The Report Committee for Nancy Elizabeth Stokes
Certifies that this is the approved version of the following report:

Mobile Electronic Dispensary System

Committee:

Suzanne Barber, Supervisor

Thomas Graser

Mobile Electronic Dispensary System

by

Nancy Elizabeth Stokes, B.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2012

Abstract

Mobile Electronic Dispensary System

Nancy Elizabeth Stokes, MSE

The University of Texas at Austin, 2012

Supervisor: Suzanne Barber

The Mobile Electronic Dispensary System (MEDS) is an indoor medical dispensary system where robots locate and travel to patients within a grid in order to deliver medication or other medical supplies based on a predefined schedule. For older people or individuals with physical or mental disabilities, it is important to ensure that medications are taken as prescribed. Missing or mixing dosages can cause unwanted and even harmful consequences. As individuals grow older or battle disabilities, it is expected that adhering to their medicine regimen will be a daily challenge without the assistance of a fulltime caregiver. Therefore, to assist individuals in maintaining their independence, MEDS ensures the proper medicine is dispensed to the patient at the prescribed time and dosage. At the core of MEDS is a scheduler that maintains the medicines to be dispensed, including the times and dosages. Once a scheduled time arrives to deliver medicine to a patient, MEDS instructs the appropriate robot to wake up, locate the patient within a defined grid, and then travel to the patient and deliver the medicine. Upon receiving the delivery, the patient will accept the medicine physically and then update their mobile device, informing MEDS that the medicine was successfully delivered. At this time, the

robot will return to its home base within the grid. The patients are within the confines of a building where GPS is not a viable solution to track items to pinpoint accuracy. Therefore, an indoor location based system with beacons and listeners are required in order to define a grid and enable robots to locate and travel to the patient. This paper defines and details the programs, database, algorithms, and hardware of MEDS using the Cricket Indoor Location System and iRobot Creates.

Table of Contents

List of Tables

List of Figures

List of Source Code

System Overview

The Mobile Electronic Dispensary System (MEDS) is an indoor medical dispensary system where robots locate and travel to patients within a grid in order to deliver medication or other medical supplies based on a predefined schedule. For older people or individuals with physical or mental disabilities, it is important to ensure that medications are taken as prescribed. Missing or mixing dosages can cause unwanted and even harmful consequences. As individuals grow older or battle disabilities, it is expected that adhering to their medicine regimen will be a daily challenge without the assistance of a fulltime caregiver. Therefore, to assist individuals in maintaining their independence, MEDS ensures the proper medicine is dispensed to the patient at the prescribed time and dosage. At the core of MEDS is a scheduler that maintains the medicines to be dispensed, including the times and dosages. Once a scheduled time arrives to deliver medicine to a patient, MEDS instructs the appropriate robot to wake up, locate the patient within a defined grid, and then travel to the patient and deliver the medicine. Upon receiving the delivery, the patient will accept the medicine physically and then update their mobile device, informing MEDS that the medicine was successfully delivered. At this time, the robot will return to its home base within the grid. The patients are within the confines of a building where GPS is not a viable solution to track items to pinpoint accuracy. Therefore, an indoor location based system with beacons and listener are required in order to define a grid and enable robots to locate and travel to the patient. This paper defines and details the

programs, database, algorithms, and hardware of MEDS using the Cricket Indoor Location System and iRobot Creates.

Below is a system overview graphic. The system is broken down into segments and described at a high level. The details of the system will follow in the subsequent sections.

