

Utah State University

DigitalCommons@USU

Undergraduate Honors Capstone Projects

Honors Program

5-2010

Autonomous Security Patrol System

Jake Erramouspe
Utah State University

Follow this and additional works at: <https://digitalcommons.usu.edu/honors>



Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Erramouspe, Jake, "Autonomous Security Patrol System" (2010). *Undergraduate Honors Capstone Projects*. 46.

<https://digitalcommons.usu.edu/honors/46>

This Thesis is brought to you for free and open access by the Honors Program at DigitalCommons@USU. It has been accepted for inclusion in Undergraduate Honors Capstone Projects by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.



Autonomous Security Patrol System

by

Jake Erramouspe

**Thesis submitted in partial fulfillment
of the requirements for the degree**

of

DEPARTMENTAL HONORS

in

**Electrical Engineering
in the Department of Electrical & Computer Engineering**

Approved:

Thesis/Project Advisor
Dr. YangQuan Chen

Departmental Honors Advisor
Dr. Wynn Walker

Director of Honors Program
Dr. Christie Fox

UTAH STATE UNIVERSITY
Logan, UT

Spring 2010

Senior Design Project for
AUTONOMOUS SECURITY PATROL SYSTEM (ASPS)

ECE 5340 Mobile Robots / ECE 4850 Design 3

December 8, 2009

Jake Erramouspe
Steve Gunderson
Brad Donohoo

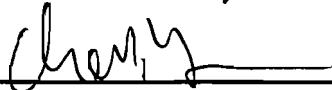
Instructor Approval



Dr. Wei Ren
Department of Electrical Engineering
Utah State University

01/14/2010
Date

Supervisor
Approval



Dr. YangQuan Chen
Department of Electrical Engineering
Utah State University

1/28/2010
Date

Abstract

This project provides an efficient and cost-effective solution to building security and active monitoring. The security is monitored and controlled by autonomous patrol robots. Any indication of a security breach will result in an immediate alarm and activation of the robot group to subdue and tranquilize the intruder.

Table of Contents

Acknowledgements	1
Introduction	2
<i>Background</i>	
<i>Problem Statement</i>	
<i>Solution Summary</i>	
<i>Similar Design Concepts</i>	
<i>Engineering Impact</i>	
<i>Future Possibilities</i>	
Problem Analysis	6
<i>General Problem Specifications</i>	
<i>Map Robot Specifications</i>	
<i>Patrol Robot Specifications</i>	
<i>Police Robot Specifications</i>	
Decision Analysis	8
<i>General Problem Considerations</i>	
<i>Map Robot Considerations</i>	
<i>Patrol Robot Considerations</i>	
<i>Police Robot Considerations</i>	
Solution Description	11
<i>General Problem Solution</i>	
<i>Map Robot Solution</i>	
<i>Patrol Robot Solution</i>	
<i>Police Robot Solution</i>	
Performance Optimization	24
Testing	26
Reflections	32
<i>Technical Challenges</i>	
Project Management	33
<i>Personnel</i>	
<i>Costs</i>	
<i>Timeline</i>	
Conclusion	39
Appendices	41

Acknowledgements

Every engineering project is the result of contributions from the project's team members as well as those whose guidance and consulting were influential and necessary through the design process. The Autonomous Security Patrol System has the following individuals to thank for its successful completion and functionality:

Dr. Wei Ren for his technical design review and counsel

Dr. YangQuan Chen for his project guidance and review

Laura Vernon for her communication editing and review

Introduction

New robotic technologies are currently being used to aid humans in maximizing the efficiency of particular tasks while in dangerous environments. We propose to provide a robotic solution to help with a unique task regarding building monitoring and security. This system is a replacement for human security guards. The system implements autonomous robots with the capability of patrolling, monitoring, and enforcing the security of an environment. Due to its functionality, the project prototype will be called the Autonomous Security Patrol System (ASPS).

Background

Population increases and widespread crime have led to an increased need for security patrols in commercial and industrial environments. Businesses desiring to protect their physical and intellectual property have long implemented human security patrols inside their buildings.

Problem Statement

Human security patrols can often become inattentive to details within the buildings as they become tired or bored. Humans can be put in harm's way should they be faced with a violent intruder. In order to more effectively monitor building security and keep human safety risks to a minimum, a new approach must be explored.

Solution Summary

Robots can be used as a safer, more efficient, and more cost-effective solution to this problem. Robots maintain constant alertness without tiring or growing bored of their job. Due to their relatively low cost and ability to move rapidly through a known environment, robots are a cost-effective solution to monitoring buildings for security purposes. Some of these tasks have already been implemented, but this is the first time they have been brought together as one complete system.

In order to elaborate the intended function of this system, the project is designed around five basic objectives:

1. Map an unknown environment
2. Patrol the environment and search for intruders
3. Report intruder presence with an alarm
4. Shoot the intruder with a tranquilizer dart
5. Call for backup

The prototype must be able to demonstrate successful functionality with each of the five design objectives. The system, in turn, is designed using three unique robot functions to complete the objectives. These robot functions are used as functional partitions to break up the project into manageable parts, and acted out by three distinct robots to illustrate their contribution to the overall objectives of the project. The three robots or functional partitions are the:

1. Map Robot: Creates the preliminary map of the environment, including all obstacles and boundaries.
2. Patrol Robot: Navigates through checkpoints while using saved map information to detect intruders. Discovering an intruder, the robot will sound an alarm, fire a tranquilizer dart, and call for backup.
3. Police Robot: Drives to the intruder after the call for backup.

During today's project exhibition, the three robots along with their associated functions will also be part of the project deliverables.

Similar Design Concepts

Research has been done to localize, if possible, other similar technologies for mapping and patrolling an environment. The only project resembling the Autonomous Security Patrol System implementation is a helicopter system worked on by MIT graduate students. This helicopter system uses laser scanning to create a similar point-based map of its surroundings for patrol and other applications. Screen shots of the MIT helicopter's map program are shown in Figures 1 and 2.

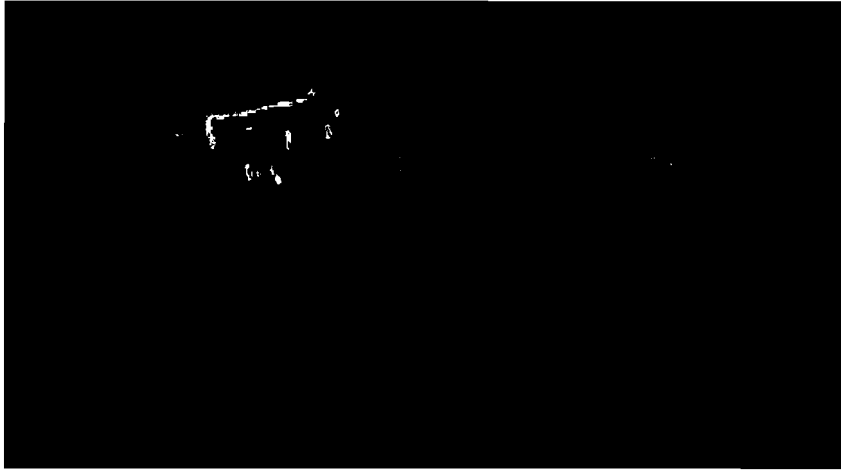


Figure 1



Figure 2

No ground based mapping technology of this type could be found in the research. Especially no robotic security system is known to currently possess similar functionality or design as this project.

Engineering Impact

The Autonomous Security Patrol System will have a globally beneficial impact because it can be used anywhere in the world to enhance security systems and keep human security guards out of harm's way. Economically, security costs will be kept to a minimum while the alertness of the robot raises the level of security effectiveness. Without effective security, a company's physical assets and intellectual property are left in jeopardy. Enhanced building monitoring, then, is an idea that appeals to companies worldwide.

Future Possibilities

Later applications of this system may include residential areas to protect families from crime and vandalism. Outdoor security patrols using similar technologies may also be designed. Beyond the functionality of this first prototype, the possibilities of expansion both in breadth and depth are limitless.

Problem Analysis

The Autonomous Security Patrol System successfully implements three robot functions that fulfill the five project objectives. The map robot is responsible for exploring an unknown environment and creating a map that is transmitted to the patrol robot. The patrol robot uses the map information to patrol the environment and search for intruders. If the robot detects an intruder, it must sound an alarm, fire a tranquilizer dart at the intruder, and call for backup assistance. The police

robot responds to the call for backup by receiving intruder coordinate information from the patrol robot and navigating to it as quickly as possible.

General Project Specifications

The robot group must be able to interface together seamlessly in order to provide an effective demonstration of their ability to monitor and control the security of an environment. The fundamental design decision to be made is the selection of the prototype's robot platform. The platform needs to have several sensors surrounding it for obstacle avoidance and detection that would allow for a fairly robust implementation. These sensors are used to read data in real-time and log it for later use in the patrol robot. The platform must also allow for localization monitoring through encoders or other sensor technology.

Map Robot Specifications

Due to its responsibility in recording a detailed map of the environment, the map robot must be fairly accurate in its detection of boundaries and obstacles. It must also be able to recognize where it is relative to its starting position so that it can accurately record all viewed obstacles in an absolute format. The robot must also follow some mapping algorithm to efficiently map a room with no areas unexplored. This technique will require it to be able to handle multiple interior obstacles and a myriad of outer wall configurations. The data file that it logs must also follow some standard format that can easily and efficiently be interpreted by the patrol robot when it uses it for intruder detection.

Patrol Robot Specifications

The patrol robot must be able to receive a data file from the map robot. This information can then be used in conjunction with its own real-time sensor feedback to patrol and search out intruders within the environment. Sensor information will need to have some type of filtering so that stray sensor data will not be mistaken for an intruder. This robot must be able to patrol the entire environment several times while constantly watching for any object that was not present during mapping. Some type of alarm flag will need to be created for the case when an intruder is detected. The robot must also be equipped with some type of gun that can fire a dart-like projectile 12-24 inches at an intruder. Care must be taken to fire at the center of the intruder to accomplish a direct hit. Coordinate information detailing the location of the intruder upon contact will be transmitted to a police robot that is standing by for backup assistance.

Police Robot Specifications

The police robot must wait in a corner of the environment for the coordinate information of an intruder. Upon receipt of this data, the police robot must be able to quickly navigate to the intruder and avoid any obstacles that may be in the way.

Decision Analysis

Options for creating the system were found to be endless. Due to the prototype nature of the project, most decisions were made in order to create a

demonstration that would be as simple and clear as possible to illustrate the engineering idea. The technology being created to fulfill the Autonomous Security Patrol System objectives may later be easily implemented on other hardware and using other techniques. This project satisfies the creation of a platform upon which more future technological designs may be based.

General Project Considerations

Due to the prototype nature of the project, the simplest and most available robot platform could be used. Amigo bots were found to be the most capable robots for their size that would effectively demonstrate each of the five design objectives. Their perimeter sonar arrays offer ample sensor data for obstacle avoidance and detection. Their three wheeled design also helps to mitigate encoder problems from wheel slipping. The onboard computers can also be used to interface with a dart gun on the patrol robot. All robots are also outfitted with wireless communication that allows for their total operation from a base station. Other robot platforms that use laser range finding, infrared, GPS, and other mobility configurations would be equally compatible with the project objectives. Amigo bots, however, are a simple and elegant solution to minimize cost and unnecessary engineering for a proof of concept design.

Map Robot Considerations

The Amigo bot platform with its associated Aria tools are more than enough technology to fulfill the map robot's function of exploring and mapping an unknown environment.

Patrol Robot Considerations

The patrol robot particularly will take advantage of the onboard computer to control a gun firing system mounted above. Originally, we considered designing a complex gun system out of a solenoid, battery pack, and toy dart gun that would be triggered by the onboard computer's serial port connection. This idea was ruled out due to its complexity and the unnecessary amount of time needed to design it. Its added mechanical complexity also could be potentially prone to problems. Instead, a more elegant approach was taken using a generic brand 3-turret USB controlled missile launcher purchased off the internet. This gun system was less expensive, more versatile, and less time consuming to implement. It fires a small nerf-style foam dart with a sharpened thumbtack glued to the tip with hot glue.

Police Robot Considerations

Originally the police robot was planned to receive intruder coordinate information through some wireless communication aside from the TCP/IP connection that it currently uses to run its program. Again in search of a more simple and effective solution, it was decided that we could use a base station design to relay

information from the patrol robot to the police robot through the network connection that it already had established. This eliminated the need for excess components that would have only increased design costs and work.

Solution Description

The use of Amigo bots and Aria software tools allowed for rapid software development to implement the robot functions as needed to satisfy the project requirements.

General Problem Solution

The Amigo bots are a member of the Pioneer family. They are equipped with a 44.2368 MHz Reduced Instruction Set Computer (RISC) processor. The Amigo bots have 32 KB of Random Access Memory (RAM) and 128KB of FLASH memory. They use the software package Advanced Robotic Interface for Application (ARIA) to run code, which is a C++ based open source development tool allowing users to implement their own code using existing functions. There are eight range-finding sonars that provide a complete 360 degree view of the Amigo bot's surroundings. The range of these sensors is 10 cm to 5m and each sonar sends an individual signal out at a rate of 25 Hz. The Amigo bot platform is run from a 12VDC battery. Mounted on top of the Amigo bots are portable computers that are connected to a wireless router allowing remote communication from a base station or between Amigo bots. The robot group utilizes wheel encoder data, sonar sensor readings, built in wireless

communication to the base station, and the onboard computer to fulfill all of the design objectives.

