Robotic Swarm Networks: A Low-Cost Alternative Robotic Solution for Surveillance, Documentation, and Resource Gathering

Ryan Gadsby, May 2010

Boston College Computer Science Departmen

INTRODUCTION

I. The growing role of robots in human society

Humanity has come to rely on robots in many aspects of society. In the industrial sector, we have intricate, precise, robotic installations designed to either manufacture or inspect various goods, such as automobiles. In the commercial sector, floor-crawling rovers vacuum floors, freely available to anyone with the inclination and disposable income to do so. On the surface of Mars, the twin observational rovers Spirit and Opportunity scour the dusty red landscape, transferring

job too costly, dangerous, and time-consuming for humans to perform is one of the foremost applications of robotic technology. (Mars Exploration Rover project)

Robot-assisted or completely unmanned surgery has become a way for doctors to reliably perform different operations with a degree of precision impossible to duplicate by human hands alone. Robots can delicately perform minimally invasive tasks that human hands cannot, simply because of the smaller size and locations of interaction that can be handled by machine. Robotic

their sensory data with each other. (K.A.Hawick, H.A.James, J.E.Story and R.G.Shepherd, 2007)

Applications for swarm robotics present themselves at many levels of technology. Where primitive swarm members are appropriate for doing a basic physical task - such as mining, surveying, or foraging – swarms can be a preferable alternative to using human labor due to health risks or general tedium. Where miniaturization is a factor, swarm robotics can be applied to nanotechnology or micromachinery to handle distributed sensory tasks in the human body. By decentralizing intelligence we allow for more primitive swarm members, instead relying on parallel computation to perform tasks using macroscopic control over the entire swarm. As a whole, however, swarm robotics has yet to emerge outside of the research sector, and there are no current commercial, military, or recreational implementations of it. (

swarm member (or a subset of the swarm) 'figures out' how to accomplish something, the entire swarm is able to capitalize upon this knowledge. This combination of simulating social insect behavior with artificial evolution allows SYMBION swarms to perform tasks such as linking up

they are required to encounter, robots tend to be rather costly to engineer in terms of time and, subsequently, currency. As a result of these costs, it often becomes difficult to justify subjecting these machines to harsh environments that may damage them. While addressable, these concerns must weigh heavily in the mind of any engineer before making any attempt to approach a problem from a robotic point of view.

Power consumption is one of the largest barriers to effectively applying robotic technology to humanity's problems. Batteries simply do not last long enough to power a robot's motors and sensors for many tasks, especially those which last a long time such as exploration. A common solution is to use solar panels, such as on the Mars rovers Spirit and Opportunity, but solar panels don't generate enough energy to keep robots functioning constantly, and certain environmental factors can render them useless altogether. Robots need access either to a constant energy source or intermittent contact with an energy source in order to maintain function as

produced (after great care is taken to select the right array of motors and sensors to accomplish

In terms of power consumption, robotic swarms tend to have many advantages when compared to larger, more complicated robots. Their smaller, more primitive motors, sensors, and CPUs simply use less energy, although they are also typically equipped with smaller batteries. However, since the swarm is composed of many smaller units, allowing one to rest and recharge via solar panels or some sort of energy source allows the swarm to continue its work relatively unimpeded. The collective capacity to do work before recharging in a swarm will be greater than that done by a single robot, and work will always still be done even as individual members have entered an inactive mode in order to recharge their batteries. (Schmickl and Crailsheim, 2006)

Imprecise movement is still a problem for robotic swarms, but swarms are better equipped to meliorate the impact of the issue than singular robots. By being able to pinpoint or even estimate each robot's position relative to its brothers, we can construct a much more reliable model of a given robotic operation than relying on one robot's impressions of its surroundings. Multiple cameras can be used to create a stereoptic analysis of a system, and the collective estimations therein can be averaged to provide a realistic if imperfect snapshot of the environment the swarm is deployed to. This way, we can have a proficient system of reckoning without having to rely on a costly absolute system such as GPS or radar. (Moeslinger, Schmickl and Crailsheim, 2009)