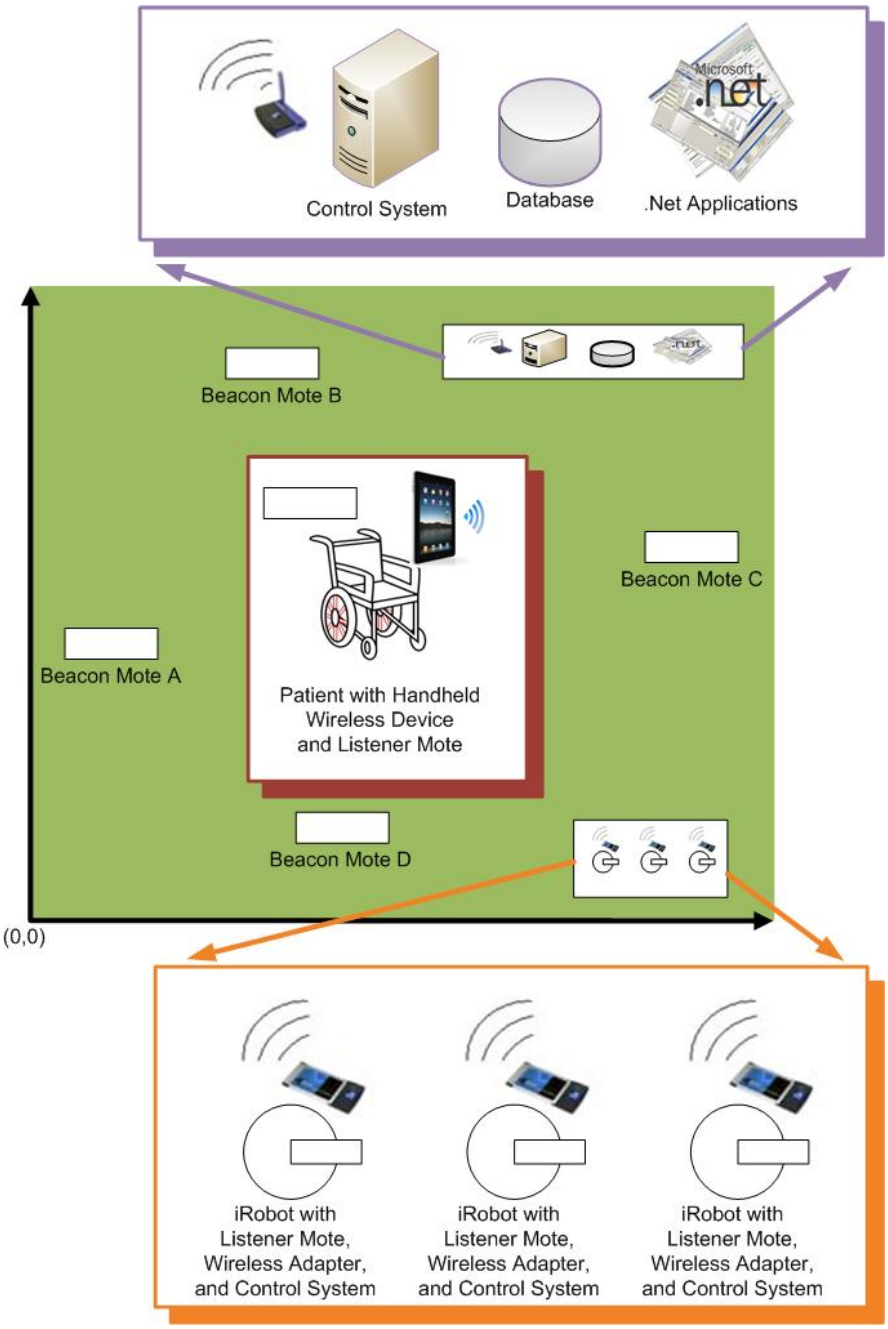


Figure 1 - Overall MEDS System

System Segment	Description
Control System	The Control System segment of MEDS includes a main computer running Microsoft Windows and a .NET environment. The database is Microsoft SQL Server 2008. The system is connected to the Internet via a modem and a wireless router is attached, enabling the additional system to connect wirelessly.
Grid	The Grid segment is a section of the patient's home that is defined using x- and y-coordinates. Within the grid, there are Beacon Motes affixed to the ceiling at certain coordinates. The Grids are the areas in which the robots and patients travel through and define the Indoor Location System boundaries.
Robots	The Robots segment is a series of iRobot Creates which are affixed with a control system, a wireless adapter, and a listener mote. Together, these aspects enable the Robots to communicate with the Control System and the Indoor Location System.
Patient	The Patient segment represents the physical location of the patient within their home. The Patient includes a wireless handheld device in order to communicate with the Control System. The listener mote attached to the patient's wheelchair, or other physical device, is used to communicate with the Control System and the Indoor Location System.

Table 1 - Overview System Segments

Indoor Location System

An Indoor Location System (ILS) is a system that provides positioning of items through coordinates in an indoor setting. The U.S. Global Positioning System (GPS) is a widely used navigation system that provides both location and time position. However, due to the requirement that three or more satellites must be visible in order to determine precise location coordinates, the system is not effective for very specific indoor location calculations. [3] In order for MEDS to perform properly, an alternative system must be defined that works in relatively small confined areas such as a home or an apartment.

The ILS selected for MEDS combines a Cricket Indoor Location System [1] with an iRobot Create running Player Server. An iRobot Create [2] equipped with a Cricket mote determines its own location within a coordinate system and maneuvers to a location defined by the system. In order to determine the current location of the iRobot Create, a process to extract the distance data from the Cricket system was developed. Once extracted, a trilateration [5] computation to calculate the x- and y- coordinates of the iRobot Create is performed. Based on this location, the iRobot Create continues to move closer to the destination through a series of checks and commands.

The following sections describe the hardware aspects of MEDS' ILS and how the entities work together to determine locations and process movements to a destination.

Hardware

iRobot Create

iRobot Create is a programmable robot which has two powered wheels, 32 built-in sensors and an expansion port for incorporating a command module. The iRobot Create, designed for educational purposes, is fashioned after the Roomba vacuum cleaner. The ability to program the iRobot is made possible through the iRobot Open Interface. [2]

The iRobot Create connects to a Cricket mote formatted as a listener through a USB to Serial cable. Additionally, the iRobot Create connects to a laptop through a USB to Serial cable in order to receive movement commands, discussed in a later section.

Figure 2 - iRobot Create

Cricket Indoor Location System

In order to determine location in an indoor setting, the Cricket Indoor Location System was selected. Developed at MIT, the system combines radio frequencies and ultrasonic transmissions to determine distances based on time-to-send results. [1]

Cricket requires the installation and setup of two types of motes: beacons and listeners. Cricket beacons are positioned on a ceiling facing down and Cricket listeners are attached to movable items, such as an iRobot Create, facing up. Cricket beacons send out RF signals and ultrasonic pulses which are received by the listener. Using the time differences between the RF and ultrasonic transmissions, the listener determines its position within the coordinate system.

The process of setting up and programming the motes is detailed in the Grid Establishment and Setting up Motes sections below.

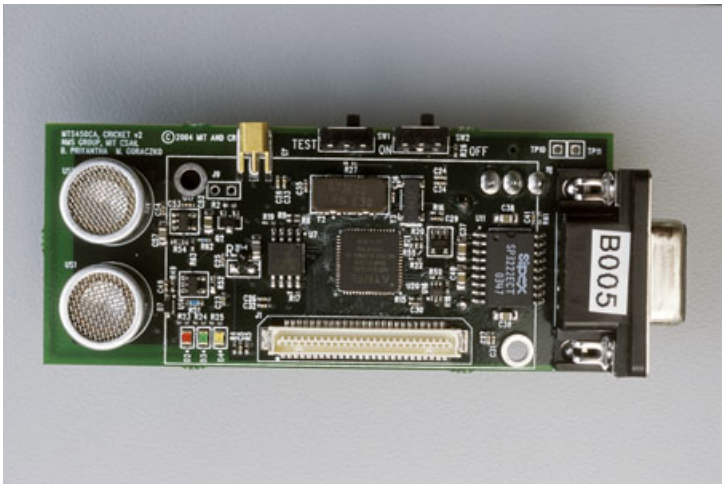


Figure 3 - Cricket Mote

Coordinate System

The first step in setting up a patient's home with MEDS is to define the areas that will be utilized for the system. The ILS is then defined and installed based on the area(s) selected.

Below is an example of a simple ILS layout. This includes one open room with four (4) Cricket mote beacons, one (1) iRobot Create, and a main controller system.

