sensors and motors. When the extra chip was removed it became imperative to simplify the design so that it might fit on the microcontroller. In the process of reducing the solution, some improvements were made. Perhaps the best improvement was the use of a route made from an action sequence instead of action details to direct the follower robots. This makes the relayed directions easier to follow for the followers. All in all, anything to make a swarm intelligence system more simple and efficient while remaining effective is in keeping with the ideals that make these systems attractive.

CHAPTER V

OVERVIEW AND SUMMARY

Overview

There are two foundations that AAROM has been built upon. Swarm intelligence makes up the main overall foundation, and the individual one is ant-based. Mainly, AAROM can be considered an attempt at modeling a situation where autonomous robots use the optimizing benefits of the ant-based shortest path algorithm. The algorithm that was developed was merely a virtual one based upon the strengths of the ant's simple yet skilled solution. Yet AAROM represents at a broader level what any system that is based upon swarm intelligence tries to achieve. The goals of simplicity and efficiency are not unique to each kind of collective system. A program based upon bird flight formation is going to have the same general goals that a system such as this one has. Essentially, the good qualities inherent in any swarm intelligence system include broad coverage of an area due to numerous agents, inexpensive and simple agents that cooperate, uncomplicated algorithms, and most important of all, the possibility for adaptive behavior. Adaptive behavior should develop from an agent that will react quickly to its environment and adjust to any changes by optimizing its actions. While these behaviors are developing it is also important that the different agents cooperate by either communication or simply by their actions for the good of the entire swarm to reach

a goal. This is collective behavior and some swarm in the natural world works in this way.

Features of Swarm Intelligence

The benefits of an artificial swarm are many. Because the swarm is made of hardware that is relatively simple and inexpensive, it will be easy to replace an agent that might get lost or broken in the field. A simple and uniform solution that can be installed on every unit makes the algorithm portable to all of the agents in the swarm. But, perhaps the best benefits come from the solution itself. Programs that are written with emergent phenomena and cooperation in mind will be able to reach an optimal goal more efficiently. This solution attempts to produce these goals. It is created with simple sensors and hardware. The implementation has been kept uniform and simple so that it can be installed easily on all of the agents and it also does not use much memory space. Despite the outward simplicity of the algorithm, the agents change dynamically to sensor readings and have the ability for optimization at any step in the implementation. While scouts have the main job of finding an initial route, follower ants also have the ability to find even better solutions. This is a prime example of emergent phenomena. As a general example of swarm intelligence, this solution achieves the standards that it should. On the level of ant behavior it also is a good example.

The Shortest Path Algorithm

The process by which ants scout and find food has been described in chapter two. A trail is essentially made from the dropping of pheromones and the other ants follow it. The ants communicate through smelling the pheromones and following its trail.

This is called stigmergy. When the ants follow this form of indirect communication along a trail, this is called mass recruitment. The discovery of the shortest path to the food source is reliant on the fact that the first ant to return to the nest trailing its scent will have found the shortest trail. This is what constitutes the basic facts behind the ant foraging problem. Artificial ant path optimization is basically a theoretical solution. The programmer's challenge is to design an algorithm that will find the shortest path to the food in a way that will be efficient and keep hardware cost effective. Of course, after the food is found, the communication of the source is also an important part of the cooperation of a swarm. The swarm must also be able to locate the goal after this communication. All of these factors working together should create an artificial system that is as adaptable and determined as an ant.

Written as it is, this solution strives to find the shortest path employing multiple scouts. The first scout to the goal should not have performed needless backtracking. The design is meant to keep the path as short as possible. This is achieved by utilizing a methodical search technique that keeps track of progression using different colored zones. A simultaneous search is constantly seeking the light. Once it sees this light, all other duties are forgotten, such as keeping track of the zone colors, and the food is approached. Both of these factors work together to attempt to find a short to the goal. The followers also use these abilities so that they can pursue the same path that the scout did.

Purpose and Future Improvements

Thus, the design is effective at being a general swarm intelligence system and an ant foraging model. However, AAROM could benefit from certain improvements. When a scout that has not initially found the goal receives routing instructions on how to use the action sequence used by the successful scout, the agent is not leaving from the nest location but from where it was in the environment when it received its transmission. Thus, the routing information will not describe the directions based upon the original starting location of the nest. The scout will end up becoming lost and will eventually traverse the maze until it finds the goal. A solution could be designed in the future that could address this issue that involves performing a calculation on both routes. This calculation might determine a route for the losing scout that finds a common zone used by both it and the winning scout. The loser could go back to this common zone and move on from that point in the winning scout's action sequence.