The emphasis on using many smaller, less sophisticated robots to perform a task in swarm robotics is often a more cost-effective solution than having one or two intricate machines. Swarm members are typically not designed with durability or multi-tasking in mind, and can therefore be equipped with inexpensive sensors, motors, and CPUs. Some sensors can become very expensive as higher qualities are necessary, such as cameras and infrared detectors, but we need not equip every member of the swarm similarly. Indeed, using the sensor network of the entire swarm, it is

“hive-mind” of the swarm. (Schmickl and Crailsheim, 2006)

Because of the low cost of individual machines, members of a swarm can be viewed as expendable, allowing them to perform tasks that one might think twice about using a single expensive robot for. While we might be hesitant to send a multi-million dollar automated submersible to examine undersea volcanic vents, a member of a swarm can perform this task ably with a cheap camera and whatever relevant sensors while transmitting any data back to his brothers before getting destroyed. This allows us to completely automate certain tasks that are normally left to humans themselves – or at least humans remotely controlling robots – without worry.

Aside from providing methods to overcome the main issues with mobile robotic technology, swarm robotics gives us ways to perform tasks that would simply be infeasible for normal robots. Robots tend to move sluggishly, and where time is an issue their employment is often unattractive. One or two robots, no matter how sophisticated, are not an optimal way to search for survivors in collapsed mine, to sniff out explosive devices in a sprawling urban landscape, or to alert soldiers of intruders along a wide perimeter. A swarm, made of many subunits, can cover ground much more quickly and survey a greater area afuch1m 17 (i) -5 o8m[[6sfor

number of variables that could cause something to go horribly wrong increase greatly, and must be accounted for to have any reasonable application of swarm technology. Using more primitive sensors and motors can also degrade the quality of the work done by the swarm. They are not suitable for undertaking tasks that require very specific observations or sensory input, like exhaustively exploring an entire area and surveying it to produce an accurate map.

With a single robot, one must only account for one set of sensors, one CPU, and one body of code. This is sadly not so in a swarm. By having multiple robots equipped with a variety of sensors on a robot-by-robot basis, there are several different protocols that must be accounted for

Because the physical hardware swarm robots are equipped with tend not to be terribly sophisticated, more problems arise. The lack of high-quality sensors and motors can make swarms unsuitable for tasks that require detailed sensory feedback or delicate movement. For example, equipping every robot in the swarm with a high resolution, telescopic lens camera defeats the purpose of using a swarm as a cost-effective alternative to a single sophisticated robot. They are better suited to tasks that merely require imprecise calculation: searching for the general location of a resource, or estimating the depth of a cave, for example.

METHODOLOGY

1. The NXT Brick from LEGO Mindstorms

All swarm implementation was done in Java using ten Lego NXT “bricks” equipped with leJOS firmware. These relatively inexpensive computers come in packages with modular, interchangeable sensors and motors that made it easy to test different configurations. Moreover, a third-party camera designed for the NXT was also used. Ten skeletal robots were assembled, and several different network models were tested.

The NXT Brick itself possesses an ARM7 microprocessor clocked at the relatively unimpressive speed of 46 megahertz and a meager cache of 32 kilobytes. While this is more than enough computing power to handle basic motor, sensor, and communication routines, it severely limits any sort of complicated data processing that can be done by the swarm without relying on an external source. Luckily, the distributed nature of the swarm helps alleviate this drawback, as computational tasks can be relegated to inactive members.

NXT Bricks can currently only interface with one type of motor, which is included in the Mindstorms kit. It's an electric servo motor with a high degree of motion accuracy thanks to its on-board tachometers: accurate to within one degree of specificity. Any sort of locomotive task – either moving the robot itself or manipulating an object – was undertaken using one or more of these motors.

The Mindstorms kit comes with a variety of sensors, and many third-party sensors have been developed for the NXT brick. Included are:

- A touch sensor that simply gives binary feedback for touch or release.
- A light sensor that gives quantitative feedback of light intensity overall, or the

light intensity of certain colors.