Map Robot Solution

The map robot's program routine starts with it physically placed at an origin point in the corner of an unknown environment. Its orientation is such that its left side sonar may view the room and its right side sonar may follow the wall next to it. This starting position and orientation of the map robot will be roughly marked so that the patrol robot can start at the same position and orientation.

In order to reduce errors in placing both the map robot and patrol robot at exactly the same starting positions, we implemented a localize function. This function uses the sonar data to determine the offset of the robots relative to the walls directly to the right and behind them. The code for this function is shown below:

```
void LabDataLogger::Localize(void)
{
    x_offset = myRobot->getSonarRange(7)*0.82;//sonar7 * cos(35)
    y_offset = myRobot->getSonarRange(5);

    printf("x_off: %f, y_off: %f\n",x_offset,y_offset);
}
```

If the robots are run without the localize function, the map's origin is at a relative location that changes depending on the starting position of the robot. As seen in the Figure 3, running the robot with the localize function yields a starting point of

(0, 0) and because the patrol robot also uses this function the two maps will coincide with each other, regardless of the exact starting position.

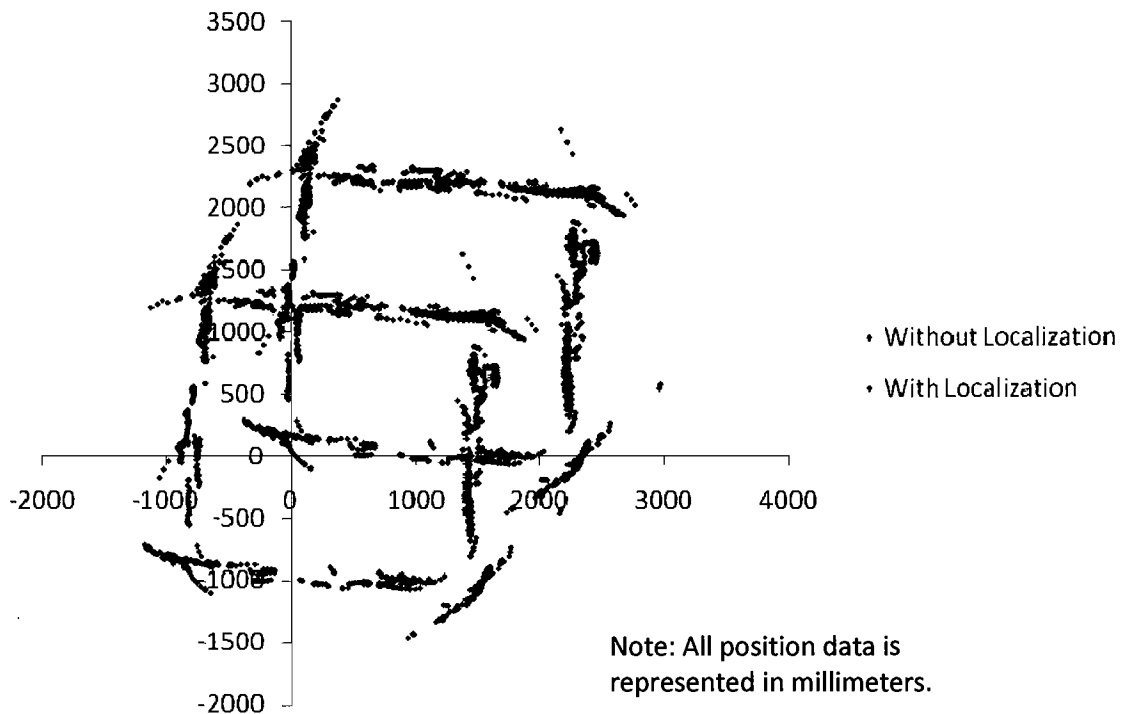


Figure 3

The mapping routine begins as the program starts and the robot follows the wall to the right of it, circling the room in a counter-clockwise manner. The left side sonar sensors are then free to view the environment and any obstacles that may be present. Sonar feedback is used in conjunction with encoder data to calculate the absolute (x, y) value of each observed obstacle or wall. The formulas used to record x and y are:

(x_r, y_r) : Robot current position

θ_r : Robot current orientation

d_s : Sonar distance read

θ_s : Sonar sensor angle

(x_{log}, y_{log}) : Logged absolute sonar coordinate

$$x_{log} = x_r + d_s * \cos(\theta_r + \theta_s)$$

$$y_{log} = y_r + d_s * \sin(\theta_r + \theta_s)$$

The code implementation for the coordinate data calculation is shown below:

```
for (sonar=0; sonar <= 7; sonar++)
{
    s = myRobot->getSonarRange(sonar);
    fprintf(dataFile, "\t%d", s);

    sonAngle = getSonarAngle(sonar);
    radAngle = (theta+sonAngle)*(3.1415926/180.0);

    if (s < 1500)
    {
        abs_x = (x + s*cos(radAngle)) + x_offset;
        abs_y = (y + s*sin(radAngle)) + y_offset;

        fprintf(coordData, "\n%.0f\t%.0f", abs_x, abs_y);
        fflush(coordData);
    }
}
```

The code has been optimized by using nested if and for loops. This technique cuts down on code size and increases the readability for later troubleshooting.

The robot records the x and y values as calculated into a data file in a standard format. If the robot sees any values greater than 5000 mm on its sonar, it will disregard this value because it is not valid data. Since multiple sensor data is

used, all parts of the map will be explored using this technique including the areas behind obstacles. Upon successful return to the starting origin, the map robot ends its program and leaves the (x, y) map data file ready to be loaded into the patrol robot's program. An example of a completed map is shown in Figure 4.

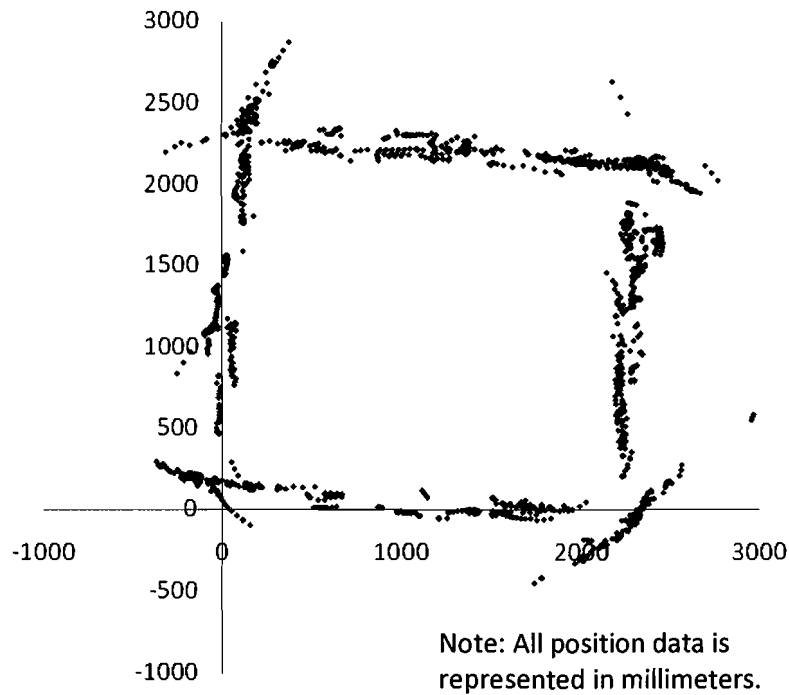


Figure 4

The map robot's state diagram is shown in Figure 5.

Map Algorithm State Diagram

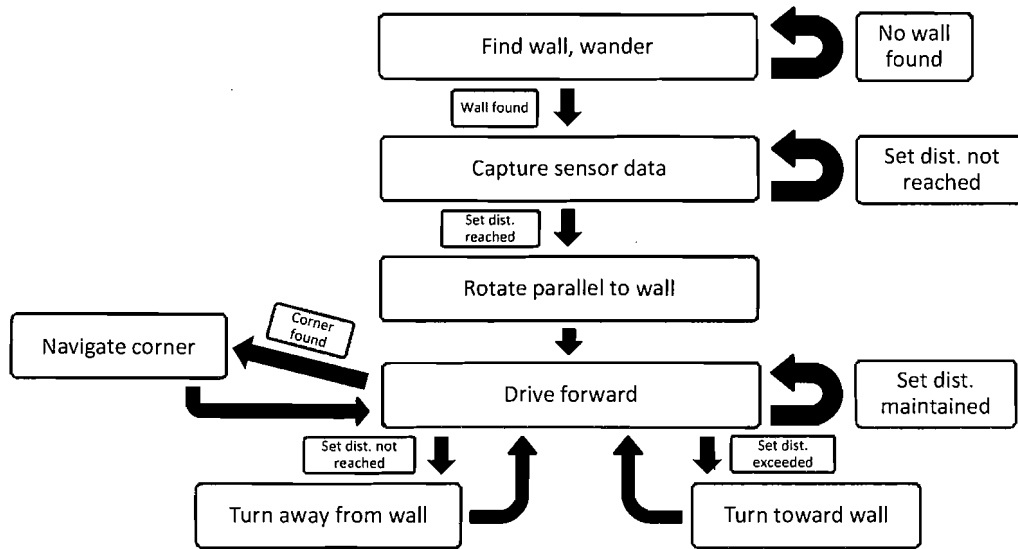


Figure 5

Patrol Robot Solution

The patrol robot's program routine starts with it physically placed in the same position and orientation as the map robot. The patrol robot does not need to be placed as precise because of the localize function being used. A remote desktop connection also needs to be established on the base station with the robot's onboard computer. The computer must be logged in, and the USB missile launcher software installed and set as the active window on the remote computer.

The patrol robot begins by loading the map information from the map robot data file. A quick sort is performed on the data to reduce the burden on the processor. The data is sorted first by the x-coordinate, then by the y-coordinate.

The robot is configured to begin patrolling the now known environment by following the wall in a counter-clockwise manner. Using real-time sonar data from all sides of the robot, it similarly calculates absolute (x, y) coordinates of sonar feedback data as done by the map robot. This robot, however, compares each data point with all data points in the map data file, looking for those outside of its tolerance range of 150 millimeters from 5 known data file points. The code implementation for the search coordinates is shown below:

```

#define TOLERANCE          150

void AnomalyDetect::SearchCoords(int x, int y, int angle)
{
    int coord_found = 0;
    int start = xcoords.size()/2;

    if(xcoords[start] >= x){

        for (int i=start; i>=0; i--)
            {
                // searching from middle of xcoords array to xcoords[0]
                if (xcoords[i] < (x+TOLERANCE) && xcoords[i] > (x-TOLERANCE))
                    {
                        if (ycoords[i] < (y+TOLERANCE) && ycoords[i] > (y-TOLERANCE))
                            {
                                coord_found++;
                            }
                    }
            }
    }

    else{

        for (int i=start; i<xcoords.size(); i++)
            {
                // searching from middle of xcoords array to xcoords.size()
                if (xcoords[i] < (x+TOLERANCE) && xcoords[i] > (x-TOLERANCE))
                    {
                        if (ycoords[i] < (y+TOLERANCE) && ycoords[i] > (y-TOLERANCE))
                            {
                                coord_found++;
                            }
                    }
            }
    }
}

```

This function first searches and compares the x and y values of the logged data array to the real-time value observed by the patrol robot's sonar, looking for points outside the 150 millimeter tolerance. The tolerance is needed because real world sensor readings are not perfectly accurate. If the observed data point anomaly cannot be found in the stored data array, indicating a potential intruder, then a coord_found counter is incremented. The code implementation for the anomaly count is shown below:

```
#define MIN_COORD_CNT    5

if (coord_found <= MIN_COORD_CNT)
    anomaly_count++;
```

If the real-time data point is not found within the 150 millimeter tolerance of 5 logged data array coordinates, an anomaly counter is incremented. When 25 anomaly counts have been observed, the anomaly points are recognized as an intruder and the alarm flag is activated as shown in the following code:

```
if (anomaly_count >= MAX_ANOMALY){
    if (anomaly_flag == false)
    {
        printf("\n\nANOMALY FOUND!\n");
        anomaly_flag = true;
        anomaly_x = x-x_offset;
        anomaly_y = y-y_offset;
        anomaly_angle = angle;
    }
}
```

The 25 points will then be averaged to find their midpoint. The patrol robot then turns toward the midpoint and approaches it until within 60 centimeters as shown in the following code:

```
if (anomalydetect.anomaly_flag == true)
    break;

printf("\ngoal: x: %d, y: %d, angle: %d\n",
anomalydetect.anomaly_x, anomalydetect.anomaly_y,
anomalydetect.anomaly_angle);

robot.lock();
robot.setVel(0);
robot.unlock();
PosDir.rotate(anomalydetect.anomaly_angle);
ArUtil::sleep(2000);
PosDir.goForward(100);

while(robot.getSonarRange(2) > 500 && robot.getSonarRange(3) >
500);

if (robot.getSonarRange(2) < robot.getSonarRange(3) > 500)
{
    while (robot.getSonarRange(3) > 600)
        PosDir.rotate(5);
}
else
{
    while (robot.getSonarRange(2) > 600)
        PosDir.rotate(-5);
}

robot.setVel(0);
robot.setRotVel(0);
printf("\nFIRE!!!!!!\n");
```

A function is then called in the program to find and set active the remote desktop window on the base station computer. Once active, the program will send a space press into the remote computer's window to trigger the firing mechanism. This procedure utilizes the following code as shown below:


```

#include "windows.h"

RemoteDesktop = FindWindow(NULL, "192.168.1.43 - Remote
Desktop");

SetForegroundWindow(RemoteDesktop);
ShowWindow(RemoteDesktop, SW_SHOWNORMAL);

ArUtil::sleep(1000);

// Send space bar command to remote desktop
keybd_event( VK_SPACE, 0x39, 0, 0);
keybd_event( VK_SPACE, 0xB9, KEYEVENTF_KEYUP, 0);

```

The USB missile launcher will fire a dart that has a sharpened thumb tack glued to its tip. This thumb tack will pop a balloon that is used to simulate the intruder. The loud bang of the balloon popping will be a great demonstration of the neutralization of the intruder.