Another feature of the implementation that could be improved is the way route information is defined. The information described by the route data is a partial path. It cannot be termed a full path because it does not contain information like time and distance along with its list of action sequences. Information that describes the time to traverse the route and the distance traveled could help in certain route calculations that might be needed, such as, the calculation just mentioned to find the most efficient path for the losing scout from its unique starting point. Maintaining time and distance would also make following the route information much easier for the followers.

Followers and scouts have the same implementation information, except for the value of a flag that determines its role. When the design is uploaded onto the agent,

the flag is set for either scout or follower and this determines the agent's function. It would be much more efficient for a switch to be added to the agent that can be used to change an agent from a scout to a follower or vice versa. This switch would mean that every time an agent needed to change roles, the switch would be responsible for changing the flag, and the design would no longer need to be uploaded every time a change was desired. ..

The benefit of creating a model like this is to get an idea of what would be needed to build a bigger system that would work in a real life environment. Even though AAROM is based upon colored ground zones and the walls of a maze, a real-life solution could be envisioned that builds upon this solution. Obstacle avoidance will be essentially the same in a real-life environment so these parts of the design could be used. However, the surface of a real-life environment will not be the same. Special robots will need to be used which can deal with surfaces like potholes and other terrain difficulties. Though, ultrasonic sensors can still be efficient in real life environments, using a vision system that utilizes an infrared camera and a GPS system can be more efficient at avoiding obstructions.

As previously mentioned, the colored ground zones are used to delineate a progression through the environment. Obviously, there will be no black and white ground surfaces in an outdoor environment. Also, as mentioned in chapter four, a flaw in the method arises if the walls of the maze are too far apart. In this case, the robot might end up going backwards and think that it is progressing when it is not. There was no solution found for this issue. The robot could sense two different color changes and this works as long as the robot does not start going backwards to mark a progress, but the robots have

no way to “see” that they are actually not making any forward progress. One way to solve this problem would be to add a GPS receiver to each robot. These would work much better than photo-resistors to gauge progress. The zones would no longer be needed because the robot would be aware that it is moving away from its starting point. In this way, zones would no longer be an important issue, but the overall idea of traveling in certain units of distance could be maintained to determine that progress has been made and to keep the robot from making unnecessary loops and backwards trips. This would not require too many changes in the design since the overall concept of making progress through certain zones would be maintained. These zones would now be made from coordinates of meters and these distances calculated accurately by a GPS receiver. The scouts could use accurate distance and location information to store and communicate a more accurate map to the followers.

Instead of looking for a light source, a search and rescue system looks for people. Since a living person radiates heat, a thermal infrared camera could take the place of photo-resistors. However, the search functionality could remain relatively unchanged. Another thing that could remain unchanged is the use of wireless communication to bring the “swarm” to the target. Seamless wireless communication is essential for any autonomous cooperative system.

Of course, the number of agents would definitely change. The number of scouts would probably involve the whole swarm. It would be much more advantageous to have the whole swarm search for the target at the same time. In this way, large areas could receive good coverage. Since these agents would be moving in precise zones because of their GPS receivers, there would be no overlap. The zones could be measured

in square meters so that adequate coverage could be maintained. This would not change the uniformity of the implementation however, since these zones would be defined from wherever the robot started.

Summary

In conclusion, AAROM was created to serve as a model for swarm intelligence and for ant intelligence in particular. But, as has been detailed a real-life design for search and rescue can be built upon this solution as a guideline and foundation and how to seek an optimal and shortest path to a target. In order to achieve a viable real-life system, some hardware changes would need to be made. Because of the changes in sensors some changes to the implementation would need to be made, but overall AAROM system can act as a template for a larger system. Using a GPS receiver as a way to sense the robots progress through an environment would actually improve the design greatly. The accuracy, and the ability to find an optimal solution could achieve a result much closer to that of a biological ant. One might wonder how ants find an optimal goal after meandering around looking for food. What is known is that on the trip back to the nest, they are able to keep track of their position in relation to the nest and they use this information to create a more direct path to the goal [12]. It seems they have something similar to a GPS system. It is interesting that these insects can be so misleading. It would seem that they are more complex than they really are. Yet it is through reactive behavior to what they sense in the environment, and cooperation that a swarm of ants can find the best path to a goal. This is truly the beauty and the purpose of imitating them. Even the ants' knowledge of their position to their nest as they return to it keeping the shortest path

possible seems mysterious and difficult. Yet, this still is only a reaction to some sort of navigation sense that gives them this ability [12]. This sounds very much like a simple robot. For what is a basic robot but a micro-controller, a motor and some sensors. The benefits of an agent that is based mainly upon reactions is clearly evident when that robot is deployed in a difficult environment where an efficient search algorithm is required and simple decisions can be made with the proper sensors. It is these simple decisions that become the emergent behavior that help to solve difficult problems. The more simple the decisions, the less demanding of a system is needed overall, and it is this practicality that makes swarm intelligence such an important element of artificial intelligence.