- A sound sensor that gives quantitative levels of sound in either dB or dBA.
- An ultrasonic sensor that approximates the distance of an external object by emitting an infrared “ping”.
- An accelerometer that measures the rotational position of the robot.
- A compass sensor to tell the heading of the robot with regards to magnetic north.
- An RFID sensor that gives feedback to match a given RFID frequency.

Moreover, cameras have been developed via third parities to give the robot a way of visually interfacing with its environment. For this thesis, the NXTCam developed by <> was used for this purpose. The proprietary sensors used were limited to the ones included in the Mindstorms kit: the touch, light, sound, and ultrasonic sensors. This is because of the cost associated with extra sensors whose function was either limited for the scope of this thesis or could readily be duplicated with the other sensors.

The NXT brick has seven ports designed for 4-conductor cables to connect to: three for motors labeled A,B, and C, and four for sensors with numeric labels. To keep the solution of cost-effectiveness in mind, most of the bricks were equipped with only 2 motors and 2 or less sensors. Many members of the swarm were not equipped with sensors at all. This kept each member's functionality at bare minimal levels in order to demonstrate the power of the swarm as a whole.

The NXT bricks have important limitations that accentuate the problem of using more primitive hardware for a swarm as compared to a single robot. The most drastic limitation by far is the connection limit of the NXT's on-board Bluecore chip that handles Bluetooth communication. A given NXT brick may only maintain three connections at once, whether they are to other NXT bricks or other Bluetooth devices such as a GPS receiver or a laptop. As such,

swarm structures were inherently limited to hierarchical configurations rather than fully-connected graphs that a swarm would ideally possess. This could be worked around by deleting and reinstating connections at runtime, but the significant increase in running time wasn't justified when any data could be relayed through the hierarchy more quickly with the proper network implementation.

The other limitation was the memory capacity of the NXT brick. Each brick only has 256K of flash memory, more than half of which was taken up by the leJOS firmware. This limited not only the amount of Java classes that could be used in implementing the swarm's behavior, but also the amount of persistent image data that could be relayed back to an external source.

II. The leJOS custom firmware

The leJOS firmware is an open source project 2 (t) 7 (a) 12 107 -6 (m) 2 (07 -6 (a) 12 1)-8whtET BT 25

used up by the firmware itself – steps must be taken to conserve the remaining amount of memory carefully. As such, it often is a smarter solution to use predefined arrays instead of

these at runtime thanks to the leJOS's lack of command line interface. The top member of the swarm hierarchy maintains a list of all positions of members of the swarm, and the swarm is initialized via the masters' requests for every member's position. After this, the swarm begins to act autonomously, although behavior varies by swarm configuration.

The master maintains three separate connections to the mid-level members of the swarm,

Each CommandCenter has an object called MovementCenter that track's the robots facing and position and handles each robot's movement subroutines. It can be calibrated for any given surface, and uses a Cartesian grid system relative to the starting position of the master robot in the swarm. It can be told to go to a point, or to simply face a point. The amount of rotation and movement are handled by using simple Cartesian distance formulas to calculate the distance, and

could also be used to simply document an object or an environment from multiple perspectives for the sake of completeness or redundancy of information.

V. Swarm configuration Parker

The second sensor configuration for the swarm is called “Parker”. It works essentially as an inversion of the Triforce configuration. The root node is equipped with camera instead of the second command level, and the second command level function as investigators using the light and ultrasonic sensors. As in Triforce, the leaf robots are used to process data and perform simple mundane tasks, as they are sensorless. Most importantly, however, they are used to relay objective data to outside sources, alerting them to the position and condition of a given objective. Compared to Triforce, Parker's strength lies in the ability to cover ground to find potential objectives to document as quickly as possible. It is akin to a search-and-rescue or resource-locating swarm as opposed to the observational nature of the Triforce configuration.