The patrol robot uses ArNetworking to communicate with the police robot. It acts as the server, giving information to the client each time a request is made. The patrol robot sends an anomaly detected flag, and its current x and y coordinates. The value of the anomaly flag is dependent upon whether the patrol robot has detected an intruder. The following code shows how the packet that the police robot requests, is stored:

```

void ASPS_ServerInfoRobot::update(ArServerClient *client,
ArNetPacket *packet)
{
    ArNetPacket sending;

    myRobot->lock();

    sending.byte4ToBuf((int)alarm_flag);
    sending.byte4ToBuf((double)myRobot->getX());
    sending.byte4ToBuf((double)myRobot->getY());
}

```

```

myRobot->unlock();

client->sendPacketUdp(&sending);

}

```

The patrol robot's state diagram is shown in Figure 6.

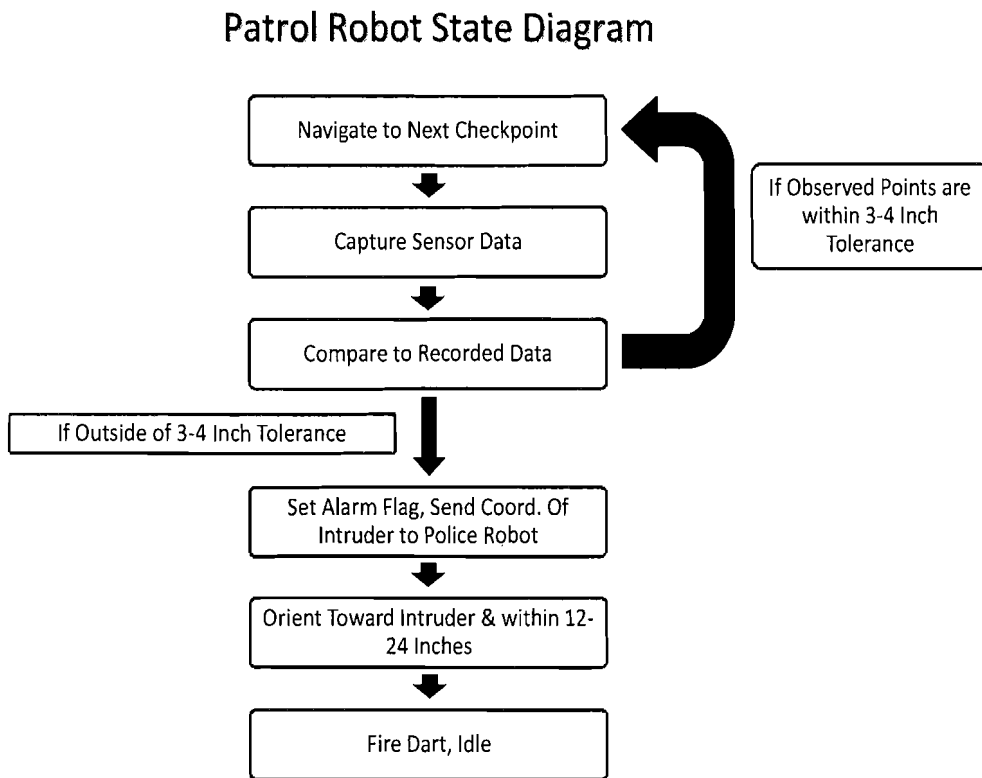


Figure 6

Police Robot Solution

The police robot is placed in a corner of the environment during both the mapping and patrolling stages. It remains inactive in that corner until it receives

the intruder coordinate from the patrol robot. It then can maneuver as quickly as possible to that point while avoiding obstacles and the patrol robot along the way. Once it reaches a distance of 60 centimeters from the intruder, it stops. The police robot uses the other half of ArNetworking and takes on the client position. It asks the patrol robot for three pieces of information every 100 ms, an anomaly flag, and the patrol robots current x and y position. Once it receives this information, the police bot checks the value of the anomaly flag as can be seen in the following code:

```
OutputHandler::handleOutput(ArNetPacket *packet)
{
    alarm_flag = (int) packet->bufToByte4();
    myX = (double) packet->bufToByte4();
    myY = (double) packet->bufToByte4();

    if (alarm_flag == 1)
    {
        printf("Alarm received!\n"
               "X: %lf, Y: %lf\n", myX, myY);
        fflush(stdout);

        myClient->requestStop("update");

        alarm = 1;
    }
}
```

If the anomaly flag has not been set, the robot stays inactive. But as soon as the police robot sees that the anomaly flag has been set, it starts running its program. In order to drive to the location of the intruder the police robot had to know where it was at relative to the starting position of the patrol robot. This would allow the two robots to localize their positions to each other. This action is accomplished in the code below:

```
#define POLICE_X_OFF    (-450)

#define POLICE_Y_OFF    1100

robot.setVel(100);

ArUtil::sleep((outputHandler.myX - POLICE_X_OFF)*10);

robot.setDeltaHeading(-90);

ArUtil::sleep(abs((POLICE_Y_OFF - outputHandler.myY)*10));

robot.setVel(100);
```

We first defined the offset between the patrol robot and the police robot's starting position. We then set the velocity of the police robot to 100 mm/sec. To figure out how far to drive, we took the current value of the patrol robot, sent to us over the network, and subtracted off the offset of the two robots. This would give us a distance in millimeters of how far we needed to travel. To convert this to a time unit, we multiplied it by 10, because this would match our units for velocity and distance. The police robot turned itself 90 degrees toward the intruder once it had gone its complete x distance. It then continued in the appropriate y direction toward the intruder to give back up to the patrol robot. The police robot's state diagram is shown in Figure 7.

Police Robot State Diagram

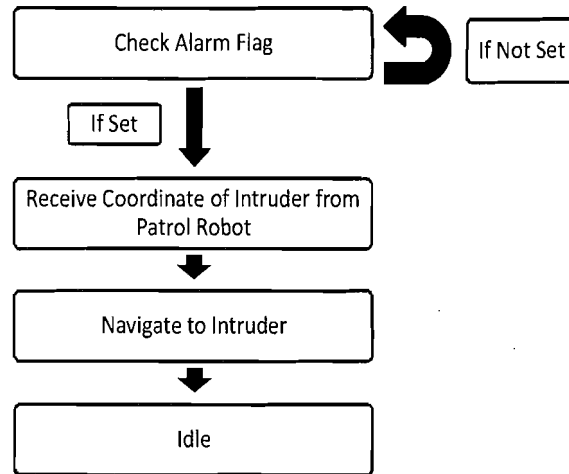


Figure 7

Performance Optimization

Several key design elements were found to be bottlenecks to smooth performance of the robot system. Care had to be taken with each of these elements to refine the design into streamlined processes.

Due to the possibility of large amounts of sonar data being logged by the mapping robot, attention had to be paid to how the (x, y) data file would store the information. We implemented vectors instead of a typical array. This allowed us to increase the size of our data field without having to know exactly how large it needed to be. We decided that after the data was stored, we would have the program numerically sort all of it in order so that the patrol robot could compare

its values more rapidly. Instead of needing to search the entire file for close values, it would only need to search less than halfway through the data for a match.

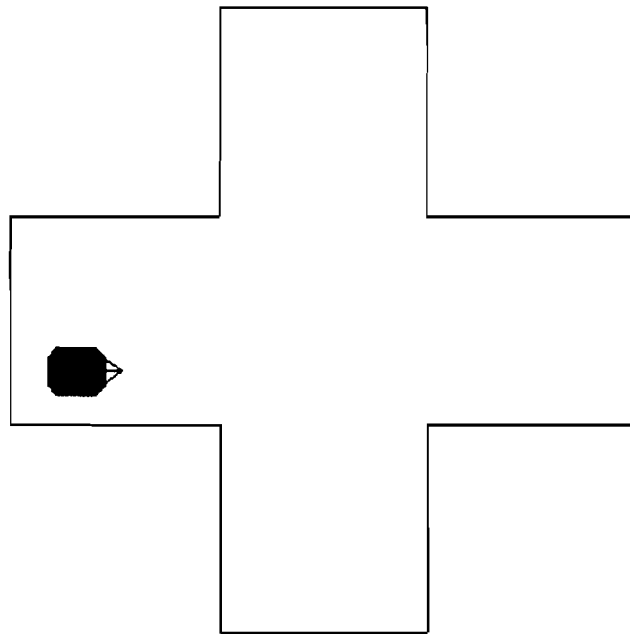
The wall following algorithm used by both the map and patrol robots also needed work to keep the robot steady. Small damping was implemented to keep the robot driving on a straighter course instead of oscillating back and forth as it tried to follow a straight wall. Another feature that was added to the wall following was the ability to re-find the wall if, for some reason the robot gets out of range of the wall.

Inside corners of the environment also posed a problem for sonar reflection. The wall following algorithm had to use better sonar monitoring techniques in the program to allow the robot to make a clean 90 degree turn at each corner. Traces of this problem still exist sometimes during testing, although we have been able to minimize it substantially.

Another helpful feature we added to all of our code was to use global variables. This allowed us to change our parameters in one place, rather than trying to keep track of changing all of the variables throughout the code. This simplified optimizing our code and cut out possibility of errors.

Testing

In order to test the project, each functional partition has first been tested individually, then all together as a harmonious system. Each robot outputs several different data files to allow for troubleshooting problems. Preliminary testing of the map robot was using a computer simulation to test its data logging functionality. Resulting data values from the (x, y) list were then plotted in excel and compared to the simulation map. These results are shown in Figures 8 and 9.