REFERENCES

REFERENCES

[1] D. Marco, “About ACO: Behavior of Real Ants,” [Online document], [2006 Nov. 8], Available at FTP: http://iridia.ulb.ac.be/~mdorigo/ACO/RealAnts. html

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford, New York: Oxford University Press, 1999.

[3] T. DeWolf, L. Jaco, T. Holvoet, and E. Steegman, “A Nested Layered Threshold Model for Dynamic Task Allocation with Ants,” *Proc. Third International Workshop on Ant Algorithms*, pp. 290-291, 2002.

[4] A.F.T., Winfield, C. Harper, and J. Nembrini, “Towards the Application of Swarm Intelligence in Safety Critical Systems,” *Proc. First Institution of Engineering and Technology International Conference*, pp. 7-14. Bristol: Univ. of the West of England. 2006.

[5] B. Steigerwald, “Shape-Shifting Robot Nanotech Swarms on Mars,” *NASA Astronaut Journal*, 2005. Retrieved April 12, 2008 from the World Wide Web: http://www.nasa.gov/vision/universe/roboticexplorers/ants.html

[6] H. Nezamabadi-pour, S. Saryazdi, and E. Rashedi, “Edge detection using ant algorithms,” *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 10, issue 7, pp. 623 – 628, May. 2006.

[7] J. Penders, “Robot Swarming Applications,” *Liber Amicorum*, H. Jaap, Van Der Herik, Maastricht University, pp. 227-234, 2007.

[8] F. Steele, Jr., and G. Thomas, “Directed stigmergy-based control for multi-robot systems,” Proc. of the ACM/IEEE international conference on Human-robot interaction, pp. 223 – 230. New York: ACM. 2007.

[9] G.K. Venayagamoorthy, and R.G. Harley, “Swarm Intelligence for Transmission System Control,” Proc. Power Engineering Society General Meeting, pp. 1-4. Florida: IEEE. 2007.

[10] D. Bruemmer, “Urban Search and Rescue: Search and Rescue Experiments,” *Idaho National Laboratory*, 2006. Retrieved April 11, 2008 from the World Wide Web: http://www.inl.gov/adaptiverobotics/urbansearch/experiments.shtml [11] M. Hinchey, R. Sterritt, and C. Rouff, “Swarms and Swarm Intelligence,” Computer, pp. 22-28, April, 2007.

[12] J. Klotz, D. Williams, B. Reid, K. Vail, and P. Koehler, “Ant Trails: A Key to Management with Baits,” [Online document], [2000 Sept.], Available at FTP: http://edis.ifas.ufl.edu/IG123

[13] H.G. Nguyen, N. Pezeshkian, A. Gupta, and N. Farrington, “Maintaining Communication Link for a Robot Operating in a Hazardous Environment,” *Space and Naval Warfare Command*, March, pp. 108-117, 2004.

[14] D. Bruemmer, D. Dudenhoeffer, M. Anderson and M. McKay, “Components of Swarm Intelligence,” *Proc. 10th International Conference on Robotics and Remote Systems for Hazardous Environments*, pp. 231-238. Idaho: INEEL. 2004.

[15] G. Prencipe and V. Gervasi, “On the Intelligent Behavior of Stupid Robots,” in VIII Convegno AI*IA, Siena (2002).

[16] M. Dorigo, G. Di Caro, and L.M. Gambardella, “Ant Algorithms for Discrete Optimization,” Artificial Life, vol. 5, no.2, 1999, pp. 137- 172.

[17] D.J. Bruemmer, D.D. Dudenhoeffer, M.D. McKay, and M.O. Anderson, “A Robotic Swarm for Spill Finding and Perimeter Formation,” *Proc. of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 231-238.

[18] J. Geier, “802.11 MAC Layer Defined,” *Wi-Fi Planet*, [Online Serial], June 4, 2002, [2005, Nov. 12], Available at FTP: http://www.wi-fiplanet.com/tutorials/article.php/ 1216351

[19] Z. Mason, “Programming with Stigmergy: Using Swarms for Construction,” Artificial Life, vol. 8, no. 2, 2002, pp. 371-374.