VI. Swarm configuration Gaga

Gaga, the last swarm configuration, differs from Triforce and Parker in that it is designed to accomplish work by physically interacting with its environment. Here, the leaf robots are used

swarm configuration is best-suited to a wide-scale environmental task such as taking soil samples or agricultural foraging.

VII. The tasks

One task was chosen for each swarm configuration. The time taken to complete each task was recorded for both the corresponding swarm and a single bot equipped to do each task on its own. All of the "objectives" used in these tasks are simply red plastic cups placed at random locations in the environment that the swarm (or single robot) explores. Three timed trials for each swarm were taken.

The task for Triforce involves documenting an objective from several perspectives. For the Triforce swarm, the root node is used to locate these objectives while the subcommanders simultaneously document it using cameras. For the analogous single robot, the robot must do all

The task for Gaga is similar to Parker's, but instead physical work must be performed on the objectives. To simulate these actions, any robots defined as workers spin in place for 30 seconds at the objective to demonstrate a physical task, such as drilling or gathering a sample. The singe robot must wait until this work is completed before it can continue searching for another objective. In all other aspects, at the root commander and subcommander level, the task is the same as that undertaken by the Parker swarm.

IV. Conclusions

The general trend with the NXT swarms was that they were consistently able to perform their given tasks significantly faster than single robots. However, this only applies when the tasks themselves were completed. Due to problems with pathing, reckoning, synchronization and communication, colliding swarm members inhibited the swarm's ability to actually finish the task at hand.

Parker, however, was unaffected by this as its immobile leaf robots and inherently divergent subcommander movement patterns prevented any collisions. Triforce and Gaga, due to their more active leaf robots, were more likely to experience a fatal crash that rendered their task not completable.

This problem could have been alleviated by more efficient pathing algorithms, or allowing the swarm to give a more thorough estimation of its current density at given locations as environmental features to avoid.

REFERENCES

- Mars Exploration Rover project, NASA/JPL document NSS ISDC 2001 27/05/2001

- Ahmed K; Khan MS; Vats A; Nagpal K; Priest O; Patel V; Vecht JA; Ashrafian H; et al. (Oct 2009). Int J Surg. 7:431-440

- K.A.Hawick, H.A.James, J.E.Story and R.G.Shepherd, *An Architecture for Swarm Robots* p2-3

- Waldner, Jean-Baptiste (2007). *Nanocomputers and Swarm Intelligence*. London: ISTE. pp. 242-p248

- Schmickl and Crailsheim, *A navigational algorithm for swarm robotics inspired by slime mold aggregation,* Second SAB 2006

- Sevan G. Ficici, Richard A. Watson, Jordan B. Pollack, *Embodied Evolution: A Response to Challenges in Evolutionary Robotics,* Eighth European Workshop on Learning Robots, 1999

- Jason Teo, *Darwin + Robots = Evolutionary Robotics: Challenges in Automatic Robot Synthesis ,*

Mobile Robot's Energy Consumption and Conservation Techniques , pp4-5, 2005

- Hakyoung Chung , Lauro Ojeda, and Johann Borenstein, *Sensor fusion for Mobile Robot Dead-reckoning With a Precision-calibrated Fiber Optic Gyroscope* , pp2-3, 2001 IEEE International Conference on Robotics and Automation, Seoul, Korea, May 21-26, pp. 3588-3593