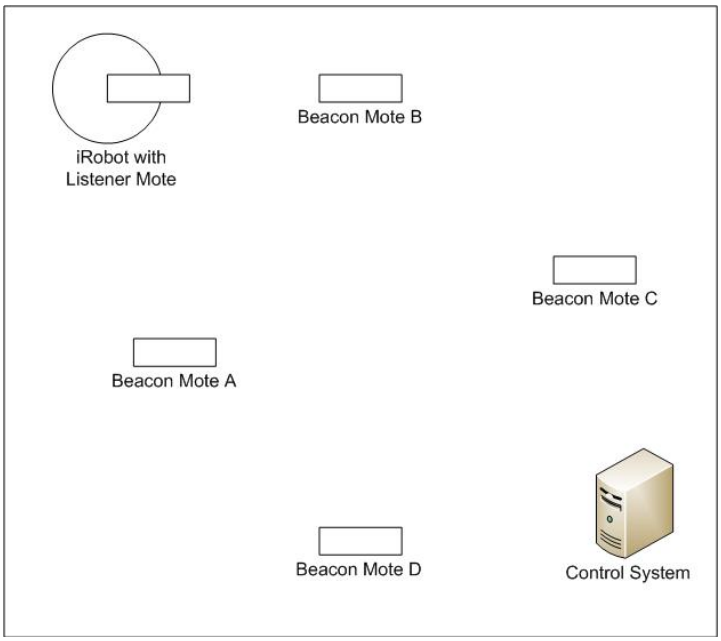


Figure 4 – Simple MEDS Layout

It is important to use four (4) or more beacons and place them in a non-circular layout. The reasons for this are discussed in more detail in the Location Calculations section below.

Once the room is defined and the items “placed” within the room, a Cartesian coordinate system must be established. The first step is selecting an origin point defined as (0,0). The next step is to determine the coordinates of each of the beacons and robot within the grid based on the origin. Using either the metric or the imperial system for measurements, the positions of each is calculated.

Below is an updated representation of the simple ILS layout with coordinates.

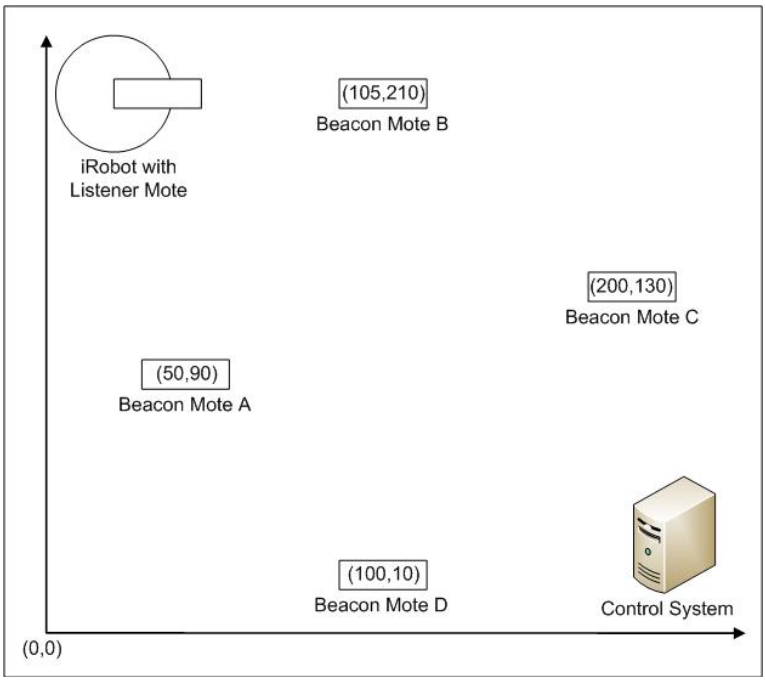


Figure 5 - Simple MEDS Layout with Coordinates

Now that the area is defined and the location coordinates calculated, the motes are ready to be programmed.

Setting up Motes

Based on our simple ILS example, there are four (4) beacon motes and one (1) listener mote that need to be setup.

For each mote, there are important items that are programmed into its configuration. In order to program the motes, they are connected to a system with a serial port or USB to serial cable. Using Minicom (a text-based modem control and terminal emulation program for Unix-like operating systems) the motes are programmed through commands. The items, descriptions, and example commands are listed below. [4]

Item	Description	Example Command
Space ID	Unique name for the group of beacons and listener(s)	P SP Space1
Mode	Listener or Beacon	Listener = P MD 1 Beacon = P MD 2
Coordinates	(x,y) location of beacon mote	P PC 1 2 3
Units	Inches or Metric	Metric = P UN 1 1 Inches = P UN 2 2
Unique ID	The unique identifier that is assigned to the mote by Cricket	

Table 2 - Mote Programming Commands

Note: The Space ID for all the motes that make up a system must be programmed with the same name in order for them to communicate with each other.

As the MEDS system in this paper continues to evolve, it will become apparent that additional listener motes will need to be installed and setup. The first example of further motes required is in respect to the patient. In order for the system to locate the patient within the ILS, a listener mote must be placed at or near the patient at all times. For the purpose of this paper, it is assumed that the patient is in a wheelchair and that the system sends an iRobot to the wheelchair's location for medicine delivery. The second requirement for additional motes is due to the setup of multiple spaces within a patient's home. Multiple spaces are addresses in further detail in the next section. But in reference to the additional motes required, each iRobot and wheelchair will require one (1) listener mote for each ILS space established within the home.

Multiple Spaces

The above sections describe the grid and mote setup for a simple, one-room area. However, it would be more applicable to assume that a patient will be in more than one place within their home. A better scenario would be to map a grid that includes multiple areas, such as a living room, bedroom, and kitchen. Rooms are separated by walls and doors which limit the ability for the motes' transmissions to be received outside of the direct area. Therefore, a separate ILS space must be created for each room. This requires a complete set of beacon motes for each room, programmed with distinctive space IDs.

To accomplish this, the same steps must be taken in each room individually. The beacon motes must be placed within the room and a unique coordinate system defined for each area. Below is a graphic exemplifying a multiple room setup with unique coordinate systems.