[20] D.W. Gage, “Sensor Abstractions to Support Many-Robot Systems,” in *Proc. of SPIE Mobile Robots VII*, 1992, pp. 235-246.

[21] H. Asama, D. Kurabayashi, K. Kawabata, T. Fujii, H. Kaetsu, I. Endo, M. Kusakabe, A. Yoshiki, T. Ebisuzaki, and H. Tashiro, “Emergence in distributed autonomous robotic systems towards symbiosis engineering by using ubiquitous devices,” Aiken Review, no. 36, June 2001, pp. 16-31.

[22] Parallax, Inc. “BASIC Stamp BS2pe Module,” 2007. Retrieved April 15, 2008 from the World Wide Web: http://www.parallax.com/Store/Microcontrollers/BASICStampModules/tabid/134/txtSearch/bs2pe/List/1/ProductID/6/Default.aspx?SortField=ProductName%2cProductName

[23] Parallax Inc., *RF Module Manual*. Rocklin, Calif.: Parallax Inc., 2000.

[24] Parallax, Inc. “Ping Ultra-Sonic Sensor,” 2007. Retrieved April 15, 2008 from the World Wide Web: http://www.parallax.com/Store/Sensors/ObjectDetection/tabid/176/CategoryID/51/List/0/Level/a/ProductID/92/Default.aspx?SortField=ProductName%2cProductName.

[25] Parallax, Inc. “Photoresistor,” 2007. Retrieved April 15, 2008 from the World Wide Web: http://www.parallax.com/Store/Sensors/ColorLight/tabid/175/CategoryID/50/List/0/Level/a/ProductID/175/Default.aspx?SortField=ProductName%2cProductName

[26] G. Fedoriw, C. Fehr, M. Goode, and S. Keown, “Application of Swarm Intelligence Solving the Shortest Route Problem,” Canadian Artificial Intelligence Magazine, Spring, 1997, pp. 33-46.

[27] S.A. Stoeter, P.E. Rybsky, M.D. Erickson, M. Gini, D.F. Hougen, D.G. Krantz, N. Papanikoloupos, and M. Wyman, “A Robot Team for Exploration and Surveillance Design and Architecture,” in *Proc. of the IEEE Mediterranean Conference on Control and Automation*, 2000, pp. 9-16.

[28] F. Michaud and M.J. Mataric, “Learning From History For Adaptive Mobile Robotic Control,” Agents 1998, 1998, pp. 442-448.

[29] C. Wood, “SWARMS IN THE MACHINE: Mimicking Biological Systems in Distributed Artificial Intelligence,” [Online document], [2005 Nov. 30], Available at FTP: student.bennington.edu/~cameo/assets/Papers/SWARMS_IN_THE_MACHINE.pdf

[30] T. White, “Swarm Intelligence: A Gentle Introduction with applications,” [Online document], [2005 Nov. 17], Available at FTP: http://www.sce.carleton.ca/netmanage/tony/swarm-presentation/index.htm

[31] M. Kristina, and P. Illias, “Movement Optimisation of Cooperating Ant Colony: A Study in Agent–based Social Simulation,” *Technical University*, 2000. Retrieved April 12, 2008 from the World Wide Web: http://www.ici.ro/ici/revista/sic2007_4/art07.html

[32] P. Barron and V. Cahill, “Using Stigmergy to Co-ordinate Pervasive Computing Environments,” in Sixth *IEEE Workshop on Mobile Computing Systems and Applications*, 2004, 112-122.