- Christoph Moeslinger1 , Thomas Schmickl1 , and Karl Crailsheim1: *A Minimalist Flocking Algorithm for Swarm Robots, 2009*

```
	public void initialize() throws IOException{
		//botLocation.add(botID, new Point(thisBot.getX(),thisBot.getY()));

		//connect to parent
		addConnection(Bluetooth.waitForConnection());
		//connect to children
		for (int i = 1; i<3;i++){
			RemoteDevice rd = Bluetooth.getKnownDevice("Gadsby" +(i+botID));
			addConnection(Bluetooth.connect(rd));
		}
		//good to go
		ready = true;
	}
	public boolean isReady(){
		return ready;
	}

}
```

```
                this.waitForDocumentation();
            }
        }

        private void requestPositionUpdates() throws IOException{
            for (int i = 0;i<3;i++){
                try{
                    DataInputStream dis = in.get(i);
                    DataOutputStream dos = out.get(i);

                    dos.writeInt(CommandCenter.INITIALIZE_LOCATIONS);
                    dos.flush();
                    LCD.drawInt(i,i,i);
                    int response = dis.readInt();
                    LCD.drawInt(response,4,4);
                    if (response != CommandCenter.BUSY && response !=
CommandCenter.NO_CHANGE){
                        //get botID and point, add it to locations.
                        botLocation.add(dis.readInt(),new
Point(dis.readInt(),dis.readInt()));
                        dos.writeInt(CommandCenter.Tm /T -6 ( v)GIR){ a
```

```
		}
		return new Point(x/botLocation.size(),y/botLocation.size());
	}

	private boolean locationOccupied(int x, int y){
		for (Point p : botLocation){
			if (x == p.x || y == p.y)
				return true;
		}
		return false;
	}

	private void waitForDocumentation()throws IOException{
		while (true){
			for (int i = 0; i<3;i++){
				int c = in.get(i).readInt();
				if (c == CommandCenter.DOCUMENTED){
					LCD.drawString("Bot " + i + " finished",i,i);
				}
			}
			Button.waitForPress();
		}
	}
}
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;
import java.awt.Point;

public class TriforceSubcommand extends CommandCenter{
	private NXTCam cam;

	private boolean selfreported;
	private boolean child1reported;
	private boolean child2reported;

	public TriforceSubcommand(int id, MovementCenter bot){
		super(id,bot);
		cam = new NXTCam(SensorPort.S1);
		ready =false;
		selfreported = false;
		child1reported = false;
		child2reported = false;
	}

	public void executeCommand() throws IOException,InterruptedException{
		while (true){
			try{
				int command = in.get(0).readInt();

				if (command == CommandCenter.INITIALIZE_LOCATIONS)
					initializeLocations();
				if (command == CommandCenter.INVESTIGATE)
					investigate();
			}catch (IOException e){
			}
		}
	}

	private void initializeLocations() throws IOException{
		LCD.drawInt(0,0,0);
		if (!selfreported){
			out.get(0).writeInt(0);
			out.get(0).writeInt(botID);
			out.get(0).writeInt(thisBot.getX());
			out.get(0).writeInt(thisBot.getY());
			out.get(0).flush();
			selfreported=true;
```

```
}
else if (!child1reported){
        out.get(0).writeInt(0);
        out.get(0).writeInt(botID);
        out.get(0).writeInt(thisBot.getX());
        out.get(0).writeInt(thisBot.getY());
        out.get(0).flush();
        child2reported=true;
}
else if (!child2reported){
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;
import java.awt.Point;

public class TriforceLeaf extends CommandCenter{
	public TriforceLeaf(int id, MovementCenter bot){
		super(id,bot);
	}
	public void executeCommand() throws IOException,InterruptedException{
		while (true){
			try{
				int command = in.get(0).readInt();
				if (command == CommandCenter.INITIALIZE_LOCATIONS)
					initializeLocations();
				if (command == CommandCenter.INVESTIGATE)
					investigate();
			}catch (IOException e){
			}
		}
	}

	public void initialize() throws IOException{
		addConnection(Bluetooth.waitForConnection());
		ready = true;
	}

	private void initializeLocations() throws IOException{
			out.get(0).writeInt(0);