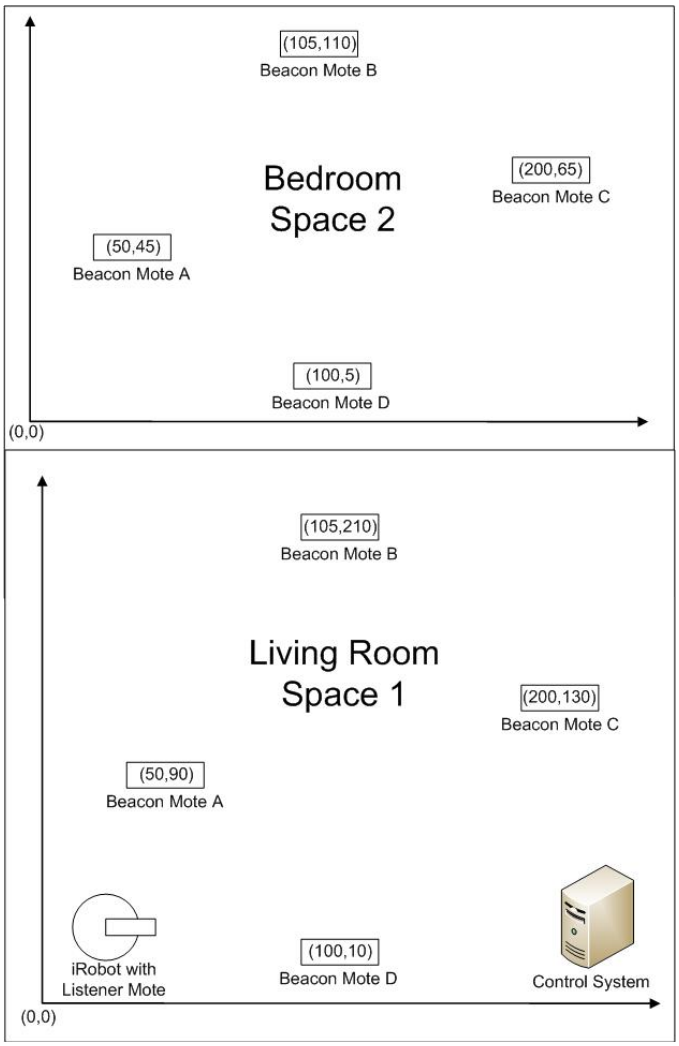


Figure 6 - MEDS Multi-Space Layout

As discussed above, each iRobot and the patient's wheelchair must be setup with a listener mote for each created ILS space. The reason for this is that a listener mote can only receive transmissions from the beacon motes programmed to their space ID. Therefore, if the patient is in the living room (Space 1), their listener mote for Space 2 will not be able to receive transmissions from the beacons in the living room. They do not "know about" each other. In addition, the motes from the bedroom (Space 2) will be too far away to be received.

Special programming is required to enable an iRobot to maneuver from one space to another. Based on the patient's distinct MEDS site, a "hallway" coordinate must be established with a "from" and "to" space ID. The hallway coordinate is a point within a space's grid that an iRobot must travel to in order to transfer into an adjacent space.

Below is an example hallway between the living room (Space 1) and the bedroom (Space 2) of our example.

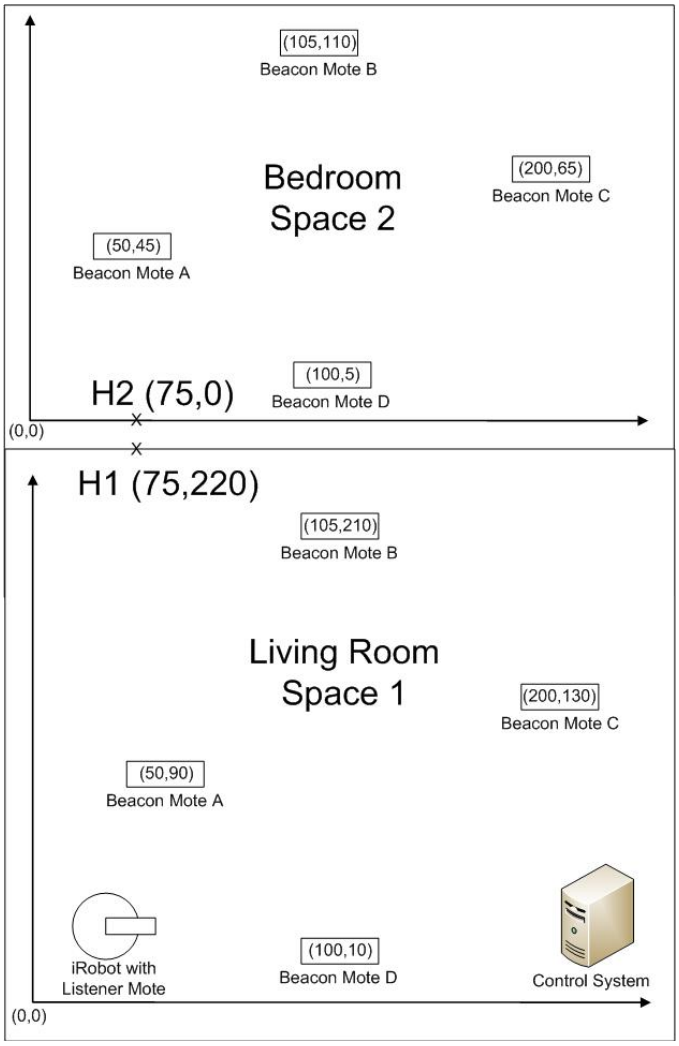


Figure 7- MEDS Multi-Space Layout with Hallway Points

Assume that the iRobot is in the living room and its destination is within the bedroom's space. In order to reach its destination, the iRobot must maneuver to the Space 1 hallway point H1 at (75,220) using its listener mote programmed to Space 1. After the iRobot has traveled a bit further into the bedroom, the Space 2 listener mote begins to receive transmissions from the Space 2 beacon motes. Space 2 now becomes the active space and the iRobot maneuvers to its desired location within the room. Upon returning, the same steps are processed in reverse. To exit Space 2 and enter Space 1, the iRobot travels to Space 2's hallway point H2 at (75,0). Additional information is provided in subsequent sections pertaining to location calculations and movements of the iRobot.