APPENDIX A

PROJECT IMPLEMENTATION

```
'{$STAMP BS2}
' {$PBASIC 2.5}
'*****************************************************************************************
'* Program: ant.BS2     Author:James Ross, Kristin Eicher
'* Date:   4/5/08      Revision: 4.0                    *
'gen ant code.
'*****************************************************************************************
'declares
'*************************************************************************
'program vars
'=========================================================================
'=========================================================================
'ping sensors
wallDistRight VAR Word  'left sonic
wallDistLeft VAR Word  'right sonic

'memory
Value  VAR Byte(6)      'the memory

'movement
rotateTrys VAR Nib     'rotation attemps
forwardSteps VAR Nib     'F steps attempts
rotateRight VAR Nib     'R movement attempts
rotateLeft VAR Nib      'L movement attempts
dir VAR Nib         'current direction for fixing if stuck in corner
ndir VAR Nib         'new direction for fixing if stuck in corner
isScout VAR Bit       'is this robot a scout

'light sensor
lightSensor VAR Word     'to detect if light(food) is found
whiteGroundCounter VAR Nib  'detect if ground color is white
blackGroundCounter VAR Nib  'detect if ground color is black
foodCounter VAR Nib     'count consecutive time see food

'change in light sensors
foundTarget VAR Bit     'found the food bit
groundChange VAR Bit     'Ground color changed 1= white 0 = black
preGroundChange VAR Bit   'prev ground color

'gen vars
genVarX VAR Nib       'general variable
genVarY VAR Nib       'general variable
```

```
counter VAR Nib       'gen counter
index VAR Nib        'gen index var

'send receve vars
Checksum VAR Byte      'Calculated Checksum
PacketCount VAR Byte     'Packet Number and Data Value Count
globalCounter VAR Word    'step counter
generalCounter VAR Byte
Packet VAR Nib       'Packet Number
DCount VAR Nib       'Data count
memPtr VAR Nib        'map memory pointer

'-------------------------------------------------------------------------------

'---------------------assignments-----------------------------------------------
'rf const
Rx   CON 1         'Receive I/O pin number
Tx   CON 0         'Transmit I/O pin number
N9600  CON $4054      'Baud mode value for 9600 baud, 8,N,1
TxFlow CON 2

rotateTrys = 0
globalCounter = 0
generalCounter = 0

isScout = 0        '1 = do scout no help: 0 = worker ant with data from scout

Value(0) = "f"      'the Map
Value(1) = "f"
Value(2) = "f"
Value(3) = "f"
Value(4) = "f"
Value(5) = "f"

memPtr = 1

'is this a follower ant(if yes listen for instructions)
 IF(isScout= 0) THEN
  GOTO waitForInstructions
 ENDIF
'-------------------------------------------------------------------------------

'Controller
'*******************************************************************************
'basic control operation of robot
'===============================================================================
'===============================================================================
Main:



  'start by reading the sensors
  GOSUB search
  IF isScout = 1 THEN
```

```
  'this is a follower ant; it will try using the 'map'
  GOSUB travelFollow
 ELSE

  'this is a scout ant try to find the targe with no 'map'
  GOSUB travelSearch
 ENDIF

GOTO main
END

'logic model:
'*****************************************************************************
'moveing, getting unstuck R/L, determining food source,
'determin progress To target, search methods,
'=============================================================================
'=============================================================================
'read all censors and evaluate
Search:

 'get the distance from the walls
 GOSUB readSonarRight:
 GOSUB readSonarLeft:

 'check ground color
 GOSUB readGroundSensor
 GOSUB evaluateGround

 'check if the front sensor see food(bright light source)
 GOSUB lookForFood:

RETURN
'=============================================================================
'=============================================================================
'determin next travel method
'(forward search default, left wall hug if progress not being made then right wall hug)
travelSearch:

  'move back if too close to wall(2 inches minimum)
  'IF (wallDistRight < 2 OR wallDistLeft < 2) THEN

  'GOSUB Move_B
  'ENDIF

  'move left or right if the left or right side is to close to a wall
  'IF (wallDistLeft < 3 AND wallDistRight > 3) THEN GOSUB Move_R
  'IF (wallDistLeft > 3 AND wallDistRight < 3) THEN GOSUB Move_L

  'count each iteration throught the travelSearch function (reset after 383 iterations [128 * 3])
  'globalCounter = globalCounter + 1
   IF(globalCounter > 383) THEN
   globalCounter = 0
   ENDIF
```

```
 'the search
 IF isScout = 1 THEN