			out.get(0).writeInt(botID);
			out.get(0).writeInt(thisBot.getX());
			out.get(0).writeInt(thisBot.getY());
			out.get(0).flush();
	}









	private void investigate() throws IOException,InterruptedException{
```

```
                while (true){
                        try{
                                int x = in.get(0).readInt();
                                int y = in.get(0).readInt();

                                //go to location
                                thisBot.goToPoint(x,y);
                                out.get(0).writeInt(CommandCenter.DOCUMENTED);
                                out.get(0).flush();
                                break;
                        }catch(IOException e){
                                                //Thread.sleep(500);
                        }
                }
        }
}
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import java.util.*;

public class MovementCenter{
	public static final int LATERAL_CONSTANT = 1500;
	public static final double RADIAL_CONSTANT = .13;

	private int posX;
	private int posY;
	private float dirX;
	private float dirY;

	public static void main(String[] args) throws InterruptedException{
		MovementCenter thisBot = new MovementCenter(0,0,0f,1.0f);
		Thread.sleep(3000); //3 second delay
		thisBot.moveToFace(1,1);
		thisBot.moveToFace(-1,-1);
		thisBot.approachPoint(4,3);
		thisBot.approachPoint(0,0);
	}

	public MovementCenter(int px, int py, float dx, float dy){
		Motor.A.setPower(50);
		Motor.C.setPower(50);
		this.posX = px;
		this.posY = py;
		this.dirX = dx;
		this.dirY = dy;
	}


	public void move(float munits) throws InterruptedException{
		//positive is forward. Motors A and C are wheels.
		if (munits>0){
			Motor.A.forward();
			Motor.C.forward();
		}
		if (munits<0){
			Motor.A.backward();
			Motor.C.backward();
			munits = munits * -1;
		}

		Thread.sleep(Math.round(munits*LATERAL_CONSTANT));
		Motor.A.stop();
```

```
		Motor.C.stop();
		//forced wait to keep motors from bucking with constant movement
		Thread.sleep(200);
	}

	public void rotate(int degrees) throws InterruptedException{
		//positive is counterclockwise
		if (degrees>0){
			Motor.A.forward();
			Motor.C.backward();
		}
		if (degrees<0){
			Motor.A.backward();
			Motor.C.forward();
			degrees = degrees * -1;
		}
		Thread.sleep((long)Math.round(50*degrees*RADIAL_CONSTANT));
		Motor.A.stop();
		Motor.C.stop();
		Thread.sleep(200);
	}

	public void goToPoint(int x, int y) throws InterruptedException{

		float v2x = x - posX;
		float v2y = y - posY;
		int angle = (int)Math.round(Math.toDegrees(Math.atan2(v2y,v2x) -
Math.atan2(dirY,dirX)));
		this.rotate(angle);
		float distance = (float)Math.sqrt((x-posX)*(x-posX)+(y-posY)*(y-posY));
		this.move(distance);
		posX = x;
		posY= y;
		//make new direction vector via projection of onto destination
		float length =(float)Math.sqrt(v2x*v2x+v2y*v2y);
		dirX= v2x/length;
		dirY= v2y/length;
	}

	public void moveToFace(int x, int y) throws InterruptedException{
		//create vector, calculate angle
		float v2x = x - posX;
		float v2y = y - posY;
		int angle = (int)Math.round(Math.toDegrees(Math.atan2(v2y,v2x) -
Math.atan2(dirY,dirX)));

		this.rotate(angle);
```

```
                //make new direction vector via projection of onto destination
                float length =(float)Math.sqrt(v2x*v2x+v2y*v2y);
                dirX= v2x/length;
                dirY= v2y/length;
        }

        public int getX(){
                return posX;
        }
        public int getY(){
                return posY;
        }

        public void approachPoint(int x, int y)throws InterruptedException{//get within 3 units of
target
                        int dx = this.posX - x;
                        int dy = this.posY - y;
                        int targetX=x, targetY=y;
                        if (dx > 0)
                                targetX = x+1;
                        if (dx < 0)
                                targetX = x-1;