Mobile Eyes Simulation

Figure 8

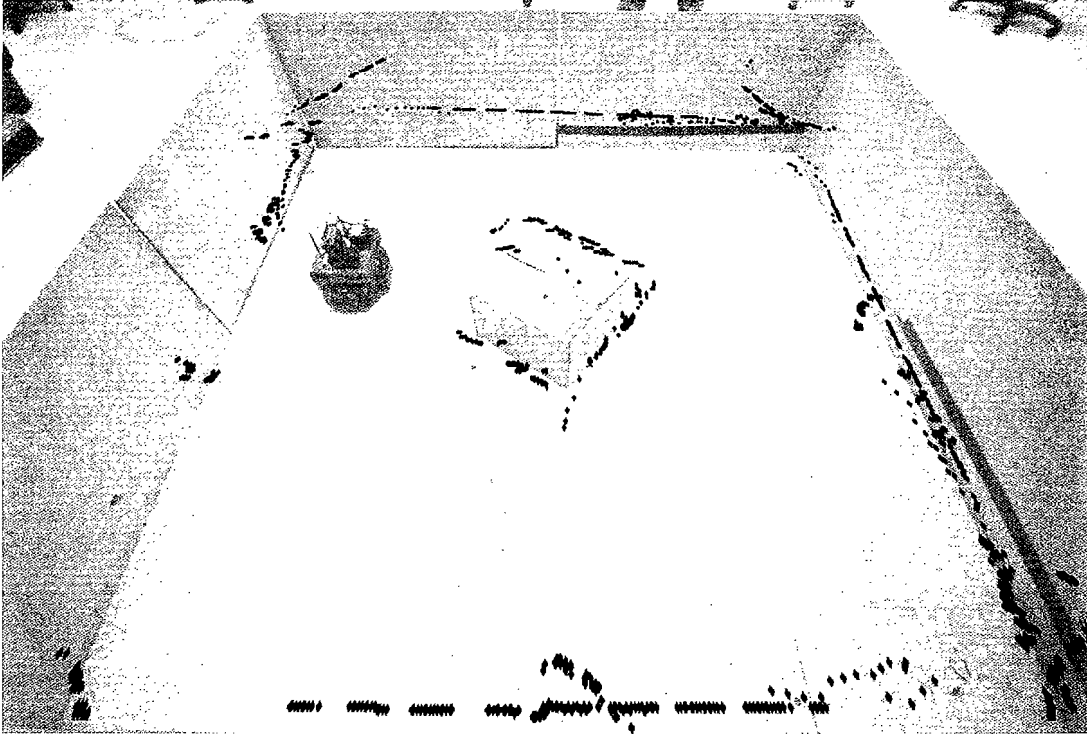


Figure 10

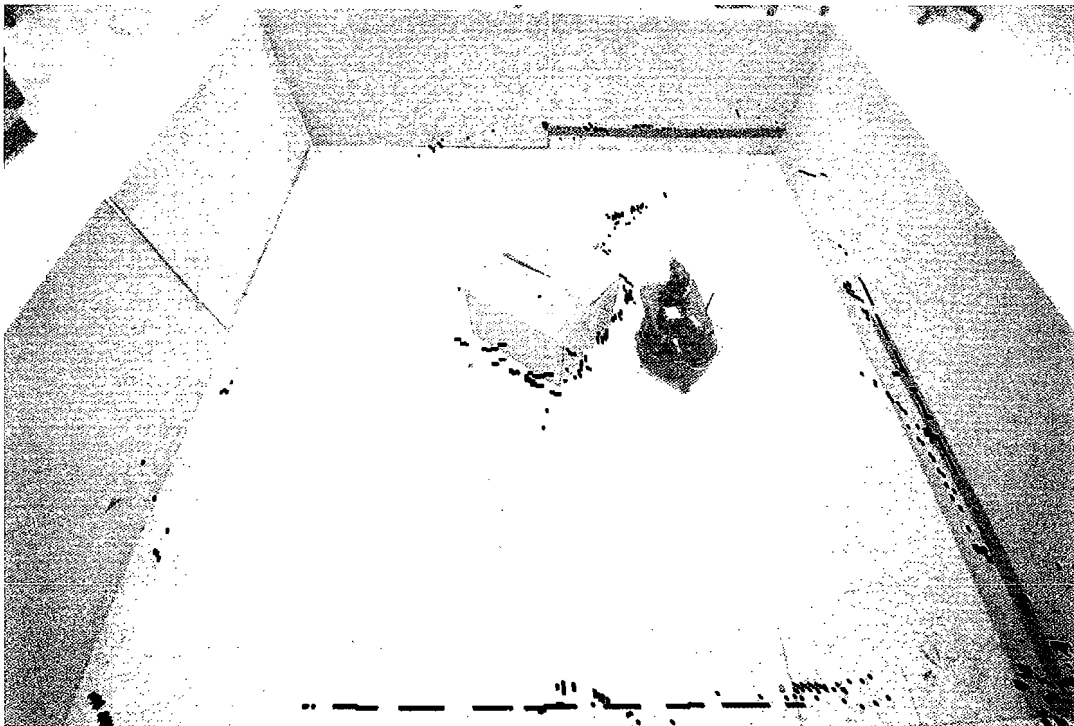


Figure 11

Much of the testing was completed during design as new functions were implemented or adjustments were made to program tolerances. Unique environments with different obstacle positions and layouts were used in order to understand the program's ability to make decisions and have the robots maneuver effectively as autonomous units. After component and functional partition testing was completed, the complete system was tested together through several runs. After tolerances had been tightened and error allowances made for inaccurate sonar reflection and other variables, the system performed magnificently. Figure 12 shows one of our runs with the complete system running. The blue represents the map that the map robot compiled and the red represents the patrol robot's map. The two maps line up quite nicely, due to our localization function. We still have some points that are not part of the actual map, but come from reflections off non ideal surfaces. These points are fine. The different robots will always get the same information because the sonar reflects off the various objects the same each time. The anomaly was placed behind the obstacle in the middle, so the patrol robot would not initially see the anomaly.

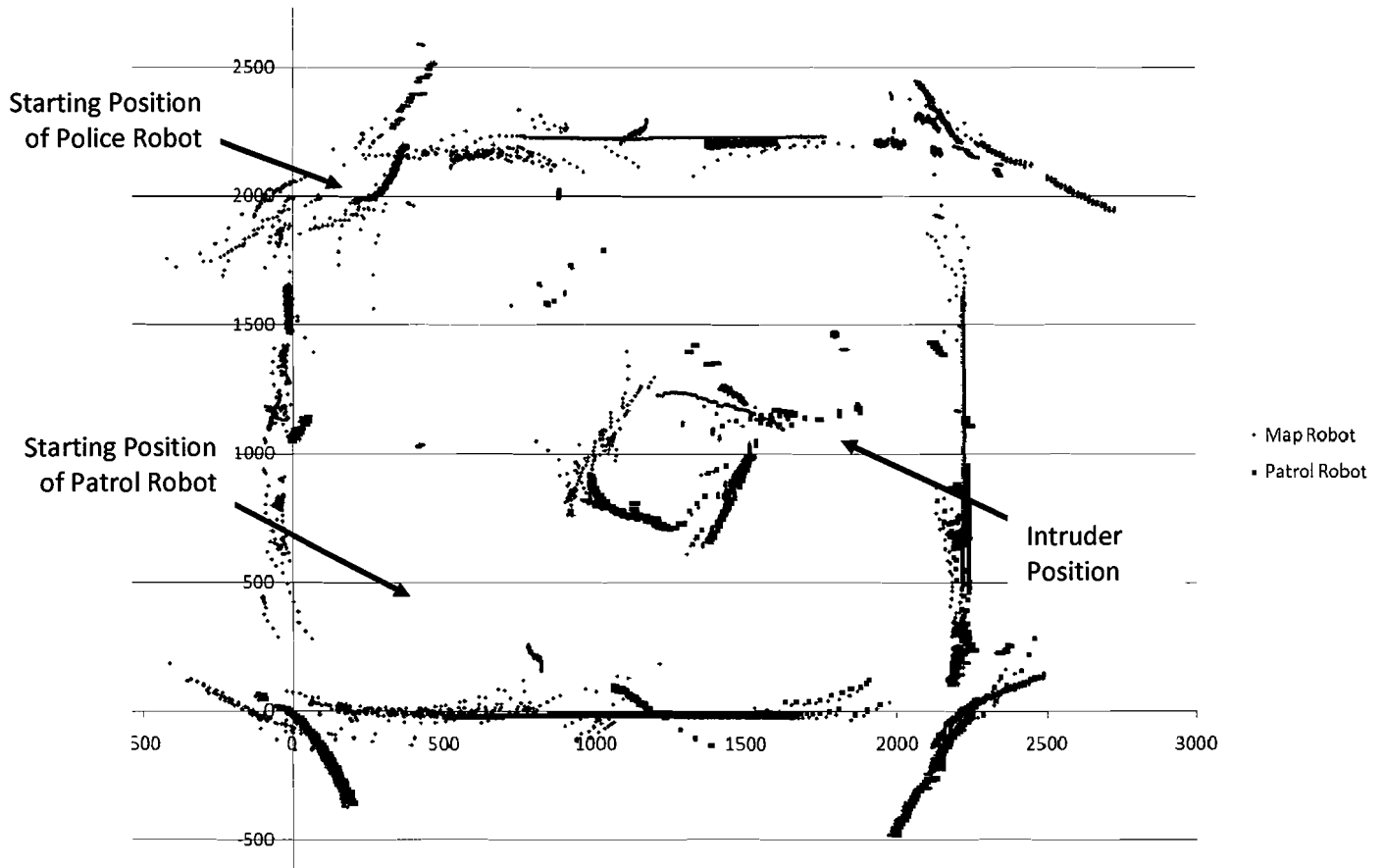


Figure 12

This test demonstrates the complete functionality of each of our components working together. Our system will work in any environment that meets our requirements.

Figures 13 and 14 shown below are examples of what our command prompts look like for the patrol robot and police robot. The command prompts were essential for our troubleshooting. They helped us to know where the robots were in their respective programs.

Patrol Robot Command Prompt

```
Command Prompt
Made it to FOLLOW WALL function
Sonar0 = 1895, Sonar1 = 1835, Sonar2 = 1965, Sonar3 =2309, Sonar4 =596, Sonar5 =
522, Sonar6 = 1323, Sonar7 = 3589
Going forward
ANOMALY FOUND!
Coming to a wall straight ahead -> Rotating 90 deg CCW
Rotating 90.00 deg
Going forward
goal: x: 1489, y: -118, angle: -15
Rotating -15.00 deg
Going forward
FIRE!!!!!!
Server shutting down.
Sending shutdown
Disconnecting from robot.
accept: No error
Failed on read UDP, error 10038
Failed on the udp read
C:\Program Files\MobileRobots\Aria\bin>
```

Figure 13

Police Robot Command Prompt

```
Command Prompt
C:\Program Files\MobileRobots\Aria\bin>ASPS_Police -rh 192.168.1.13
Client now connected to server.
Server connected to us on udp port 7272
Connected to server.
You may press escape to exit
Connected to remote host 192.168.1.13 through tcp.

Syncing 0
Syncing 1
Syncing 2
Connected to robot.
Name: not_set
Type: Pioneer
Subtype: amigo-sh
Loaded robot parameters from amigo-sh.p
Alarm received!
X: 938.000000, Y: 607.000000
Server shutting down and closed connection.
Disconnected from server.
ArSocket::read: called after socket closed
Connection to 127.0.0.1 closed
Lost connection to server (couldn't recv).
Disconnecting from robot.
^C
C:\Program Files\MobileRobots\Aria\bin>
```

Figure 14

Reflections

Upon viewing the complete system function perfectly the first time, it is easy to see how effective robots could be at actively monitoring and enforcing building security. This project was thought up, designed and implemented in one semester and yet maintains far reaching implications for security worldwide. This prototype system clearly demonstrates the robot security concept's feasibility in real world situations. If more engineering time and money were invested into similar projects that build off of this idea, we truly believe that globally implementing our system would make the world a safer place.

Technical Challenges

This section will talk about some of the problems we faced along the way of completing our system.

During our mapping robot design we were having trouble following the wall consistently. The robot would dive into walls or would lose its way when going around a corner. After heavily analyzing our output files to see what type of data the robot was outputting, we determined that we were not taking in data from our sonar often enough. The robot would get lost because it was relying on old data and the new data, when it was taken in, was quite a bit different. To fix this problem, we created a `getNewSonar` function and placed it inside our `if` and `for` loops, where it originally would not get new data.

Sonar reflections are problems that have to be dealt with when using the robots in a real environment. There is no escaping this imperfection and because of this we had to allow for tolerances when comparing our data. One of the largest problems we had was pickup up bad data when we first started our mapping and patrol robots. Because we started our robots in a corner, the sonar would be reflected and not travel back to the robot. To fix this problem, we tested several distances away from the wall to find one where our reflections were minimized. This distance was found to be 43 centimeters from the wall in each direction. Another problem we faced was starting the robot before the sonar had a chance to turn on and begin calculating distances. To fix this problem we initialized the sensors and told the robot to sleep for two seconds. This would allow the robot to turn on its sensors and collect valid data.

Project Management

As a complex engineering project, the Autonomous Security Patrol System required good planning in order to be completed on time and within budget. Without a concert effort of team members to accomplish project objectives, this prototype design would certainly have been a failure.

Personnel

This project has been brought to life through the work of Jake Erramouspe, Steve Gunderson, and Brad Donohoo. As a team we have worked together to brainstorm, plan, design, and implement the Autonomous Security Patrol

System. Each individual took on responsibilities according to their strengths and abilities. Everyone was held accountable for their responsibilities by peer review. Jake Erramouspe acted as team leader to coordinate our design efforts and assure all work being done was within the scope of the project objectives. Jake worked on all of the software design engineering. He was also responsible for his lab book and project documentation. Steve Gunderson handled most of the project documentation including the organization of presentation slides, website, lab book, poster and final report. He worked on hardware engineering with the patrol robot gun system along with its implementation. Brad Donohoo took care of scheduling and the timelines to always assure that tasks were being completed in order and according to plan. He also was involved with software design responsibilities for the robot map, patrol and networking algorithms.

Both Jake and Steve are using the Autonomous Security Patrol System as their senior design project for Utah State University's Department of Computer and Electrical Engineering. Over 480 man hours have been spent between the two in order to complete this project.

Costs

The Autonomous Security Patrol System project proposal included a preliminary budget of \$300 for material costs and \$100 for software costs. After component options were considered and design choices weighed, the preliminary design costs were down to \$40 total. After making the last change of the gun system

component, the resulting total project cost of the gun system with balloons including tax and shipping was \$28.04. The total was a mere 7% of the originally estimated budget. This reduction can be directly related to good design simplification and component research. Receipts of the project expenditures are shown in Figures 15 and 16.

