Developing a Swarm of Independent Cooperating Robots
Project Report: Spring 2013 CS Senior Seminar

Trevor Griswold
tgriswol@willamette.edu

Abstract

In my project, I developed an obstacle avoiding robot, and created a simulation of a swarm of independent "robots" which cooperated to find a goal hidden in the world. To accomplish this project, I analyzed the possible microelectronic components that could have been used to create the robots, and came up with a design that was sufficient to accomplish the desired task. I then assembled the robots, and programmed them to avoid obstacles. Later I created the simulation, which mimicked the robot's hardware, and programmed the simulated robots to find a goal hidden in the world, with the option of them either working cooperatively as a swarm, or independently.

Keywords Swarm Intelligence, Swarm Robotics, Raspberry Pi

1 Introduction

The goal for the research project was to explore a topic which I didn't have any previous experience with, in order to broaden my knowledge and experience new subject areas. For this reason, I decided to explore a field of computer science known as swarm intelligence and its relatively new counterpart in electrical engineering: swarm robotics.

2 Background

Swarm Intelligence (SI) and Swarm Robotics (SR) both have a common goal: to solve problems which are either not feasible or difficult for single autonomous agent systems. Both SI and SR use the concept of a multi agent swarm, a collection of autonomous and often simple agents, to solve specific physical or computational problems. Some examples of existing applications of SI or SR are: modeling a population of independent organisms, predicting the flow of a group of people through a specific location, pathfinding[4], and many military applications such as surveillance and patrol and mobile target tracking[1]. Although the concepts of SI was first introduced more than twenty years ago by Gerardo Beni and Jing Wang at conference in 1989[6] (and published in 1993[5]), the fields of SI and SR are still very active. This is due to recent advances in technology (allowing for fast parallel processing and a decrease in the cost of microelectronic components) which have made it possible for researchers, students, and hobbyists to design and create physical (SR) or simulated (SI) swarms, unlike in the past

where it was very cost-prohibitive.

2.1 Swarm Intelligence

Swarm intelligence (SI) has been defined as “any attempt to design algorithms or distributed problem-solving devices inspired by the collective behaviour of social insect colonies and other animal societies”[3]. Every swarm intelligence has one element in common: it uses a swarm of independent processes to solve a specific problem. One main benefit of using swarm intelligence algorithms is that they are often easily scalable; the algorithm is able to run on a supercomputer with hundreds of cores or a common desktop machine by utilizing parallel processing. There are many different swarm intelligence algorithms that have been developed in the past 20 years, but the two of the most commonly used ones are Ant colony optimization (ACO), and Particle swarm optimization (PSO).

Ant colony optimization (ACO) is an algorithm which was inspired by the foraging behavior of ants. One main component of this algorithm is how the ants use pheromones to mark paths and/or indicate the “attractiveness” of something to other ants. There are many variations of this algorithm, but in general all of them involve marking possible paths between nodes, and then strengthening (changing a measure of) the “best” (from what is known at the time) paths or weakening the worst ones[2]. This algorithm is often used in applications where you are searching for, or collecting and organizing data.

Particle swarm optimization (PSO) is an algorithm in swarm intelligence which was inspired by the foraging behavior of flocks of birds and schools of fish. The main idea of this algorithm is that while the flock flies through an area (a search space), each individual bird is attracted by its own best position, as well as the best position of its neighbors. This algorithm (as the name implies) is often used in optimization problems, where you can represent the data in some n-dimensional space. As with ACO, there are a few different variations on the PSO algorithm, but the general idea is that there are two random weights ranging from 0 to a “learning rate” which affects the degree to which an individual moves toward its own best next position, and toward the global best next position[2]. This algorithm is often used where it is either impossible or unfeasible to calculate an exact solution, but an approximate solution is desired.