                        if (dy > 0)
                                targetY = y+1;
                        if (dy < 0)
                                targetY = y-1;

                        this.goToPoint(targetX,targetY);
```

```
                    dirX= v2x/length;
                    dirY= v2y/length;
            }
    }
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;
import java.awt.Point;

public class ParkerRootCommand extends CommandCenter{

	private List<Point> botLocation;
	private List<Rectangle> rects;
	private NXTCam camera;


	public ParkerRootCommand(int id, MovementCenter bot){
		camera = new NXTCam(SensorPort.S1);
		botLocation = new ArrayList<Point>();
		rects = new ArrayList<Rectangle>();
		super(id,bot);
	}

	public void executeCommand() throws IOException,InterruptedException{
		while (true){
			try{
				for (int i = 0; i<3;i++){ //alternate between channels
					int command = in.get(i).readInt();
					if (command == CommandCenter.INVESTIGATE)
						investigate();
					else
						reportResults();
				}
			}catch (IOException e){
			}
		}
	}
	private void reportResults() throws IOException{
		//write results back to computer
		con.get(0).close(); //because of three-connection limit
		BTConnection c = Bluetooth.waitForConnection();
		DataOutputStream dos = c.openDataOutputStream();
		for (Rectangle rect:rects){
			dos.writeInt(rect.x);
			dos.writeInt(rect.y);
			dos.writeInt(rect.width);
			dos.writeInt(rect.height);
			dos.flush();
```

```
        }
}

private void requestPositionUpdates() throws IOException{
                for (int i = 0;i<3;i++){
                        try{
                                DataInputStream dis = in.get(i);
                                DataOutputStream dos = out.get(i);

                                dos.writeInt(CommandCenter.INITIALIZE_LOCATIONS);
                                dos.flush();
                                LCD.drawInt(i,i,i);
                                int response = dis.readInt();
                                LCD.drawInt(response,4,4);
                                if (response xa
```

```
private void requestInvestigation()throws IOException, InterruptedException{
        out.get(0).writeInt(CommandCenter.INVESTIGATE);
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;
import java.awt.Point;

public class ParkerLeaf extends CommandCenter{

	public ParkerLeaf(int id, MovementCenter bot){
		selfreported = false;
		super(id,bot);
	}

	public void executeCommand() throws IOException,InterruptedException{
		while (true){
			try{
				int command = in.get(0).readInt();

				if (command == CommandCenter.INITIALIZE_LOCATIONS)
					initializeLocations();
				if (command == CommandCenter.REPORT)
					report();
			}catch (IOException e){
			}
		}
	}

	public void initialize() throws IOException{
		//connect to parent
		addConnection(Bluetooth.waitForConnection());
		ready = true;
	}

	private void initializeLocations() throws IOException{
			out.get(0).writeInt(0);
			out.get(0).writeInt(botID);
			out.get(0).writeInt(thisBot.getX());
			out.get(0).writeInt(thisBot.getY());
			out.get(0).flush();
	}

	private void report() throws IOException,InterruptedException{
		while (true){
			try{
				int x = in.get(0).readInt();
```

```
                int y = in.get(0).readInt();

                addConnection(Bluetooth.waitForConnection());
                in.get(1).writeInt(x);
                in.get(1).writeInt(y);
            }catch(IOException e){
                                //Thread.sleep(500);
            }
        }
    }
}
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;
import java.awt.Point;

public class GagaRootCommand extends CommandCenter{

	private List<Point> botLocation;
	private List<Rectangle> rects;
	private NXTCam camera;

	public GagaRootCommand(int id, MovementCenter bot){
		camera = new NXTCam(SensorPort.S1);
		botLocation = new ArrayList<Point>();
		rects = new ArrayList<Rectangle>();
		super(id,bot);
	}