  'if this is a scout ant
  'choose forwardSearch,leftWallHug OR rightWallHug AND
  'update 'map'(1 time everytime gnd color changes)

  'forward search
  IF globalCounter < 128 THEN
   GOSUB forwardSearch
   Value(memPtr) = "f"
  ENDIF

  'left wall hug travel
  IF globalCounter >= 128 AND globalCounter < 255 THEN
   GOSUB hugRightWallSearch:
   Value(memPtr) = "l"
  ENDIF

  'right wall hug travel
  IF globalCounter >= 255 THEN
   GOSUB hugLeftWallSearch
   Value(memPtr) = "r"
  ENDIF

  GOSUB evaluateLastMove:
 ENDIF

RETURN
'================================================================================
'================================================================================
'determin next travel method
'(forward search default, left wall hug if progress not being made then right wall hug)
travelFollow:


 'move back if too close to wall(2 inches minimum)
 IF (wallDistRight < 3 OR wallDistLeft < 3) THEN GOSUB Move_B
 IF (wallDistRight < 3 AND wallDistLeft < 3) THEN
 GOSUB Move_B
 GOSUB Move_B
 GOSUB Move_B
 GOSUB Move_B
 GOSUB Move_B
 ENDIF

 'move left or right if the left or right side is to close to a wall
 IF (wallDistLeft < 3 AND wallDistRight > 3) THEN GOSUB Move_R
 IF (wallDistLeft > 3 AND wallDistRight < 3) THEN GOSUB Move_L


'count each iteration throught this function (reset after 128 iterations )
'try this search method for 128 steps before giving up
```

```
IF globalCounter > 128 THEN
 globalCounter = 0
 memPtr = memPtr + 1
ENDIF

'select the search method
SELECT Value(memPtr)
CASE "f"
 GOSUB forwardSearch
CASE "l"
 GOSUB hugRightWallSearch:
CASE "r"
 GOSUB hugLeftWallSearch
CASE ELSE

 'junk data don't know what to do look at the next instruction
 memPtr = memPtr + 1
ENDSELECT

'look at last movment
GOSUB evaluateLastMove:

'if did not find the target after 6 instructions give up (follower-to->scouts)
IF memPtr > 6 THEN
 isScout= 1
ENDIF

RETURN
'================================================================================
'================================================================================
'search by always moving forward until can't move
forwardSearch:

 'always move back IF about TO hit a wall
 IF (wallDistRight <= 2 OR wallDistLeft <= 2) THEN

  IF(wallDistRight < wallDistLeft) THEN
  DO WHILE wallDistLeft < 5 AND wallDistRight <5
   GOSUB readSonarleft
   GOSUB readSonarRight
   GOSUB Move_R
  LOOP
  ENDIF

  IF(wallDistRight > wallDistLeft) THEN
  DO WHILE wallDistRight < 5 AND wallDistLeft <5
   GOSUB readSonarRight
   GOSUB readSonarleft
   GOSUB Move_L
  LOOP
  ENDIF

   IF(wallDistRight = wallDistLeft) THEN
  DO WHILE wallDistLeft < 5 AND wallDistRight <5
```

```
  GOSUB readSonarleft
  GOSUB readSonarRight
  GOSUB Move_R
 LOOP
 ENDIF

 ENDIF

 'too close to the right wall move away and count times I had to move away from right wall
 IF(wallDistRight < 3 AND wallDistleft > 3) THEN
  GOSUB Move_R
  rotateLeft = rotateLeft + 1
  rotateTrys= rotateTrys+1
  forwardSteps = 0
 ENDIF

 'too close to the left wall move away and count times I had to move away from left wall
 IF(wallDistLeft < 3 AND wallDistRight >3 ) THEN
  GOSUB Move_L
  rotateRight = rotateRight + 1
  rotateTrys= rotateTrys+1
  forwardSteps = 0
 ENDIF

 'not close to left or right wall. move forward and reset rotation counter
 IF(wallDistRight >=3 AND wallDistLeft >=3 ) THEN
  rotateTrys = 0
  forwardSteps = forwardSteps +1
  GOSUB move_F
 ENDIF

RETURN
'================================================================================
'================================================================================
'search by following the right wall
hugRightWallSearch:

 IF(wallDistLeft <=3 ) THEN
   GOSUB move_l
 ENDIF

 IF(wallDistLeft >=7 ) THEN
   GOSUB move_R
 ENDIF

 IF(wallDistLeft => 2 AND wallDistRight => 2 ) THEN
 GOSUB move_f
 ENDIF

 IF(wallDistLeft <=2 OR wallDistRight <=2 ) THEN

  DO WHILE wallDistRight < 5 AND wallDistLeft <5
   GOSUB readSonarRight
   GOSUB readSonarleft
```

```
   GOSUB Move_L
  LOOP

 ENDIF
 GOSUB move_f

RETURN
'================================================================================
'================================================================================
'search by following the left wall
hugLeftWallSearch:


 IF(wallDistRight <=3 ) THEN
   GOSUB move_R
 ENDIF

 IF(wallDistRight >=7 ) THEN
   GOSUB move_L
 ENDIF

 IF(wallDistRight => 2 AND wallDistLeft => 2 ) THEN
 GOSUB move_F
 ENDIF

 IF(wallDistRight <=2 OR wallDistLeft <=2 ) THEN

  DO WHILE wallDistLeft < 5 AND wallDistRight <5
   GOSUB readSonarRight
   GOSUB readSonarleft
   GOSUB Move_R
  LOOP

 ENDIF
 GOSUB move_F


RETURN
'================================================================================
'================================================================================
'look For food
lookForFood:
 GOSUB readTopFoodSensor