2.2 Swarm Robotics

Swarm robotics (SR) is a field which has changed greatly since the development of the concept of SI 20 years ago. SR has been defined as “the study of how large numbers of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment”.[3] The exact methods of swarm robotics vary greatly between applications, but they all use ideas from SI in conjunction with electronics to create

autonomous robots which work together to accomplish a certain task. One difference between SI and SR, is that in SI the agents are often identical, but in SR that isn't always the case. In some applications of SR there are many different types of agents, each with a specific specialization, although there are often multiples of each type. Some common applications of SR are those which involve searching for a specific target, surveying, or mapping an area. SR is most commonly used in military applications or other dangerous environments, where it would be dangerous for a human, or where there is no constant contact with a human controller.

One application of swarm robotics is the Swarm-bots project[7]. The goal of the swarm-bots project was to create multiple s-bots which could combine and work together as one entity, a swarm-bot. Each s-bot was identical, containing multiple sensors such as light, sound, infrared, humidity, accelerometer, force sensors, a camera and more. The s-bots also have a set of LEDs, speakers, rigid and flexible arms, and a drive mechanism composed of both tracks and wheels. The s-bots run linux and have a wi-fi antenna to communicate (as well as other sensors / outputs). In this project, many different algorithms were tested for the control and coordination of the robots, including evolutionary algorithms and neural networks. In the end, the bots were able to perform simple searching tasks on their own, and were able to combine and work together to perform tasks that would be impossible alone, such as crossing large gaps or moving large objects.

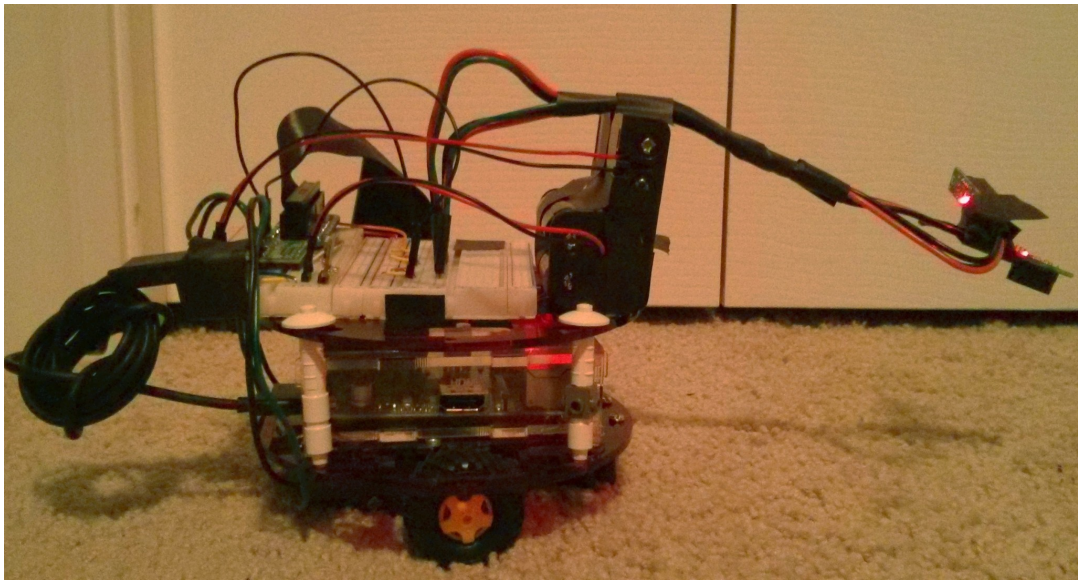
Another application is the I-Swarm project[8]. Unlike the Swarm-bots project, the I-Swarm project involves tiny robots - about as small as a three millimeter cube. Rather than running a complicated control system, these robots use very simple logic along with their sensors to accomplish their goal. These robots have a locomotion unit consisting of three legs, as well as various sensors. This project still is very much a work in progress, but it serves as a good example that the size of the robot in a swarm can vary greatly.

There are many other applications of swarm robotics, but these two show how the hardware, logic, size of the swarm robots can vary greatly, and are greatly influenced by the tasks they are designed to accomplish.