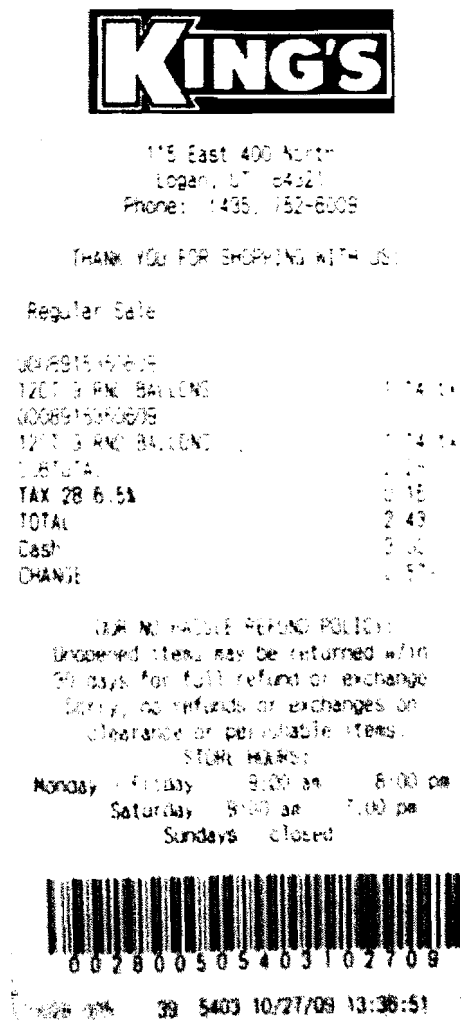


Figure 15

Order Placed: October 22, 2009
Amazon.com order number: 104-1028638-9207432
Order Total: \$25.61

Shipped on October 22, 2009	
Items Ordered	Price
1 of: <i>USB Missile Launcher with 3 Foam Missiles</i>	\$17.99
Condition: New	
Sold by: Computer Geeks (seller profile)	
Shipping Address:	Item(s) Subtotal: \$17.99
Steve Gunderson	Shipping & Handling: \$7.62
930 N 600 E APT 55	-----
LOGAN, UT 84321-3497	Total Before Tax: \$25.61
United States	Sales tax: \$0.00

Shipping Speed:	Total for this Shipment: \$25.61
Standard	-----

Payment Information	
Payment Method:	Item(s) Subtotal: \$17.99
MasterCard Last digits: 4736	Shipping & Handling: \$7.62

Billing Address:	Total Before Tax: \$25.61
Steve Gunderson	Estimated Tax: \$0.00
930 N 600 E APT 55	-----
LOGAN, UT 84321-3497	Grand Total: \$25.61
United States	-----

Figure 16

Timeline

Both a Gantt Chart and Pert Chart are used to understand the timeline of project functional partitions and specific tasks. The Gantt Chart shows an overall burn down of project related tasks and their associated due dates. The Pert Chart illustrates these tasks as dependent bodies that are completed sequentially and grouped according to the different functional partitions. The charts are shown in Figures 17 and 18.

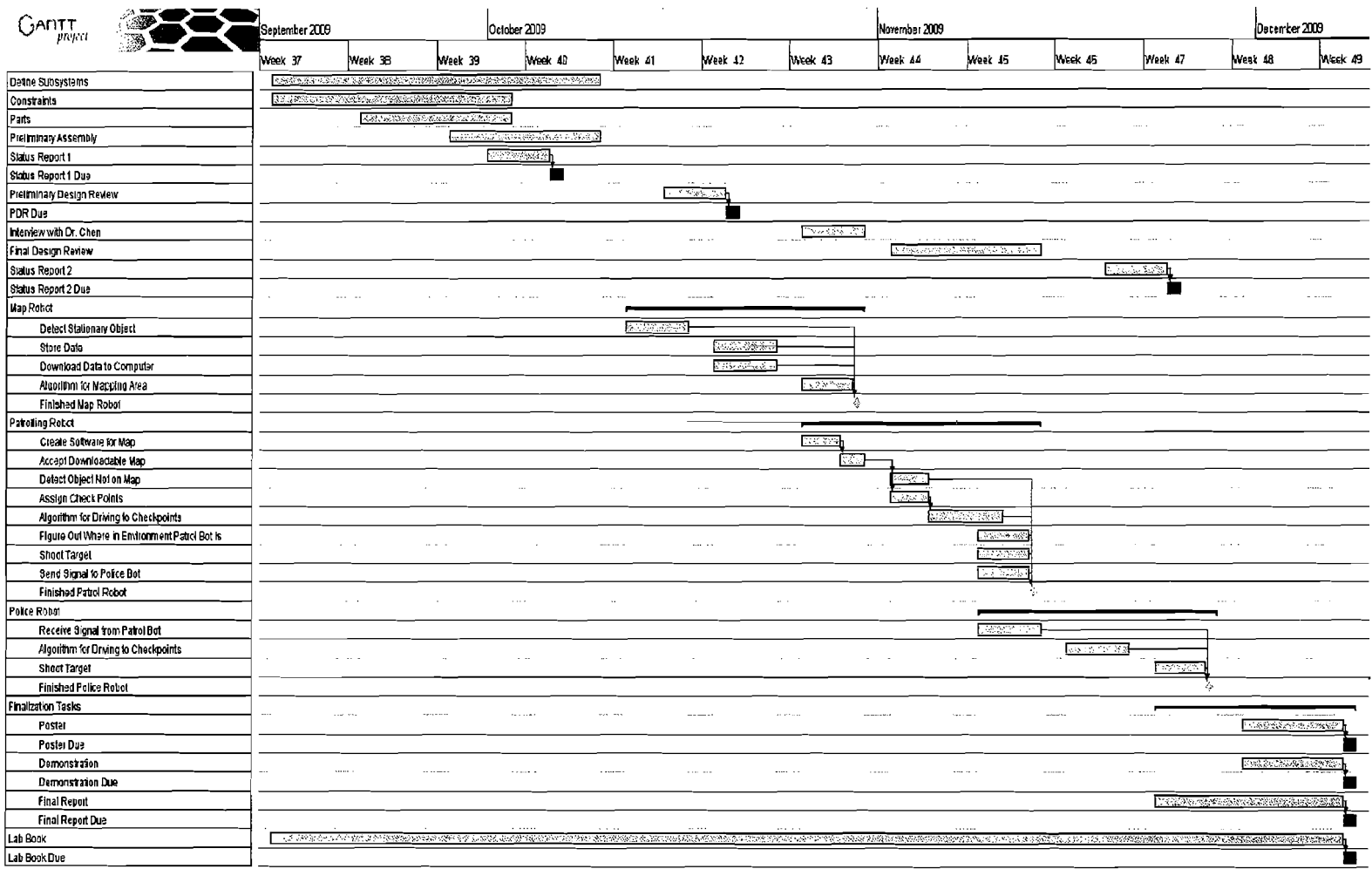


Figure 17

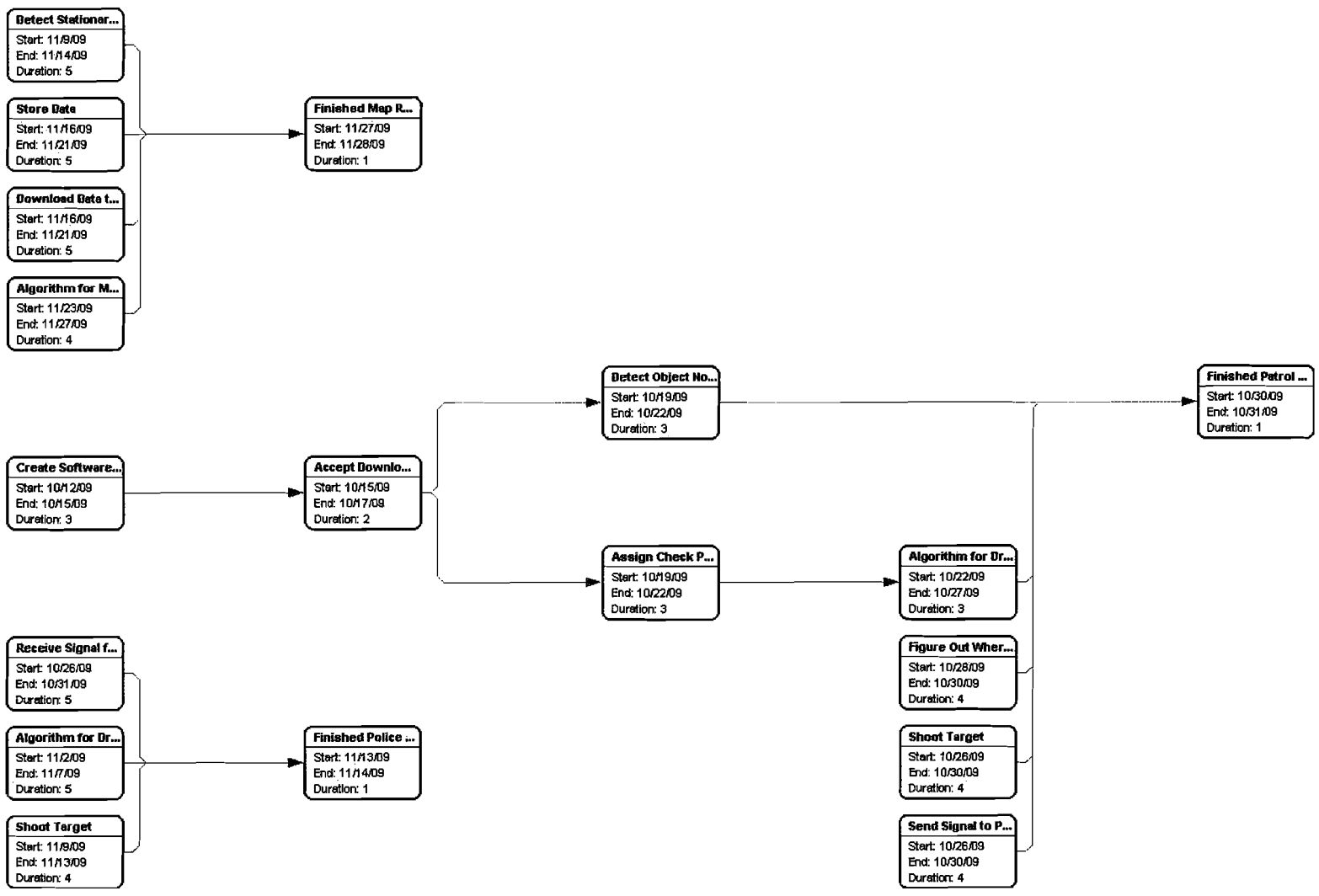


Figure 18

Conclusion

This report serves to explain in detail all engineering aspects of the Autonomous Security Patrol System project. This project is an efficient and cost-effective solution to building security and active monitoring. Robots are a great solution to aid humans in helping secure and monitor dangerous environments. This system does not eliminate the human element; it simply enhances the current process.

The successful fulfillment of five project objectives makes this prototype successful. The objectives are:

1. Map an unknown environment
2. Patrol the environment and search for intruders
3. Report intruder presence with an alarm
4. Shoot the intruder with a tranquilizer dart
5. Call for backup

To demonstrate this functionality effectively, three distinct robot functions are designed uniquely. The robot functional partitions are:

1. Map Robot: Creates the preliminary map of the environment, including all obstacles and boundaries.
2. Patrol Robot: Navigates through checkpoints while using saved map information to detect intruders. Discovering an intruder, the robot will sound an alarm, fire a tranquilizer dart, and call for backup.
3. Police Robot: Drives to the intruder after the call for backup.

The video demonstration shown during the project exhibition illustrates how effectively robots can work to maintain a high level of security in an environment. As a prototype and proof of concept, the Autonomous Security Patrol System has been an overwhelming success. Many qualified individuals have already made comments regarding the future impact of this type of research. It is our hope that continued engineering time will be invested into this idea to make the world a better place through autonomous robot solutions.