 'did the robot see any food(detect the light)
 IF(lightSensor <= 30 ) THEN

  foodCounter = 0

  folowLight:
   GOSUB readSonarRight
   GOSUB readSonarLeft
   GOSUB readTopFoodSensor
```

```
  'params
  genVarX = 1 'times TO sweep R AND L
  genVarY = 1 'passed or failed to find the light

  'food source found let the other robots know how I got here
  IF (lightSensor <=1 ) THEN
   GOSUB SendMessage
   END
  ENDIF

  IF lightSensor > 30 THEN
   'lost the light try to find it again
   GOSUB findLight:
  ELSE
    'follow the light
    GOSUB motionFindLightFix
  ENDIF

  'keep tack of # times failed to find the light
  IF(genVarY = 0) THEN
   foodCounter = foodCounter +1
  ELSE
   foodCounter = 0
  ENDIF

  'failed to find the light 4 times go back to other search methods
  IF foodCounter > 4 THEN
   RETURN
  ENDIF

  GOTO folowLight

 ENDIF
RETURN
'================================================================================
'================================================================================
'determin if last move was successful(is robot stuck?)
evaluateLastMove:

 'did i rotate more then 8 times without going forward?
 IF (rotateTrys > 8) THEN
  rotateTrys = 0

    GOSUB readSonarRight:
    GOSUB readSonarLeft:

    'stuck and saw right wall last
    IF(rotateRight > rotateLeft) THEN
     IF(wallDistRight < 5) THEN
      GOSUB findNewHeadingRight
     ENDIF

     'stuck and saw left wall last
     ELSE
```

```
    IF(wallDistLeft < 5) THEN
     GOSUB findNewHeadingLeft
    ENDIF
    ENDIF
 ENDIF


RETURN
'================================================================================
'================================================================================
'stuck on left side rotate and try to find new direction on left
findNewHeadingLeft:

 dir = 0
 'move right and look for the direction where the distance
 'of a wall is the furthest from the right ping sensor
 FOR counter = 1 TO 8
  GOSUB Move_R

  GOSUB readSonarRight
  IF dir < wallDistRight THEN
   dir = wallDistRight
   ndir = index
  ENDIF
 NEXT

 'the area has been scouted the new direction is to rotate (8 - ndir) times from current position
 FOR counter = 2 TO (8 - ndir)
  GOSUB Move_L
  GOSUB readSonarRight
  IF(dir < wallDistRight-10) OR (dir > wallDistRight+10) THEN RETURN
 NEXT

RETURN
'================================================================================
'================================================================================
'stuck on right side rotate and try to find new direction on right
findNewHeadingRight:


 dir = 0

 'move left and look for the direction where the distance
 'of a wall is the furthest from the left ping sensor
 FOR counter = 0 TO 8

  GOSUB Move_L
  GOSUB readSonarLeft
  IF dir < wallDistLeft THEN
   dir = wallDistLeft
   ndir = index
  ENDIF
 NEXT
```

```
 'the area has been scouted the new direction is to rotate (8 - ndir) times from current position
 FOR counter = 2 TO (8 - ndir)
  GOSUB Move_R
  GOSUB readSonarLeft
  IF(dir < wallDistLeft-10) OR (dir > wallDistLeft+10) THEN RETURN
 NEXT

RETURN
'================================================================================
'================================================================================
'used to determin if progress is being made in the travel
evaluateGround:

 'keep track of the ground color on last evaluation
 preGroundChange = groundChange

 'prevent false reading on ground color
 '(COUNT times I see a single color before deciding the reading is true)
 'does bottom light senesor see a dark color?
 IF(lightSensor > 320 ) THEN
  'count times I see the dark ground and reset the whiteGroundCounter
  blackGroundCounter = blackGroundCounter + 1
  whiteGroundCounter = 0
 ENDIF

 'does bottom light senesor see a light color?
 IF(lightSensor <= 320 ) THEN
  'count times I see the light ground and reset the blackGroundCounter
  whiteGroundCounter = whiteGroundCounter + 1
  blackGroundCounter = 0
 ENDIF

 'I have read the ground color is black 15 times in a row[the ground color is now black]
 IF(blackGroundCounter > 15) THEN
  groundChange = 1
 ENDIF

 'I have read the ground color is white 15 times in a row[the ground color is now white]
 IF(whiteGroundCounter > 15) THEN
  groundChange = 0
 ENDIF

 'has the ground color changed from the last iteration?
 IF preGroundChange <> groundChange THEN
  'reset the globalCounter and incrament the map memory pointer
  globalCounter = 0
  memPtr = memPtr +1
 ENDIF

RETURN
'================================================================================
'================================================================================
'rotates left and right to try to find the light again
findLight:
```

```
 '1=(true)-found the light
 genVarY = 1

 'try right side for light(move right genVarX times)
 FOR counter = 0 TO genVarX