3 Project Implementation

The goal of my project was to develop a swarm of independently operating robots, and program them to accomplish a specific task. Initially, I wanted to do the entire project in hardware, creating two identical robots. However, I ran into many issues which delayed my project, such as not having the right hardware components, and having to order and wait for them to be shipped. Eventually, I realized that I would not have enough time to create a functioning swarm, so I decide to split my project into two parts: a proof-of-concept robot which could drive around and avoid obstacles, and a software simulation of robots acting cooperatively as a swarm.

3.1 Hardware Design



The first stage of my project was decide on a task for the robots to accomplish, and then research and order components which would be sufficient for the task. I decided that the task of searching an area for a goal, and getting all the robots to gather around it, was a task which could demonstrate the effectiveness of a swarm. This task wouldn't require any specialized hardware (such as an arm to grab an object), so I set out researching components to build a standard obstacle avoiding robot, with the intention of adding a webcam later to recognize the goal.

The first component that I researched was the microcontroller, the "brain" of the robot which is responsible for processing and sending out electrical signal to control all the other components. Usually, for robotics applications similar to this, an Arduino microcontroller would be used. Arduino is designed to make interfacing with electrical components easier, but it forces you to use Processing or C++ to program it. I decided that for my project I'd would use a Raspberry Pi, a "credit-card-sized" single-board computer to act as the microcontroller for my robot. The Pi (I'll shorten Raspberry Pi to Pi from now on) has many benefits over the Arduino, but also a few deficits. Firstly, the Pi is more than a microcontroller, it is a complete computer: it contains all the components and input/output ports to function as a standalone computer (along with a keyboard, mouse, and display). The Pi also has much more powerful hardware than the Arduino, with 512MB of ram and a 700 MHz processor, allowing it to perform more complicated computations. Lastly, the Pi allowed me to program the robot in practically any language I wanted, because it is designed to run a distribution of Linux called Raspbian (based on the Debian distribution). The deficit of using the Pi for my robots was that it required a lot more setup and learning than an Arduino would have. The Pi has GPIO pins which can be used for controlling external components, but some are also used for other functions inside the Pi, so finding

out which specific pins to use was tedious. Also, using the Pi required me to configure the Raspbian distribution and download the necessary Python packages.

The next component of the robot which was chosen was the chassis. When I was designing the robot, I wanted to it be able to spin in place (without moving forward and backward) and the easiest way to do this was to have a chassis designed to only use 2 independently drivable wheels. However, with only two wheels balance becomes an issue, and thus the robot needs another component to help balance the weight: usually a ball bearing which can freely spin. I eventually found a chassis kit from Pololu Robotics and Electronics which suited my needs perfectly. The kit included a twin-motor gearbox, a single housing that contains two motors for driving two separate wheels, a ball caster, a freely-spinning ball bearing in a housing to be used for balance, and rubber tires. The chassis kit was used along with a circular Pololu robot chassis plate to form the base of the robot.

After deciding on a microcontroller, chassis, and motors, the only main component left was the sensors which the robot would used to gather information about its environment. I decided to use distance sensors to detect objects around the robot. Since the robot was able to spin in place without movement, I decided that only two sensors were needed: one to detect obstacles directly in front of the robot, and one to detect floor below the robot, in order to tell if it was about to drive off the edge of a table or other surface. Since the objects that the robot would be detecting were relatively close to the it, I decided to use digital distance sensors which could sense objects between 2 and 10 cm away.

Although I had chosen all the main components for the robots, there were still other small components that were necessary to support the larger ones. To power the Pi, I used a USB-A port and a USB-A to Micro USB-B cable, a 5V step-up/step down voltage regulator, and a 2 AA battery holder. The battery holder was connected to the voltage regulator, and sent through the USB port and cable, in order to provide the Pi with a steady source of battery power. I also used a PiBox enclosure to contain the Pi, a Pi Cobbler breakout kit which connects the GPIO pins from the Pi to the breadboard, and an 8GB SD card to store the Rasbian operating system and code to run Pi. To control the motors I used a dual motor driver carrier, which converts the signals from the Pi to the proper voltage to power the motors, and a 4-AA battery holder to supply the power. The only other components used are 2 400 point breadboards and wires to connect all the components together.