	public void executeCommand() throws IOException,InterruptedException{
		while (true){
			try{
```

```
	}

	private void requestPositionUpdates() throws IOException{
			for (int i = 0;i<3;i++){
				try{
					DataInputStream dis = in.get(i);
					DataOutputStream dos = out.get(i);

					dos.writeInt(CommandCenter.INITIALIZE_LOCATIONS);
					dos.flush();
					LCD.drawInt(i,i,i);
					int response = dis.readInt();
					LCD.drawInt(response,4,4);
					if (response != CommandCenter.BUSY && response !=
CommandCenter.NO_CHANGE){
						//get botID and point, add it to locations.
						botLocation.add(dis.readInt(),new
Point(dis.readInt(),dis.readInt()));
						dos.writeInt(CommandCenter.CONFIRMED);
					}
				}catch(IOException e){
						LCD.drawInt(34,0,0);
				}
			}
	}

	private void investigate(int i) throws IOException,InterruptedException{
		while (true){
			try{
				int x = in.get(i).readInt();
				int y = in.get(i).readInt();

				//go to location
				thisBot.goToPoint(x,y);
				//camera captures rect information and saves for later
				Rectangle rect = cam.getRectangle(0);
				rects.add(new Rectangle(thisBot.getX(),thisBot.getY(),
rect.height,rect.weight); //records size and location of each tracked object
				break;
			}catch(IOException e){}
		}
	}
}
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;
import java.awt.Point;

public class  GagaSubcommand extends CommandCenter{
	private UltrasonicSensor ultra;
	private LightSensor light;
	private Random gen;

	private boolean autonomous;
	private boolean ready;
	private int botID;

	public GagaSubcommand(int id, MovementCenter bot){
		gen = new Random();
		ultra = new UltrasonicSensor(SensorPort.S1);
		light = new LightSensor(SensorPort.S3);
		autonomous=true;
		super(id,bot);
	}



	public void executeCommand() throws IOException, InterruptedException{
		if(autonomous){//explore, then send
			Thread.sleep(200); //let sensors warm up
			if (ultra.getDistance() < 160 || light.readValue()>40){
				Sound.playTone(1760,1000);
				//found something, send notify parent of coordinates!
				this.requestInvestigation();
			}
			int dX = gen.nextInt(3);
			int dY = gen.nextInt(3);
			if (gen.nextInt()%2==1)
				dX = dX*-1;
			if (gen.nextInt()%2==1)
				dY = dY*-1;
			thisBot.goToPoint(thisBot.getX()+dX,thisBot.getY()+dY);

		}
	}
```

```
	private void requestInvestigation()throws IOException, InterruptedException{
		out.get(0).writeInt(CommandCenter.INVESTIGATE);
		out.get(0).writeInt(thisBot.getX());
		out.get(0).writeInt(thisBot.getY());
		out.get(0).flush();
		//send workers
		out.get(1).writeInt(CommandCenter.WORK);
		out.get(1).writeInt(thisBot.getX());
		out.get(1).writeInt(thisBot.getY());
		out.get(1).flush();
		out.get(2).writeInt(CommandCenter.WORK);
		out.get(2).writeInt(thisBot.getX());
		out.get(2).writeInt(thisBot.getY());
		out.get(2).flush();
	}
}
```

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import lejos.nxt.addon.*;
import javax.bluetooth.*;
import java.io.*;
import java.util.*;
import java.awt.Point;

public class GagaLeaf extends CommandCenter{

	public GagaLeaf(int id, MovementCenter bot){
		super(id,bot);
	}
	public void executeCommand() throws IOException,InterruptedException{
		while (true){
			try{
				int command = in.get(0).readInt();

				if (command == CommandCenter.INITIALIZE_LOCATIONS)
					initializeLocations();
				if (command == CommandCenter.WORK)
					work();
			}catch (IOException e){
			}
		}
	}

	public void initialize() throws IOException{
		//connect to parent
		addConnection(Bluetooth.waitForConnection());
		ready = true;
	}

	private void initializeLocations() throws IOException{
			out.get(0).writeInt(0);
			out.get(0).writeInt(botID);
			out.get(0).writeInt(thisBot.getX());
			out.get(0).writeInt(thisBot.getY());
			out.get(0).flush();
	}

	private void work() throws IOException,InterruptedException{
		while (true){
			try{
				int x = in.get(0).readInt();
				int y = in.get(0).readInt();
```

```
//go to location
thisBot.goToPoint(x,y);
```