Appendices

MapRobot.cpp

```
/*
  Brad Donohoo, Steve Gunderson, Jake Erramouspe
  ASPS: Automated Security Patrol System
  Map Robot
  27 October 2009
*/

#include "Aria.h"
#include "LabDataLogger.h"
#include "RobotPosDir.h"

int main(int argc, char** argv)
{
    bool start = true;

    // mandatory init
    Aria::init();

    // set up our parser
    ArArgumentParser parser(&argc, argv);
    // set up our simple connector
    ArSimpleConnector simpleConnector(&parser);
    // robot
    ArRobot robot;
    // add sonar
    ArSonarDevice sonarDev;
    // wall follower
    RobotPosDir PosDir(&robot);

    // load the default arguments
    parser.loadDefaultArguments();

    // parse the command line
    if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
    {
        Aria::logOptions();
        exit(1);
    }

    // a key handler so we can do our key handling
    ArKeyHandler keyHandler;
    // let the global aria stuff know about it
    Aria::setKeyHandler(&keyHandler);
    // toss it on the robot
    robot.attachKeyHandler(&keyHandler);
    printf("You may press escape to exit\n");

    // add the sonar to the robot
    robot.addRangeDevice(&sonarDev);
}
```

```

// set up the robot for connecting
if (!simpleConnector.connectRobot(&robot))
{
    printf("Could not connect to robot... exiting\n");
    Aria::exit(1);
}

/*****DATALOGGER CODE
HERE*****/
LabDataLogger datalogger(&robot);

datalogger.NewLogFile("output.txt", "coords.txt");

datalogger.LogData();

/*****/

// start the robot running, true so that if we lose connection the
run stops
robot.runAsync(true);

// turn on the motors
robot.comInt(ArCommands::ENABLE, 1);

// This is how we will send commands to the robot.
ArActionInput input("input");

robot.addAction(&input, 50);

while(robot.isConnected())
{
    ArUtil::sleep(4000);

    PosDir.wallFound = 0;

    if (start == true)
    {
        datalogger.Localize();
        datalogger.CleanLogfile("output.txt", "coords.txt");
        start = false;
    }

    while(1)
    {
        PosDir.findWall();

        while(PosDir.wallFound == 1){
            PosDir.followWall();
        }

    }
}

robot.waitForRunExit();

```

```

    // now exit
    Aria::exit(0);
    return 0;
}

```

LabDataLogger.h

```

/*****
    Brad Donohoo, Steve Gunderson, Jake Erramouspe
    ASPS: Automated Security Patrol System
    LabDataLogger Class
    24 September 2009
*****/

#ifndef _LABDATALOgger_H_
#define _LABDATALOgger_H_

#include <fstream>
#include <iostream>

using std::ios;
using std::ofstream;

class LabDataLogger
{
protected:
    ArRobot *myRobot;
    FILE* dataFile; //File for saving data
    FILE* coordData; //File for saving coords
    ArFunctorC<LabDataLogger> myFunctor;

    double x_offset;
    double y_offset;

public:
    /// Constructor
    AREXPORt LabDataLogger::LabDataLogger(ArRobot *robot);
    /// Destructor
    AREXPORt LabDataLogger::~LabDataLogger();

    AREXPORt int getSonarAngle(int s);
    AREXPORt void Localize(void);
    AREXPORt void LogData(void);
    AREXPORt void OpenLogFile(const char* logFileName1, const char*
logFileName2);
    AREXPORt void CloseLogFile(void);
    AREXPORt void CleanLogfile(const char* logFileName1, const char*
logFileName2);
    AREXPORt void NewLogFile(const char* logFileName1, const char*
logFileName2);
};

#endif // _LABDATALOgger_H_

```


LabDataLogger.cpp

```
/*
*****
Brad Donohoo, Steve Gunderson, Jake Erramouspe
ASPS: Automated Security Patrol System
LabDataLogger Class
24 September 2009
*****
*/

#include "Aria.h"
#include "ArExport.h"
#include "ArRobot.h"
#include "LabDataLogger.h"
#include <fstream>
#include <iostream>
#include <cmath>

AREXPORT LabDataLogger::LabDataLogger(ArRobot *robot) : myFunctor(this,
&LabDataLogger::LogData)
{
    myRobot = robot;

    myRobot->addUserTask("LabDataLogger", 50, &myFunctor);
}

AREXPORT LabDataLogger::~LabDataLogger(void)
{
    if(dataFile)
        fclose(dataFile);
    if(coordData)
        fclose(coordData);
}

AREXPORT int LabDataLogger::getSonarAngle(int sonar)
{
    switch(sonar)
    {
        case 0:
            return 90;
        case 1:
            return 41;
        case 2:
            return 15;
        case 3:
            return -15;
        case 4:
            return -41;
        case 5:
            return -90;
        case 6:
            return -145;
        case 7:
            return 145;
    }
}

AREXPORT void LabDataLogger::Localize(void)
```

```

{
    x_offset = myRobot->getSonarRange(7)*0.82;          //sonar7 *
cos(35)
    y_offset = myRobot->getSonarRange(5);

    printf("x_off: %f, y_off: %f\n",x_offset,y_offset);
}

AREXPORT void LabDataLogger::LogData(void)
{
    int sonar, s, sonAngle;
    float radAngle;
    double abs_x, abs_y;
    double x = myRobot->getX();
    double y = myRobot->getY();
    double theta = myRobot->getTh();

    fprintf(dataFile, "\t%.0f\t%.0f\t%.0f", x, y, theta);

    for (sonar=0; sonar <= 7; sonar++)
    {
        s = myRobot->getSonarRange(sonar);
        fprintf(dataFile, "\t%d", s);

        sonAngle = getSonarAngle(sonar);
        radAngle = (theta+sonAngle)*(3.1415926/180.0);

        if (s < 1500)
        {
            abs_x = (x + s*cos(radAngle)) + x_offset;
            abs_y = (y + s*sin(radAngle)) + y_offset;

            fprintf(coordData, "\n%.0f\t%.0f", abs_x, abs_y);
            fflush(coordData);
        }
    }

    fprintf(dataFile, "\n");
    fflush(dataFile);
}

AREXPORT void LabDataLogger::OpenLogFile(const char* logFileName1,
const char* logFileName2)
{
    dataFile = fopen(logFileName1, "a");

    coordData = fopen(logFileName2, "a");
}

AREXPORT void LabDataLogger::CloseLogFile(void)
{
    fclose(dataFile);
    fclose(coordData);
}

```

```

AREXPORT void LabDataLogger::CleanLogfile(const char* logFileName1,
const char* logFileName2)
{
    fclose(dataFile);
    fclose(coordData);

    dataFile = fopen(logFileName1, "w");
    fprintf(dataFile,
"\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n\n", "X", "Y", "Theta",
"sonar0", "sonar1", "sonar2", "sonar3", "sonar4", "sonar5", "sonar6",
"sonar7");

    coordData = fopen(logFileName2, "w");
}

AREXPORT void LabDataLogger::NewLogFile(const char* logFileName1, const
char* logFileName2)
{
    dataFile = fopen(logFileName1, "w");
    fprintf(dataFile,
"\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n\n", "X", "Y", "Theta",
"sonar0", "sonar1", "sonar2", "sonar3", "sonar4", "sonar5", "sonar6",
"sonar7");

    coordData = fopen(logFileName2, "w");
}

```

RobotPosDir.h

```

/*****
Brad Donohoo, Steve Gunderson, Jake Erramouspe
ASPS: Automated Security Patrol System
Follow Wall Class
20 October 2009
*****/

#ifndef ROBOTPOSDIR_H
#define ROBOTPOSDIR_H

#define SPEED 70

class RobotPosDir
{
protected:
    ArRobot *myRobot;

public:

    bool initialSonarAcquired ;

    AREXPORT RobotPosDir::RobotPosDir(ArRobot *robot);
//Constructor
    AREXPORT RobotPosDir::~RobotPosDir();

```

```

    int newSonarData[9];
    const static int minDist = 590;
    const static int maxDist = 610;
    int wallFound;

    AREXPORT void rotateTowardsWall(int MinSonNum);
    AREXPORT int getMinSonarNum(int s0, int s1, int s2, int s3, int
s4, int s5, int s6, int s7);
    AREXPORT int getMinSonar(int s);
    AREXPORT bool findWall(void);
    AREXPORT void getNewSonar(void);
    AREXPORT bool notpointingWall(float);
    AREXPORT void rotate(double);
    AREXPORT void specialDistance();
    AREXPORT void goForward(double velocity);
    AREXPORT void followWall(void);
};
#endif // ROBOTPOSDIR_H

```

RobotPosDir.cpp

```

/*****
    Brad Donohoo, Steve Gunderson, Jake Erramouspe
    ASPS: Automated Security Patrol System
    Follow Wall Class
    20 October 2009
*****/

#include <fstream>
#include <iostream>
#include <math.h>
using namespace std;

#include "Aria.h"
#include "ArRobot.h"

#include "RobotPosDir.h"
#include "ArExport.h"

AREXPORT RobotPosDir::RobotPosDir(ArRobot *robot)
    //Constructor
{
    myRobot = robot;
}
AREXPORT RobotPosDir::~RobotPosDir()
    //Destructor
{
    delete myRobot;
}

AREXPORT int RobotPosDir::getMinSonarNum(int s0, int s1, int s2, int
s3, int s4, int s5, int s6, int s7)
{

```

```

int min1, min2, min3, min4, min5, min6, min7;

min1 = s0 > s1 ? s1 : s0;
min2 = s2 > s3 ? s3 : s2;
min3 = s4 > s5 ? s5 : s4;
min4 = s6 > s7 ? s7 : s6;

min5 = min1 > min2 ? min2 : min1;
min6 = min3 > min4 ? min4 : min3;

min7 = min5 > min6 ? min6 : min5;

if (min7 == s0)
    return 0;
else if (min7 == s1)
    return 1;
else if (min7 == s2)
    return 2;
else if (min7 == s3)
    return 3;
else if (min7 == s4)
    return 4;
else if (min7 == s5)
    return 5;
else if (min7 == s6)
    return 6;
else if (min7 == s7)
    return 7;
}

AREXPOR int RobotPosDir::getMinSonar(int s)
{
    switch(s)
    {
        case 0:
            return newSonarData[0];
            break;
        case 1:
            return newSonarData[1];
            break;
        case 2:
            return newSonarData[2];
            break;
        case 3:
            return newSonarData[3];
            break;
        case 4:
            return newSonarData[4];
            break;
        case 5:
            return newSonarData[5];
            break;
        case 6:
            return newSonarData[6];
            break;
        case 7:
            return newSonarData[7];
    }
}

```

```

        break;
    }
}

/***** find the wall *****/
AREXPORT bool RobotPosDir::findWall()
{
    int diff;
    int minSonar = 5000;
    int minSonNum = 50;
    int BadSonarData = true;
    int bogusSonarcount = 0;

    printf("\nMade it to Find Wall function\n");

    getNewSonar();
    printf("\nSonar0 = %d, Sonar1 = %d, Sonar2 = %d, Sonar3 = %d,
    Sonar4 = %d, Sonar5 = %d, Sonar6 = %d, Sonar7 = %d", newSonarData[0],
    newSonarData[1], newSonarData[2], newSonarData[3], newSonarData[4],
    newSonarData[5], newSonarData[6], newSonarData[7]);

    printf("\nComputing min sonar...\n");
    minSonNum =
    getMinSonarNum(newSonarData[0], newSonarData[1], newSonarData[2], newSonar
    Data[3], newSonarData[4], newSonarData[5], newSonarData[6], newSonarData[7]
    );
    minSonar = getMinSonar(minSonNum);

    printf("\nminSonar = %d, minSonNum = %d", minSonar, minSonNum);

    if (minSonar > (minDist+10))
    {

        printf("\nminSonar is in range. Now we rotate.");

        rotateTowardsWall(minSonNum);

        printf("\nRotated towards wall");
        ArUtil::sleep(3000);

        while(minSonar > (minDist+10))
        {
            goForward(SPEED);

            getNewSonar();
            printf("\nSonar0 = %d, Sonar1 = %d, Sonar2 = %d,
            Sonar3 = %d, Sonar4 = %d, Sonar5 = %d, Sonar6 = %d, Sonar7 = %d",
            newSonarData[0], newSonarData[1], newSonarData[2], newSonarData[3],
            newSonarData[4], newSonarData[5], newSonarData[6], newSonarData[7]);

            printf("\nComputing min sonar...\n");
            minSonNum =
            getMinSonarNum(newSonarData[0], newSonarData[1], newSonarData[2], newSonar
            Data[3], newSonarData[4], newSonarData[5], newSonarData[6], newSonarData[7]
            );
            minSonar = getMinSonar(minSonNum);

```

```

        if (minSonNum != 2 || minSonNum != 3)
        {
            rotateTowardsWall(minSonNum);
        }

        printf("\nminSonar = %d, minSonNum = %d", minSonar,
minSonNum);
    }
    myRobot->setVel(0);

    printf("\nClose enough! Now rotate so we are facing
directly towards wall.");

    printf("\nFacing wall. Now rotate so sensor 5 is pointing
at wall.");

    rotate(90.0);
}
else // rotate so sensor 5 is facing wall
{
    switch(minSonNum)
    {
        case 0:
            printf("\nSonar 0 is min sonar");
            rotate(180.0);
            printf("\nWe are rotating 180 Pos deg");
            break;
        case 1:
            printf("\nSonar 1 is min sonar");
            rotate(131.0);
            printf("\nWe are rotating 131 Pos deg");
            break;
        case 2:
            printf("\nSonar 2 is min sonar");
            rotate(105.0);
            printf("\nWe are rotating 105 Pos deg");
            break;
        case 3:
            printf("\nSonar 3 is min sonar");
            rotate(75.0);
            printf("\nWe are rotating 75 Pos deg");
            break;
        case 4:
            printf("\nSonar 4 is min sonar");
            rotate(49.0); // lock and unlock are in rotate
            printf("\nWe are rotating 49 Pos deg");
            break;
        case 5:
            break;
        case 6:
            printf("\nSonar 6 is min sonar");
            rotate(-55.0); // lock and unlock are in rotate
            printf("\nWe are rotating 55 Neg deg" );
            break;
        case 7:
            printf("\nSonar 7 is min sonar");

```

```

        rotate(-125.0); // lock and unlock are in rotate
        printf("\nWe are rotating 125 Neg deg" );
        break;
    default:
        printf("\nWe are hosed hit default");
        break;
    }
}

wallFound = 1;

ArUtil::sleep(5000);
return true;
}

/***** Get new Sonar Data *****/
AREXPORT void RobotPosDir::getNewSonar (void)
{
    float x;
    float y;
    int sonar;
    int range;

    for (sonar=0; sonar <= 7; sonar++)
    {
        range = myRobot->getSonarRange(sonar);
        newSonarData[sonar] = range;
    }

    myRobot->getEncoderPose();
    x= myRobot->getEncoderPose().getX() ;
    y= myRobot->getEncoderPose().getY() ;
}

/***** What to do if not pointing to the wall *****/
AREXPORT bool RobotPosDir::notpointingWall(float limit)
{
    float diff;
    diff = newSonarData[2] - newSonarData[3];

    if ((abs(diff) <= limit) && (newSonarData[2] <= maxDist) )
    {
        printf("\nWe are now pointing at the wall");
        return false;
    }
    else
        return true;
}

AREXPORT void RobotPosDir::rotateTowardsWall(int minSonNum)
{
    //point towards nearest sonar reading
    switch(minSonNum)
    {
        case 0:

```



```

        printf("\nSonar 0 is min sonar");
        rotate(90.0);
        printf("\nWe are rotating 90 Pos deg");
        break;
    case 1:
        printf("\nSonar 1 is min sonar");
        rotate(41.0);
        printf("\nWe are rotating 41 Pos deg");
        break;
    case 2:
        break;
    case 3:
        break;
    case 4:
        printf("\nSonar 4 is min sonar");
        rotate(-41.0); // lock and unlock are in rotate
        printf("\nWe are rotating 41 Neg deg");
        break;
    case 5:
        printf("\nSonar 5 is min sonar");
        rotate(-90.0); // lock and unlock are in rotate
        printf("\nWe are rotating 90 Neg deg" );
        break;
    case 6:
        printf("\nSonar 6 is min sonar");
        rotate(-145.0); // lock and unlock are in rotate
        printf("\nWe are rotating 145 Neg deg" );
        break;
    case 7:
        printf("\nSonar 7 is min sonar");
        rotate(145.0); // lock and unlock are in rotate
        printf("\nWe are rotating 145 Pos deg" );
        break;
    default:
        printf( "\nWe are hosed hit default");
        break;
}
}

/***** Rotate the Robot *****/
AREXPORT void RobotPosDir::rotate( double degrees )
{
    myRobot->lock(); //Lock robot so paramcan be altered
    myRobot->setDeltaHeading(degrees);
    myRobot->unlock(); // Unlock to allow action

    printf ("\nRotating %3.2f deg", degrees);
}

/***** Set Velocity *****/
AREXPORT void RobotPosDir::goForward(double velocity)
{
    myRobot->lock(); //Lock robot so paramcan be altered
    myRobot->setVel(velocity);
    printf("\nGoing forward");
    myRobot->unlock(); // Unlock to allow action
}

```

```

/***** Distance From Wall while Following *****/
AREXPORT void RobotPosDir::specialDistance( )
{

    getNewSonar();
    if (newSonarData[4] < (minDist-50))
    {
        myRobot->setDeltaHeading(8);
        myRobot->setVel(SPEED);
    }
    if(newSonarData[5]<=minDist) //robot is too close on right side
    {
        if (newSonarData[4] < maxDist) //turn left
        {
            myRobot->setDeltaHeading(5);
            myRobot->setVel(SPEED);
        }
    }
    else if (newSonarData[5]>maxDist) //robot is too far away from
wall
    {
        myRobot->setDeltaHeading(-5);
    }
}

```

```

/***** Follow the Wall *****/
AREXPORT void RobotPosDir::followWall(void)
{
    int minSonar = 5000;
    int minSonNum = 50;

    printf("\nMade it to FOLLOW WALL function\n");

    getNewSonar();

    printf("\nSonar0 = %d, Sonar1 = %d, Sonar2 = %d, Sonar3 = %d,
Sonar4 = %d, Sonar5 = %d, Sonar6 = %d, Sonar7 = %d", newSonarData[0],
newSonarData[1], newSonarData[2], newSonarData[3], newSonarData[4],
newSonarData[5], newSonarData[6], newSonarData[7]);

    minSonNum =
getMinSonarNum(newSonarData[0], newSonarData[1], newSonarData[2], newSonar
Data[3], newSonarData[4], newSonarData[5], newSonarData[6], newSonarData[7]
);
    minSonar = getMinSonar(minSonNum);
    if (minSonar > 3000)
    {
        wallFound = 0;
        return;
    }

    goForward(SPEED);
    ArUtil::sleep(400);
    specialDistance(); // maintain distance from wall

```

```

getNewSonar();

if((newSonarData[2] < maxDist) || (newSonarData[3] < maxDist))
{

    myRobot->lock();
    myRobot->setVel(0);
    myRobot->unlock();

    ArUtil::sleep(1000);
    /*****
    INSIDE CORNER
    *****/
    printf("\nComing to a wall straight ahead -> Rotating 90
deg CCW\n");

    rotate(90);
    ArUtil::sleep(2000);
    myRobot->lock();
    myRobot->setRotVel(0.0);
    myRobot->unlock();
    goForward(SPEED);

}

getNewSonar();

/*****
OUTSIDE CORNER
*****/

// if sonar 3, 4 and 5 are infinite -> assume at at least a 90
degree corner
if((newSonarData[5] > 1000) && (newSonarData[4] > 1000) &&
(newSonarData[3] > 1000))
{

    minSonNum =
getMinSonarNum(newSonarData[0], newSonarData[1], newSonarData[2], newSonar
Data[3], newSonarData[4], newSonarData[5], newSonarData[6], newSonarData[7]
);

    minSonar = getMinSonar(minSonNum);
    if (minSonar > 3000)
    {
        wallFound = 0;
        return;
    }

    printf("\nOutside corner -> Slowly rotating CW\n");

    myRobot->lock(); //Lock robot so paramcan be altered
    myRobot->setRotVel(-12.0);
    myRobot->unlock(); // Unlock to allow action

    getNewSonar();

    while(newSonarData[4]>1200 && newSonarData[3]>1000)

```

```

        {
            minSonarNum =
getMinSonarNum(newSonarData[0],newSonarData[1],newSonarData[2],newSonar
Data[3],newSonarData[4],newSonarData[5],newSonarData[6],newSonarData[7]
);
            minSonar = getMinSonar(minSonarNum);

            printf("\nOutside corner, Sonar3 = %d, Sonar4 = %d",
newSonarData[3],newSonarData[4]);

            if (minSonar > 3000)
            {
                wallFound = 0;
                return;
            }
            getNewSonar();
        }

myRobot->setRotVel(0.0);
}

getNewSonar();
}

```

PatrolRobot2.cpp

```

/*****
Brad Donohoo, Steve Gunderson, Jake Erramouspe
ASPS: Automated Security Patrol System
Patrol Robot
12 November 2009
*****/

#include "Aria.h"
#include "ArNetworking.h"
#include "ASPS_ServerInfoRobot.h"
#include "AnomalyDetect.h"
#include "RobotPosDir.h"
#include "windows.h"
using namespace std;

int main(int argc, char **argv)
{
    bool start = true;

    HWND RemoteDesktop;

    Aria::init();

    // our base server object
    ArServerBase server;

```

```

// set up our parser
ArArgumentParser parser(&argc, argv);
// set up our simple connector
ArSimpleConnector simpleConnector(&parser);
// robot
ArRobot robot;

ArServerSimpleOpener simpleOpener(&parser);

// set up a gyro, if installed
ArAnalogGyro gyro(&robot);

// sonar, must be added to the robot, for teleop and wander
ArSonarDevice sonarDev;
// wall follower
RobotPosDir PosDir(&robot);
// gotoAnomaly action
ArActionGoto gotoAnomaly("goto", ArPose(0.0, 0.0), 200, 400, 150, 7);
robot.addAction(&gotoAnomaly, 90);

// load the default arguments
parser.loadDefaultArguments();

ArClientSwitchManager clientSwitchManager(&server, &parser);

// parse the command line... fail and print the help if the parsing
fails
// or if the help was requested
if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
{
    Aria::logOptions();
    Aria::exit(1);
}

// Set up file directories
char fileDir[1024];
ArUtil::addDirectories(fileDir, sizeof(fileDir),
Aria::getDirectory(),
    "ArNetworking/examples");

// first open the server up
if (!simpleOpener.open(&server, fileDir, 240))
{
    if (simpleOpener.isUserFileBad())
        printf("Bad user/password/permissions file\n");
    else
        printf("Could not open server port\n");
    exit(1);
}

// a key handler so we can do our key handling
ArKeyHandler keyHandler;
// let the global aria stuff know about it
Aria::setKeyHandler(&keyHandler);
// toss it on the robot
robot.attachKeyHandler(&keyHandler);
printf("You may press escape to exit\n");

```

```

// add the sonar to the robot
robot.addRangeDevice(&sonarDev);

// attach services to the server
ASPS_ServerInfoRobot serverInfoRobot(&server, &robot);
//here is where the robot send to the client
ArServerInfoSensor serverInfoSensor(&server, &robot);
ArServerInfoDrawings drawings(&server);
drawings.addRobotsRangeDevices(&robot);

// set up the robot for connecting
if (!simpleConnector.connectRobot(&robot))
{
    printf("Could not connect to robot... exiting\n");
    Aria::exit(1);
}

/*****ANOMALY DETECTION CODE
HERE*****/
AnomalyDetect anomalydetect(&robot);

anomalydetect.OpenCoordFile("coords.txt", "patrol_output.txt");

anomalydetect.ReadData();

anomalydetect.Detect();

/*****/

// log whatever we wanted to before the runAsync
simpleOpener.checkAndLog();
// now let it spin off in its own thread
server.runAsync();

printf("Server is now running...\n");

clientSwitchManager.runAsync();

// start the robot running, true means that if we lose connection the
run thread stops
robot.runAsync(true);

// turn on the motors
robot.comInt(ArCommands::ENABLE, 1);

while(robot.isConnected())
{
    ArUtil::sleep(4000);

    PosDir.wallFound = 0;

    if (start == true)
    {
        anomalydetect.Localize();
        anomalydetect.start_detect = true;
    }
}

```

```

        start = false;
    }

while(1)
{
    PosDir.findWall();

    while(PosDir.wallFound == 1){
        PosDir.followWall();
        if (anomalydetect.anomaly_flag == true)
            break;
    }

    if (anomalydetect.anomaly_flag == true)
        break;
}

    printf("\ngoal: x: %d, y: %d, angle:
%d\n",anomalydetect.anomaly_x,anomalydetect.anomaly_y,anomalydetect.ano
maly_angle);

    robot.lock();
    robot.setVel(0);
    robot.unlock();
    PosDir.rotate(anomalydetect.anomaly_angle);
    ArUtil::sleep(2000);
    PosDir.goForward(100);

while(robot.getSonarRange(2) > 500 && robot.getSonarRange(3) >
500);

    if (robot.getSonarRange(2) < robot.getSonarRange(3) > 500)
    {
        while (robot.getSonarRange(3) > 600)
            PosDir.rotate(5);
    }
    else
    {
        while (robot.getSonarRange(2) > 600)
            PosDir.rotate(-5);
    }

    robot.setVel(0);
    robot.setRotVel(0);
    printf("\nFIRE!!!!!!\n");

    // open remote desktop window
    RemoteDesktop = FindWindow(NULL, "192.168.1.10 - Remote
Desktop");
    SetForegroundWindow(RemoteDesktop);
    ShowWindow(RemoteDesktop, SW_SHOWNORMAL);

    ArUtil::sleep(1000);

    // Send space bar command to remote desktop
    keybd_event( VK_SPACE, 0x39, 0, 0);

```

```

    keybd_event( VK_SPACE, 0xB9, KEYEVENTF_KEYUP, 0);

    // Set alarm flag to notify Police Bot
    serverInfoRobot.alarm_flag = 1;

    ArUtil::sleep(1000);
    // reset alarm flag
    serverInfoRobot.alarm_flag = 0;

    ArUtil::sleep(10000);

    Aria::exit(0);

}

robot.waitForRunExit();
Aria::exit(0);
return 0;
}

```

AnomalyDetect.h

```

/*****
    Brad Donohoo, Steve Gunderson, Jake Erramouspe
    ASPS: Automated Security Patrol System
    AnomalyDetect Class
    12 November 2009
*****/

#ifndef _ANOMALYDETECT_H_
#define _ANOMALYDETECT_H_

#include <cstdlib>
#include <cstdio>
#include <fstream>
#include <iostream>
#include <vector>
using namespace std;

class AnomalyDetect
{
protected:
    ArRobot *myRobot;
    FILE* dataFile;    //File for reading coords
    FILE* outputFile; //File for writing coords
    ArFunctorC<AnomalyDetect> myFunctor;
    vector<int> xcoords;
    vector<int> ycoords;
    double x_offset;
    double y_offset;

public:
    /// Constructor
    AREXPORT AnomalyDetect::AnomalyDetect(ArRobot *robot);

```



```

    /// Destructor
    AREXPORT AnomalyDetect::~AnomalyDetect();

    bool start_detect;

    AREXPORT void QuickSort(int left, int right);
    AREXPORT void PrintLists();
    AREXPORT int getSonarAngle(int s);
    AREXPORT void ReadData(void);
    AREXPORT void SearchCoords(int x, int y, int angle);
    AREXPORT void Localize(void);
    AREXPORT void Detect(void);
    AREXPORT void OpenCoordFile(const char* logFileName1, const char*
logFileName2);

    bool anomaly_flag;
    int anomaly_x;
    int anomaly_y;
    int anomaly_angle;
};

#endif // _ANOMALYDETECT_H_