Included with this paper is a listing of all the parts used on the robots, links where they were purchased, and a diagram showing how all the components were connected.

3.2 Robot Programing

One of the benefits of using a Raspberry Pi as the microcontroller for the robots is that they have the flexibility to run most programming languages (anything that you can run

in Linux). I decided to use Python to program the robots for a couple of reasons. First, Python has an interpreter that let me quickly test commands and verify that the hardware components were working properly without having to save and compile my code. A second reason is that a package was created called RPi.GPIO (https://pypi.python.org/pypi/RPi.GPIO) which allowed me to easily interact with the GPIO pins on the Raspberry Pi through Python commands, in order to control the hardware components.

The code that I wrote to control the obstacle-avoiding robots was actually pretty simple. First, I performed the necessary setup for interacting with the GPIO pins, and created functions which moved each wheel forward, backward and stop. After that, I added functions to perform all the available movements for the robot: moving forward, moving backward, stopping, spinning in place to the left and right, and pivoting forward and back on each wheel. Finally, I added functions which detected if an object was in front of and below the robot. The code that drives the robot simply checks to see if there is floor below the robot, and if there isn't, it stops, moves backwards until it sees the floor again, and spins to the right. If there is an object in front of the robot, it stops, and spins right until it no longer detects that object. If the floor is detected, and no object is detected in front of the robot, it drives forward. The full Python code for the robot is listed below.

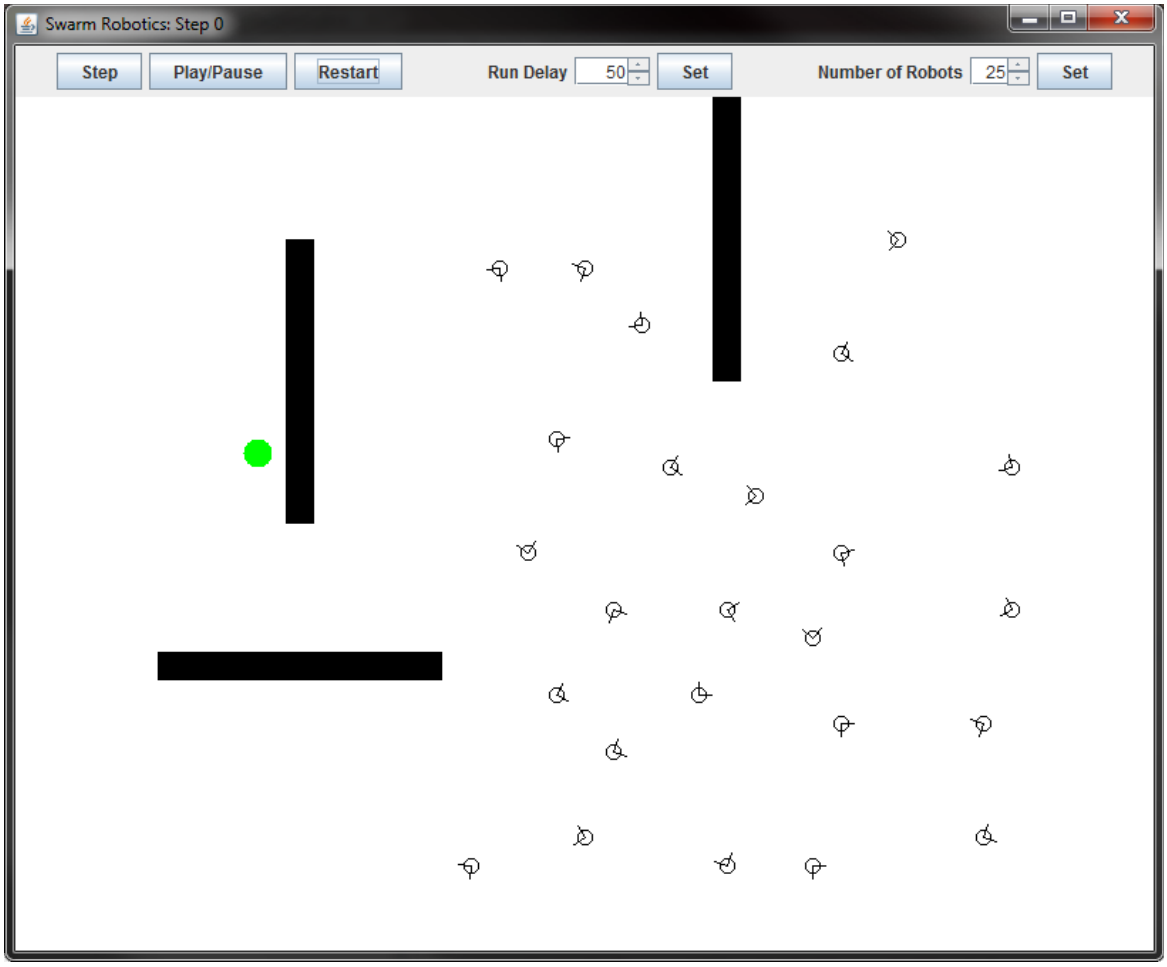
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD);
GPIO.setup(16, GPIO.OUT);
GPIO.setup(18, GPIO.OUT);
GPIO.setup(24, GPIO.OUT);
GPIO.setup(26, GPIO.OUT);
GPIO.setup(15, GPIO.IN);
GPIO.setup(13, GPIO.IN);
GPIO.setup(12, GPIO.IN);