Location Calculations

The sections above described the MEDS' hardware, example layouts, and introduced the notion of an iRobot traveling to a location for medicine delivery. This section provides information on how the system determines the location of the robots and patient. The next section describes how, based on the location calculations, the iRobots are maneuvered.

As detailed above, an ILS space is setup by affixing four (4) or more beacon motes to the ceiling. Next, the coordinate system is defined and each mote is programmed appropriately. Each mote is given a label within the space for programming purposes, for example A-D. Below is an example of a set of beacon motes on a ceiling with example coordinates and labels.

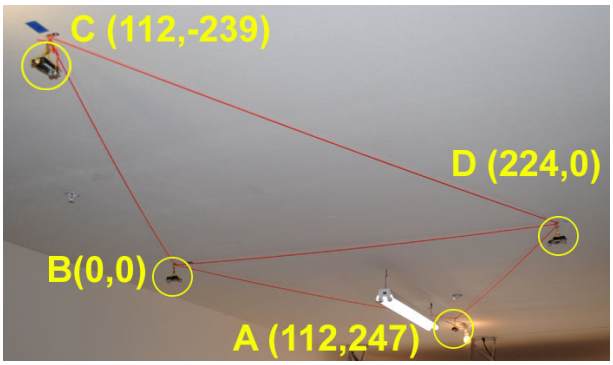


Figure 8 - Mote Locations

With the beacons configured in the coordinate system and hung in place, the location of a listener mote within the space is possible. The first step is to attach a mote to an iRobot and measure the vertical distance of this mote to the beacon motes. This measurement represents the "height" of the space.

When the system is active and the beacon motes are transmitting, the listener motes will repeatedly report the distance to each beacon (DB). This calculation performed by the motes is determined through differences in the time it takes radio frequency and ultrasonic pulses to reach the listener mote.

At this point, there are two (2) known distances – the "height" and the "distance to beacon". The next step is to calculate the horizontal distance of the listener mote to each of the beacons. This is accomplished through Pythagorean's Theorem ($a^2 + b^2 = c^2$) to calculate the sides of a right triangle. Replacing "a" with the "height" measurement and "c" with the

"distance to beacon" value, the third side of the triangle ("radius") can be calculated.

An example of this is shown below.

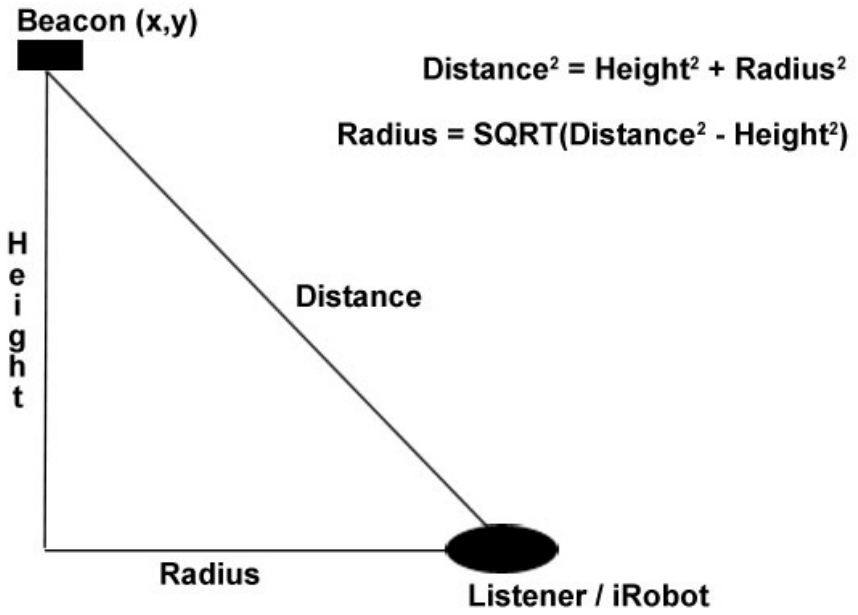
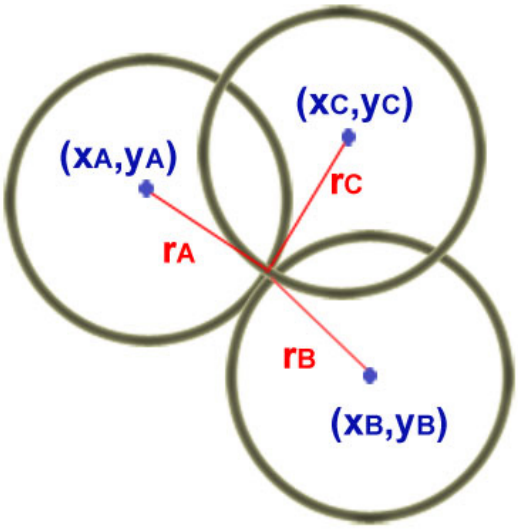


Figure 9 - Radius Calculation

This process is repeated for each of the beacons in which the listener shares a space. Once all of horizontal distances are measured, they then represent the radius of a circle with the center point being its corresponding beacon coordinates. Using the equation of a circle with the center point of the represented by the beacon coordinates, the point where the circles intersect is calculated. This print represents the location of the listener within the coordinate system.

Even though four (4) beacon motes are setup, only three (3) are used for each location calculation. The beacon mote with the largest distance reading from the listener is omitted.