```

AnomalyDetect.cpp

```

/*****
    Brad Donohoo, Steve Gunderson, Jake Erramouspe
    ASPS: Automated Security Patrol System
    AnomalyDetect Class
    12 November 2009
*****/

#include "Aria.h"
#include "ArExport.h"
#include "ArRobot.h"
#include "AnomalyDetect.h"
#include <fstream>
#include <iostream>
#include <cmath>
#include <string>
#include <vector>
using namespace std;

#define BUF_SIZE          256
#define TOLERANCE        300
#define MAX_ANOMALY      25
#define MIN_COORD_CNT    5

int anomaly_count = 0;

AREXPORT AnomalyDetect::AnomalyDetect(ArRobot *robot) : myFunctor(this,
&AnomalyDetect::Detect)
{
    myRobot = robot;

```

```

myRobot->addUserTask("AnomalyDetect", 50, &myFunctor);

start_detect = false;

//SET ANOMALY FLAG HERE
anomaly_flag = false;
}

AREXPORT AnomalyDetect::~AnomalyDetect(void)
{
    if(dataFile)
        fclose(dataFile);
}

AREXPORT int AnomalyDetect::getSonarAngle(int sonar)
{
    switch(sonar)
    {
    case 0:
        return 90;
    case 1:
        return 41;
    case 2:
        return 15;
    case 3:
        return -15;
    case 4:
        return -41;
    case 5:
        return -90;
    case 6:
        return -145;
    case 7:
        return 145;
    }
}

AREXPORT void AnomalyDetect::QuickSort(int left, int right)
{
    int i = left, j = right;
    int tmp;
    int pivot = xcoords[(left + right) / 2];

    /* partition */
    while (i <= j) {
        while (xcoords[i] < pivot)
            i++;
        while (xcoords[j] > pivot)
            j--;
        if (i <= j) {
            tmp = xcoords[i];
            xcoords[i] = xcoords[j];
            xcoords[j] = tmp;

            tmp = ycoords[i];
            ycoords[i] = ycoords[j];

```

```

        ycoords[j] = tmp;

        i++;
        j--;
    }
};

/* recursion */
if (left < j)
    QuickSort(left, j);
if (i < right)
    QuickSort(i, right);
}

AREXPORT void AnomalyDetect::PrintLists()
{
    int i;
    for(i=0;i<xcoords.size();i++)
        printf("x: %d, y: %d\n",xcoords[i],ycoords[i]);
}

AREXPORT void AnomalyDetect::ReadData(void)
{
    char buf[BUF_SIZE];
    int x, y, i;

    while(!feof(dataFile)){
        fgets(buf,BUF_SIZE,dataFile);
        x = atoi( strtok(buf, "\t" ) );
        y = atoi( strtok(NULL, "\n" ) );
        xcoords.push_back(x);
        ycoords.push_back(y);
    }

    fclose(dataFile);

    QuickSort(0,(xcoords.size()-1));

    PrintLists();
    printf("\n");
}

AREXPORT void AnomalyDetect::SearchCoords(int x, int y, int angle)
{
    int coord_found = 0;
    int start = xcoords.size()/2;

    if(xcoords[start] >= x){
        for (int i=start; i>=0; i--)
        {
            // searching from middle of xcoords array to
xcoords[0]
TOLERANCE))
            if (xcoords[i] < (x+TOLERANCE) && xcoords[i] > (x-
TOLERANCE))
            {
                if (ycoords[i] < (y+TOLERANCE) && ycoords[i] >
(y-TOLERANCE))

```

```

        {
            coord_found++;
        }
    }
}
else{
    for (int i=start; i<xcoords.size(); i++)
    {
        // searching from middle of xcoords array to
xcoords.size()
        if (xcoords[i] < (x+TOLERANCE) && xcoords[i] > (x-
TOLERANCE))
        {
            if (ycoords[i] < (y+TOLERANCE) && ycoords[i] >
(y-TOLERANCE))
            {
                coord_found++;
            }
        }
    }
}

if (coord_found <= MIN_COORD_CNT)
    anomaly_count++;

if (anomaly_count >= MAX_ANOMALY){
    if (anomaly_flag == false)
    {
        printf("\n\nANOMALY FOUND!\n");
        anomaly_flag = true;
        anomaly_x = x - x_offset;
        anomaly_y = y - y_offset;
        anomaly_angle = angle;
    }
}
}

AREXPORT void AnomalyDetect::Localize(void)
{
    x_offset = myRobot->getSonarRange(7)*0.82;        //sonar7 *
cos(35)
    y_offset = myRobot->getSonarRange(5);

    printf("x_off: %f, y_off: %f\n",x_offset,y_offset);
}

AREXPORT void AnomalyDetect::Detect(void)
{
    int sonar, s, sonAngle;
    float radAngle;
    double abs_x, abs_y;
    double x = myRobot->getX();
    double y = myRobot->getY();
    double theta = myRobot->getTh();
}

```

```

    if (start_detect == true)
    {
        for (sonar=0; sonar <= 7; sonar++)
        {
            s = myRobot->getSonarRange(sonar);

            sonAngle = getSonarAngle(sonar);
            radAngle = (theta+sonAngle)*(3.1415926/180.0);

            if (s < 1500)
            {
                abs_x = x + s*cos(radAngle) + x_offset;
                abs_y = y + s*sin(radAngle) + y_offset;

                fprintf(outputFile, "\n%.0f\t%.0f", abs_x,
abs_y);

                fflush(outputFile);

                //check xcoord array for points within
tolerance
                SearchCoords(abs_x,abs_y,getSonarAngle(sonar));
            }
        }
    }
}

```

```

AREXPORT void AnomalyDetect::OpenCoordFile(const char*
logFileName1,const char* logFileName2)
{
    char buf[BUF_SIZE];

    dataFile = fopen(logFileName1, "r");
    fgets(buf,BUF_SIZE,dataFile);

    outputFile = fopen(logFileName2, "w");
}

```

ASPS_ServerInfoRobot.h

```

/*****
    Brad Donohoo, Steve Gunderson, Jake Erramouspe
    ASPS: Automated Security Patrol System
    ASPS_ServerInfoRobot
    30 November 2009
*****/

#ifdef ASPS_SERVERROBOTINFO_H
#define ASPS_SERVERROBOTINFO_H

#include "Aria.h"

```

```

#include "ArServerBase.h"

class ArServerClient;

class ASPS_ServerInfoRobot
{
public:
    /// Constructor
    AREXPORT ASPS_ServerInfoRobot(ArServerBase *server, ArRobot *robot);
    /// Destructor
    AREXPORT ~ASPS_ServerInfoRobot();
    /// The function that sends updates about the robot off to the client
    AREXPORT void update(ArServerClient *client, ArNetPacket *packet);

    int alarm_flag;

protected:
    ArServerBase *myServer;
    ArRobot *myRobot;

    ArFunctor2C<ASPS_ServerInfoRobot, ArServerClient *, ArNetPacket *>
    myUpdateCB;
};

#endif

```

ASPS_ServerInfoRobot.cpp

```

/*****
    Brad Donohoo, Steve Gunderson, Jake Erramouspe
    ASPS: Automated Security Patrol System
    ASPS_ServerInfoRobot
    30 November 2009
*****/

#include "Aria.h"
#include "ArExport.h"
#include "ASPS_ServerInfoRobot.h"
#include "ArServerMode.h"

AREXPORT ASPS_ServerInfoRobot::ASPS_ServerInfoRobot(ArServerBase
*server,
                                                    ArRobot *robot) :
    myUpdateCB(this, &ASPS_ServerInfoRobot::update)
{
    myServer = server;
    myRobot = robot;
    alarm_flag = 0;
    myServer->addData("update",
                    "gets an update about the patrol robot status (you
should request this at an interval)",

```

```

        &myUpdateCB, "none",
        "byte4: alarm_flag, byte4: x, byte4: y", "RobotInfo",
        "RETURN_SINGLE");
    }

AREXPORT ASPS_ServerInfoRobot::~ASPS_ServerInfoRobot()
{
}

AREXPORT void ASPS_ServerInfoRobot::update(ArServerClient *client,
                                           ArNetPacket *packet)
{
    ArNetPacket sending;

    myRobot->lock();

    sending.byte4ToBuf((int)alarm_flag);
    sending.byte4ToBuf((double)myRobot->getX());
    sending.byte4ToBuf((double)myRobot->getY());

    myRobot->unlock();

    client->sendPacketUdp(&sending);
}

```

PoliceRobot.cpp

```

/*****
    Brad Donohoo, Steve Gunderson, Jake Erramouspe
    ASPS: Automated Security Patrol System
    Police Robot
    30 November 2009
*****/

#include "Aria.h"
#include "ArNetworking.h"

#define POLICE_X_OFF    (-450)
#define POLICE_Y_OFF    1100
#define POLICE_TH_OFF  0

/** This class requests continual data updates from the server and
prints them
* out.
*/
class OutputHandler
{
public:
    OutputHandler(ArClientBase *client);
    virtual ~OutputHandler(void);
}

```

```

    /// This callback is called when an update on general robot state
    arrives
    void handleOutput(ArNetPacket *packet);

    int alarm;
    double myX;
    double myY;

protected:

    /// The results from the data update are stored in these variables
    //@{
    int alarm_flag;

    //@}
    ArClientBase *myClient;

    /** These functor objects are given to the client to receive updates
    when they
    * arrive from the server.
    */
    //@{
    ArFunctor1C<OutputHandler, ArNetPacket *> myHandleOutputCB;
    //@}

};

OutputHandler::OutputHandler(ArClientBase *client) :
    myClient(client),
    myHandleOutputCB(this, &OutputHandler::handleOutput)
{
    alarm = 0;

    /* Add a handler for anomaly position, and request it to be called
    every 100 ms */
    myClient->addHandler("update", &myHandleOutputCB);
    myClient->request("update", 100);
}

OutputHandler::~OutputHandler(void)
{
    /* Halt the request for data updates */
    myClient->requestStop("update");
}

void OutputHandler::handleOutput(ArNetPacket *packet)
{
    alarm_flag = (int) packet->bufToByte4();
    myX = (double) packet->bufToByte4();
    myY = (double) packet->bufToByte4();

    if (alarm_flag == 1)
    {
        printf("Alarm received!\n"
            "X: %lf, Y: %lf\n", myX, myY);
    }
}

```



```

        fflush(stdout);

        myClient->requestStop("update");

        alarm = 1;
    }
}

/* Key handler for the escape key: shutdown all of Aria. */
void escape(void)
{
    printf("esc pressed, shutting down aria\n");
    Aria::shutdown();
}

int main(int argc, char **argv)
{
    /* Aria initialization: */
    Aria::init();

    // set up our parser
    ArArgumentParser parser(&argc, argv);
    // set up our simple connector
    ArSimpleConnector simpleConnector(&parser);
    // robot
    ArRobot robot;
    // sonar, must be added to the robot, for teleop and wander
    //ArSonarDevice sonarDev;

    // Create our client object

    ArClientBase client;

    ArClientSimpleConnector clientConnector(&parser);

    // load the default arguments
    parser.loadDefaultArguments();

    /* Check for -help, and unhandled arguments: */
    if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
    {
        Aria::logOptions();
        exit(0);
    }

    /* Connect our client object to the remote server: */
    if (!clientConnector.connectClient(&client))
    {
        if (client.wasRejected())
            printf("Server '%s' rejected connection, exiting\n",
client.getHost());
        else
            printf("Could not connect to server '%s', exiting\n",
client.getHost());
        exit(1);
    }
}

```

```

}

printf("Connected to server.\n");

/* Create a key handler and also tell Aria about it */
ArKeyHandler keyHandler;
Aria::setKeyHandler(&keyHandler);
// toss it on the robot
robot.attachKeyHandler(&keyHandler);
printf("You may press escape to exit\n");

// add the sonar to the robot
//robot.addRangeDevice(&sonarDev);

// set up the robot for connecting
if (!simpleConnector.connectRobot(&robot))
{
    printf("Could not connect to robot... exiting\n");
    Aria::exit(1);
}

// run the client in a background thread
client.runAsync();

// start the robot running, true means that if we lose connection the
run thread stops
robot.runAsync(true);

// turn on the motors
robot.comInt(ArCommands::ENABLE, 1);

// create output handler object that will receive the alarm
OutputHandler outputHandler(&client);

while(robot.isConnected())
{
    if (outputHandler.alarm == 1)
    {
        robot.lock();
        robot.setVel(100);
        robot.unlock();

        ArUtil::sleep((outputHandler.myX - POLICE_X_OFF)*10);

        robot.lock();
        robot.setVel(0);
        robot.unlock();

        robot.lock();
        robot.setDeltaHeading(-90);
        robot.unlock();

        robot.lock();
    }
}

```

```
        robot.setVel(100);
        robot.unlock();

        ArUtil::sleep(abs((POLICE_Y_OFF -
outputHandler.myY)*10));

        robot.lock();
        robot.setVel(0);
        robot.unlock();

        Aria::exit(0);
    }
}

Aria::shutdown();
return 0;
}
```