if (GPIO.input(12) == 1):
    GPIO.cleanup();
    exit();

def rightWheelForward():
```

```
    GPIO.output(24, False);
    GPIO.output(26, True);

def rightWheelBackward():
    GPIO.output(24, True);
    GPIO.output(26, False);

def rightWheelStop():
    GPIO.output(24, False);
    GPIO.output(26, False);

def leftWheelForward():
    GPIO.output(16, True);
    GPIO.output(18, False);

def leftWheelBackward():
    GPIO.output(16, False);
    GPIO.output(18, True);

def leftWheelStop():
    GPIO.output(16, False);
    GPIO.output(18, False);

def goForward():
    leftWheelForward();
    rightWheelForward();

def goBackward():
    leftWheelBackward();
    rightWheelBackward();

def stop():
    leftWheelStop();
    rightWheelStop();
```

```
def spinRight():
    leftWheelForward();
    rightWheelBackward();

def spinLeft():
    leftWheelBackward();
    rightWheelForward();

def pivotForwardRight():
    leftWheelForward();
    rightWheelStop();

def pivotBackwardRight():
    leftWheelBackward();
    rightWheelStop();

def pivotForwardLeft():
    leftWheelStop();
    rightWheelForward();

def pivotBackwardLeft():
    leftWheelStop();
    rightWheelBackward();

def waitForNextCommand():
    time.sleep(3);

def objectInFront():
    return (GPIO.input(15) == 0);

def objectBelow():
    return (GPIO.input(13) == 0);
```

```
backingUp = False;
while (True):
    if (not objectBelow()):
      stop();
      time.sleep(1);
      while (not objectBelow()):
          goBackward();
      stop();
      time.sleep(1);
      spinRight();
      time.sleep(1);
      stop();
      time.sleep(1);
    elif (objectInFront()):
      stop();
      time.sleep(1);
      while (objectInFront()):
          spinRight();
      stop();
      time.sleep(1);
    else:
      goForward()
    time.sleep(0.1);