This process of determining the location of the listener is referred to as a trilateration calculation [5]. A pictorial example is shown below.



$$x = (r_A^2 - r_C^2 + x_C^2) / (2 * x_C)$$

$$y = (r_A^2 - r_B^2 - x^2 + (x-x_B)^2 + y_B^2) / (2 * y_B)$$

Figure 10 - Trilateration Calculation

The first step in calculating trilateration is to get the distances from each beacon to the listener. The code below reads all the output from the listener mote and saves the distance measurement to a file for the appropriate beacon mote. The program code below looks through the listener output for the text 'DB='. If this string is in the output line, then the line represents a distance calculation made from the listener to one of

the beacons. If the string exists, the next step is to determine which beacon the distance measurement represents. Because each mote has a unique identifier, the program compares the output text to the different beacon IDs. Once the matching mote is found, the distance is written to the corresponding file. This *RecordDistances()* process runs repeatedly for each listener mote when its location is needed. This includes when a robot is actively traveling to deliver medicine to a patient and anytime a patient's location is required.

```
void RecordDistances()
{
        int i = 0, j = 0;
        size_t found;
        ofstream A, B, C, D;
        string distance;
        string text;

        while (i > -1)
        {
                cin >> text;
                for (int k = 0; k < text.length(); k++)
                        text[k] = toupper (text[k]);
                found=text.find("DB=");
                if (int(found) > 0)
                {
                        for (j=3;j<10;j++)
                        {
                                if (text[found+j] == ',')
                                {
                                        distance = text.substr(int(found)+3,j -3 );
                                        j = 10;
                                }
                        }

                        found=text.find("00:00:87");
                        if (int(found) > 0)
                        {
                                cout << " writing A ";
                                A.open ("A.txt");
                                A << distance;
                                A.close();
                        }
```

Source Code 1 - RecordDistances()

```
                        else
                        {
                                found=text.find("00:00:75");
                                if (int(found) > 0)
                                {
                                        cout << " writing B ";
                                        B.open ("B.txt");
                                        B << distance;
                                        B.close();
                                }
                                else
                                {
                                        found=text.find("00:00:1E");
                                        if (int(found) > 0)
                                        {
                                                cout << " writing C ";
                                                C.open ("C.txt");
                                                C << distance;
                                                C.close();
                                        }
                                        else
                                        {
                                                found=text.find("00:00:50");
                                                if (int(found) > 0)
                                                {
                                                        cout << " writing D ";

                                                        D.open ("D.txt");
                                                        D << distance;
                                                        D.close();
                                                }
                                        }
                                }
                        }
                }
                i++;
        }
}
```

Source Code 1 - RecordDistances(), cont.

Below is the *CurrentLocations()* function that calculates a listener mote's current location using the trilateration calculation described above. The example is using the Space 1 beacon coordinates and a ceiling height of 257 for easier representation. This process is executed after every movement of a robot to determine where it is after the movement. The function calls *GetDist()*, shown below, which reads the current readings from files written to in *RecordDistances()*.

```
void CurrentLocations()
{
	double rA, rB, rC, rD; // radii
	double distA, distB, distC, distD; // current dist of beacon from reader
	double ceiling = 257; // ceiling height;
	int xA = 50, yA = 90, xB = 105, yB =21 0; // beacon coordinates
	int xC = 200, yC = 130, xD = 100, yD = 10; // beacon coordinates
	// get radii of each beacon
	distA = GetDist('A');
	if (distA > 500)
		distA = lastdistA;
	else
		lastdistA = distA;
	cout << " distA= " << distA;
	rA = sqrt(fabs(distA*distA - ceiling*ceiling));
	cout << " rA= " << rA;
	distB = GetDist('B');
	if (distB > 500)
		distB = lastdistB;
	else
		lastdistB = distB;
	cout << " distB= " << distB;
	rB = sqrt(fabs(distB*distB - ceiling*ceiling));
	cout << " rB= " << rB;
	distC = GetDist('C');
	if (distC > 500)
		distC = lastdistC;
	else
		lastdistC = distC;
	cout << " distC= " << distC;
	rC = sqrt(fabs(distC*distC - ceiling*ceiling));
	cout << " rC= " << rC;
	distD = GetDist('D');
	if (distD > 500)
		distD = lastdistD;
	else
		lastdistD = distD;
	cout << "distD= " << distD;
	rD = sqrt(fabs(distD*distD - ceiling*ceiling));
	cout << " rD= " << rD << endl;
	// trilateration calculation
	if (y > 0) // if listener/robot is on A half of the grid
	{
		x = (rB*rB-rD*rD+xD*xD)/(2*xD);
		y = (rB*rB-rA*rA-x*x+(x-xA)*(x-xA)+yA*yA)/(2*yA);
		cout << " using A, x= " << x << " y= " << y << endl;
	}
	else // if listener/robot is on C half of the grid
	{
		x = (rB*rB-rD*rD+xD*xD)/(2*xD);
		y = (rB*rB-rC*rC-x*x+(x-xC)*(x-xC)+yC*yC)/(2*yC);
		cout << " using C, x= " << x << " y= " << y << endl;
	}
}
```

Source Code 2 – CurrentLocations()

The code above calculates the listener's location using the distance to the listener from each of the beacons in the space. The distance measurements are read from a file with the same name of the beacon. The *GetDist()* function below reads the appropriate file and returns the requested distance measurement.

```
// get the distance readings from each of the beacons
double GetDist(char filename)
{
        double distance = 0;
        char line[256];
        FILE *inputFilePtr;

        switch ( filename )
        {
                case 'A':
                        inputFilePtr= fopen("A.txt", "r");
                        break;
                case 'B':
                        inputFilePtr = fopen("B.txt", "r");
                        break;
                case 'C':
                        inputFilePtr = fopen("C.txt", "r");
                        break;
                case 'D':
                        inputFilePtr = fopen("D.txt", "r");
                        break;
        }

        fgets(line,256,inputFilePtr);
        distance = atof(line);
        fclose(inputFilePtr);

        return distance;
}
```

Source Code 3 - GetDist()

Robot Movements

The goal of MEDS is to deliver medication to a patient based on a schedule. In order to achieve the delivery within the ILS spaces, the patient's location must be determined and an iRobot sent for delivery. At this point, decisions must be made as to what points within the system the robot needs to traverse in order to reach the patient. This area is the grid

definition and path determination which will be covered in the next section. For the purpose of this section, the path the robot will take is assumed to be known. Under this assumption, this section describes how the iRobot uses its current location calculation to determine the next movement to make.

Each step of the way, the robot knows its last location, its current location, and the next point that it is attempting to reach. The process is a continual set of calculations, comparisons, and movement commands to the robot. The iRobot is commanded to turn a certain degree, either clockwise or counterclockwise, for a certain amount of time at a certain forward speed. The degree, direction, time, and speed are sent to the robot based on what the program determines will direct it closer to the next point in the path. Once the goal point is reached within a certain degree of error, the next point in the path is retrieved and the process repeated until the final goal point is reached.