  'check TO see IF I see a BRIGHT light after rotating right)
  GOSUB Move_R
  GOSUB readTopFoodSensor
  IF lightSensor < 30 THEN
   'yes see the light now(returns with genVarY = 1)
   RETURN
  ENDIF
 NEXT

 'did not find on last try now try left side FOR light
 FOR counter = 0 TO 3*genVarX

  'check TO see IF I see a BRIGHT light after rotating left
  GOSUB Move_L
  GOSUB readTopFoodSensor
  IF lightSensor < 30 THEN
   'yes see the light now(returns with genVarY = 1)
   RETURN
  ENDIF
 NEXT

 'failed to find the light (returns with genVarY = 0)
 'rotate robot to previous dierection
 genVarY = 0
 FOR counter = 0 TO genVarX
  GOSUB Move_R
 NEXT

RETURN

'================================================================================
'================================================================================
'keep from running into walls when following light
motionFindLightFix:
GOSUB forwardSearch
RETURN
'================================================================================
'================================================================================

'low level operations
'*******************************************************************************
' basic operations: send/receve, move f/l/r/b,
'pingSensor R/L, grnd Censor T/B, memory read/write'
'================================================================================
'================================================================================
readSonarRight:
```

```
 PULSOUT 5, 5 ' activate sensor
 PULSIN 5, 1, wallDistRight
 wallDistRight = wallDistRight /74
 'DEBUG "distL ", DEC5 wallDistRight, CR
 RETURN
'====================================================================================
'====================================================================================
readSonarLeft:

 PULSOUT 6, 5 ' activate sensor
 PULSIN 6, 1, wallDistLeft
 wallDistLeft = wallDistLeft /74
  '   DEBUG "---distR ", DEC5 wallDistLeft, CR

 RETURN
'====================================================================================
'====================================================================================
readTopFoodSensor:
 HIGH 7
 PAUSE 2
 RCTIME 7,1, lightSensor
 'DEBUG "t7= ", DEC5 lightSensor, CR
 RETURN
'====================================================================================
'====================================================================================
readGroundSensor:
 HIGH 4
 PAUSE 2
 RCTIME 4,1, lightSensor
 'DEBUG "tl= ", DEC5 lightSensor, CR
 RETURN
'====================================================================================
'====================================================================================
Move_F:
 FOR index = 1 TO 7
   PULSOUT 12, 700
   PULSOUT 13, 800
 NEXT
RETURN
'====================================================================================
'====================================================================================
Move_B:
 FOR index = 1 TO 14
   PULSOUT 12, 800
   PULSOUT 13, 700
 NEXT
RETURN
'====================================================================================
'====================================================================================
Move_R:
 FOR index = 1 TO 7
   PULSOUT 12, 765
   PULSOUT 13, 765
   PAUSE 20
```

```
 NEXT
RETURN
'================================================================================
'================================================================================
Move_L:
 FOR index = 1 TO 7
   PULSOUT 12, 725
   PULSOUT 13, 725
   PAUSE 20
 NEXT
RETURN
'================================================================================
'================================================================================
SendMessage:
 'This function reads the path data from memory, calculates the checksum for each value in the data path
 ' AND constructs the packets AND calls the transmit routine

 DCount = -1      'Initialize Data Count
 Packet = Packet + 1  'Increment packet number (1-15)
 Packet = Packet MIN 1
 Checksum = Packet   'initialize checksum to packet number

GetNext:
 DCount = DCount + 1
 Checksum = Checksum ^ Value(DCount)    'Calculate current checksum
 IF DCount => 5 THEN MessageDone      'If max data size, send message now
 GOTO GetNext               'Get next data dirArray

MessageDone:
 'send out the map
 SEROUT Tx\TxFlow,N9600,[STR Value\DCount+1]
GOTO MessageDone
 DEBUG "----sent this->", STR Value\DCount+1 ,CR

RETURN
'================================================================================
'================================================================================
'the receved msg
waitForInstructions:

 DEBUG "list___"
 'WAIT 4 seconds before trying again
 SERIN Rx, N9600, 4000, main2,[STR Value\6]
 DEBUG "----rec this->", STR Value

 'receved the map now follow it
 isScout= 1
 GOSUB move_b
 GOTO Main


 main2:
 'got nothing try again FOR msg
```