GPIO.cleanup();
```

3.3 Software Simulation



The goal of my software simulation was to create a swarm of “robots” which would cooperate to find a goal hidden in the world, completing what I had originally intended to do in hardware. I created a graphical interface with user controls to play, pause, and step through the simulation, and set the number of robots to randomly place in the world.

The simulation was based on a 2D world containing SimObjects. SimObjects had two main components: a location in the world, and an act() function which defined what the SimObject would do each timestep in the simulation. SimObject had 3 main subclasses, Wall (an obstacle), Goal (the object the robots were searching for in the world), and SwarmRobot (the software simulation of my robots).

The SwarmRobot had all the same base capabilities as my hardware robots: it could move forward and backward, spin in place, and detect objects in front of it. I also added the ability for the robot to detect far away robots and goals in front of it (simulating the webcam I intended to add to the robots). In order to allow the robots to communicate, I added a way for them to produce “sound” (a signal broadcasted across the world) and to “hear” (detect the signal) other robots, and a way to specify if they were successful (simulating an LED or other visual notification). Using these basic specifications, I was able to program the robots to find the goal in the grid, while avoiding bumping into other

robots and obstacles.

In order for a robot to be successful, it needed to be adjacent to the goal (with the goal directly in front of it in sensor view) or adjacent to another successful robot. To find the goal, the robots wandered the grid avoiding obstacles, occasionally stopping to spin in a circle and scan for the goal. If the goal was detected, the robot would stop spinning and drive forward until it was adjacent to it. After successfully finding the goal, the robot would notify the others by producing a sound and signalling it was successful. If a sound was heard, the robots would stop and spin in a circle, scanning the world for a successful robot, and moving adjacent to it if detected. If no goal or successful robot were detected, the robots would continue wandering the grid avoiding obstacles, stopping to scan again occasionally.

Included with this paper is the full source code for the software simulation, a class diagram, and a executable jar file to run the simulation.

4 Lessons Learned

Throughout the duration of this project, I learned many lessons. One main lesson was that my estimations for how long it would take to get a working obstacle-avoiding robot was extremely inaccurate. I should have realized that complications with not having the right parts, or debugging hardware issues could greatly delay the project. I don't regret doing the hardware simulation though, as I learned a lot about assembling and programming for microelectronics, and how to solder.

5 Future Work

If I have more time to work on this project in the future, I'd like to finish the hardware component of my project. Although I ran out of time, and decided to move to a software simulation, I think that I would still be able to accomplish my original goal in hardware. The main work that was left was to add in a webcam to recognize the goal (probably a glyph on the floor) and add a way for the robots to communicate with each other.

I'd also like to add some more complicated tasks for the robots to complete in my software simulation. Being a simulation, I don't have to worry about expensive or specialized hardware that would be required to build the robots, so the possible tasks are nearly endless.

6 Bibliography

1. Sauter, John A., Robert Matthews, H. Van Dyke Parunak, and Sven A. Brueckner (2005), Performance of digital pheromones for swarming vehicle control." In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, 903-910, ACM, New York, NY, USA, URL http://doi.acm.org/10.1145/1082473.1082610

2. Sudholt, Dirk (2012), “Theory of swarm intelligence." *In Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, GECCO Companion '12, 1215-1238, ACM, New York, NY, USA, URL http://doi.acm.org/10.1145/2330784.2330938
3. Sharkey, A. J. (2007). Swarm robotics and minimalism. *Connection Science,19,* 3, 245-260.
4. Szymanski, Breitling, Seyfried, Wörn (2006). “Distributed Shortest-Path Finding by a Micro-robot Swarm.” *Ant Colony Optimization and Swarm Intelligence,* Springer Berlin / Heidelberg, 404-411, URL http://dx.doi.org/10.1007/11839088_39
5. Beni, Gerardo, and Jing Wang. "Swarm intelligence in cellular robotic systems." Robots and Biological Systems: Towards a New Bionics? (1993): 703-712.
6. Beni, G., Wang, J. Swarm Intelligence in Cellular Robotic Systems, Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30 (1989)
7. Dorigo, Marco, et al. "The swarm-bots project." Swarm Robotics (2005): 31-44.
8. Seyfried, Jörg, et al. "The I-SWARM project: Intelligent small world autonomous robots for micro-manipulation." Swarm Robotics (2005): 70-83.