Below is the algorithm that is used in the program to determine what movement command to send to the robot. At each point in the program, there are eight (8) variables that are maintained. These are described in the table below.

Variable	Description	How Determined
prevX	x-coordinate of robot's previous calculated position	Calculated
prevY	y-coordinate of robot's previous calculated position	Calculated
currX	x-coordinate of robot's current calculated position	Calculated
currY	y-coordinate of robot's current calculated position	Calculated
nextX	x-coordinate of next point the robot is travelling to	Path generation
nextY	y-coordinate of next point the robot is travelling to	Path generation
goalX	x-coordinate of the patient's position	Patient location
goalY	y-coordinate of the patient's position	Patient location
xDiff	Absolute difference between currX and prevX	Calculated
yDiff	Absolute difference between currY and prevY	Calculated

Table 3 - Movement Algorithm Variables

Using the variables above, the pseudo code below is processed in order to move the robot through the path points to the patient's location.

```
foreach point in the path
{
        while currX != nextX and currY != nextY
        {
                if currX < nextX
                        if currY < nextY
                                if (xDiff and yDiff less than previous)
                                        continue on current course
                                else if (xDiff more)
                                        turn clockwise and move
                                else
                                        turn counterclockwise and move
                                else
                                        if (xDiff and yDiff less than previous)
                                                continue on current course
                                        else if (xDiff more)
                                                turn counterclockwise and move
                                        else
                                                turn clockwise and move
                        else
                                if currY < nextY
                                        if (xDiff and yDiff less than previous)
                                                continue on current course
                                        else if (xDiff more)
                                                turn counterclockwise and move
                                        else
                                                turn clockwise and move
                                else
                                        if (xDiff and yDiff less than previous)
                                                continue on current course
                                        else if (xDiff more)
                                                turn clockwise and move
                                        else
                                                turn counterclockwise and move
        }
}
```

Source Code 4 - Movement Algorithm Pseudo Code

Below is an example of a potential process flow that a robot could take from one point to the next.

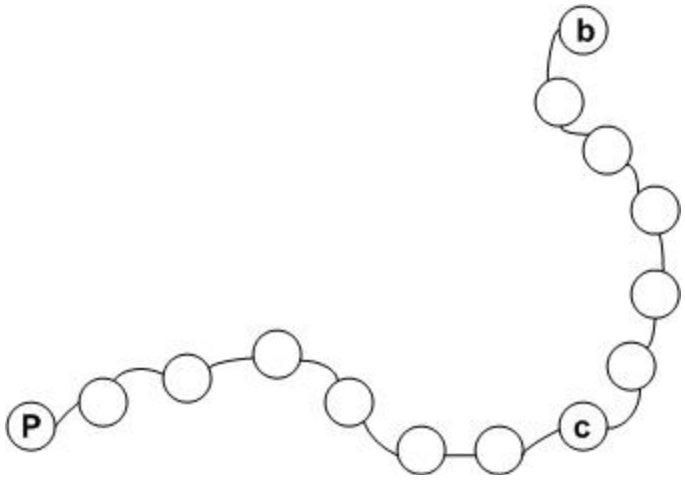


Figure 11 - Robot Movements between Points

The graphic above represents a subsection of a path that crosses from point “P” to point “b” through point “c”. This path is an actual subset in the example grid in the next section. The robot, in this example, travels from point “P” to point “c” in seven (7) steps. Once “c” is reached, the program provides the robot with the next goal point of “b”. The robot makes six (6) movements in order to reach “b”.

Paths

Grid and Path Definition

At this point the patient’s home has been organized into spaces, mote locations determined, and coordinate systems defined. The next step is to review each space and determine the points within the room that are available for the iRobot to travel. These points will include areas that are not occupied by furniture, cabinets, or other impassible sections. The individual points are given (x,y) coordinates within the particular space’s coordinate system. Defining all the

accessible points within the space defines the space's grid. Below is an example of a living room space with its grid points defined.

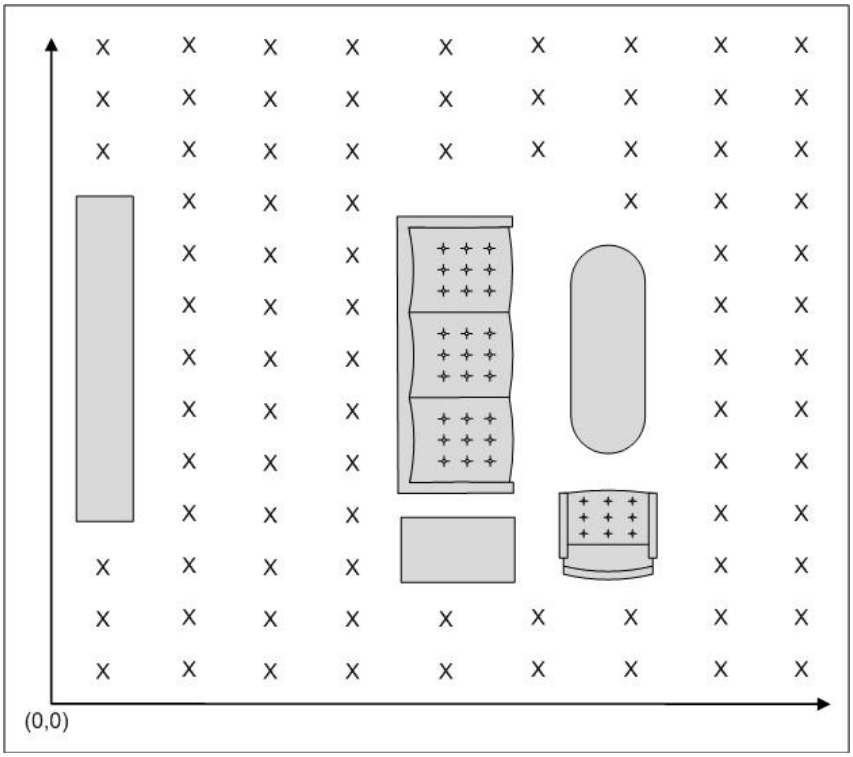


Figure 12 - Grid Points Available for Travel

Once the grid is established for a space, each point's adjacent point(s) is determined. In other words, if a robot is at a particular point in the grid, what points are available for it to travel to next? With this information assigned to each point in the grid, it is then possible to determine a path that a robot would take to travel throughout the space.

To demonstrate this more specifically, each X mark in the grid has been assigned a unique ID. Additionally, the adjacent points of each have been noted with an arrow.

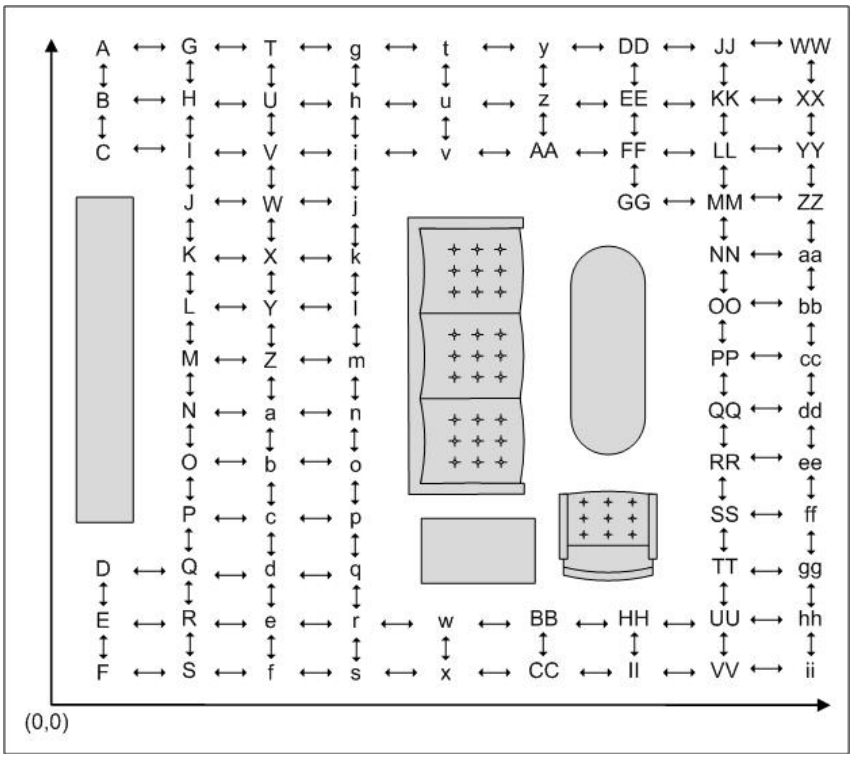


Figure 13 – Named Grid Points with their Adjacent Points

Below is a table listing various points in our example grid and their adjacent points.

Point	Coordinate	Adjacent Points
A	(20,215)	B, G
H	(80,210)	B, G, I, U
Q	(95,15)	d, p, r
CC	(140,5)	x, BB, II
Aa	(220,140)	NN, ZZ, bb

Table 4 - Grid Points with Adjacent Points

Using our example grid and adjacent points, the table below shows two (2) possible paths that can be taken to get from point A to point RR. The process of determining the possible paths, as well as the optimum path, is discussed in the Path Algorithm section below.

Path	# Steps
A -> G -> T -> g -> t -> y -> DD -> JJ -> KK -> LL -> MM -> NN -> OO -> PP -> QQ -> RR	15
A -> B -> H -> I -> V -> W -> j -> k -> l -> m -> n -> o -> p -> q -> r -> w -> BB -> HH -> UU -> TT -> RR	20

Table 5 - Potential Paths from AA to ii

In order for the robot to travel from point A to point B, it must maneuver through the available points within the grid. Therefore, it must go around the couch in the middle of the room. The first path takes 15 steps and the second path takes 20 steps.

Path Algorithm

Once the complete grid system is defined, the next step is to select and implement a path algorithm. This algorithm will search through the potential options available to travel from a starting to a finishing point. These types of algorithms are referred to as "Shortest Path" [6] or "Search" algorithms. There has been extensive research and development on path finding algorithms. Therefore, this paper will merely describe the process of determining the path as well as select and describe the algorithm to be used by MEDS.

Finding a Path

In MEDS, once the Scheduler program has determined that a delivery is to be made, a series of steps are processed. The first is to determine where the patient is within the home. The next is to analyze how the robot, from its current position, is to travel through the grid to the patient. This is the process of finding the delivery path.

There are obstacles within a patient's home that limit a robot's movement. These include furniture, walls, and non-traversable flooring. When the grid is defined, these are taken into consideration. Therefore, the only points defined within the grid are the ones that are traversable.

Assuming that the robot and patient are within the same Space ID, the points that need to be traveled between robot and patient makes a single, simple path within the same room. Using our example gird, one can visually see the path(s) that would, not only define a valid path, but also represent the fastest path from one point to another.

This process becomes more complex when there are multiple spaces to travel, the options are more numerous, and the path determination must be calculated by a program instead of a human.

The next section describes the path selection algorithm that is used by MEDS.

A* Algorithm

There are many articles, papers, and books focusing on a fairly large number of search algorithms. Some of these algorithms include A*, B*, Depth-first, Dijkstra, and Dijkstra-Scholten. [7] The Dijkstra algorithm, published in 1959, "is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms." [8] In 1968, an extension of the

Dijkstra algorithm, known as A* was released. It provided better performance by using a “best-first search”. [9]

Essentially, the algorithm will span out and search multiple paths simultaneously. Once a path is deemed to be a dead-end, it will be ignored as a potential path. After determining all successful paths, the number of steps is compared in order to select the fastest path choice. An additional feature of the algorithm employed for MEDS is to take the shortest path with the least number of turns. In order words, if there are two (2) shortest path options with two (2) and five (5) turns respectively, the first path will be chosen.

Below is an updated version of the grid example to demonstrate the how a path is chosen. In order to provide a better example, a rug has been added to the grid, representing a non-traversable section of grid.

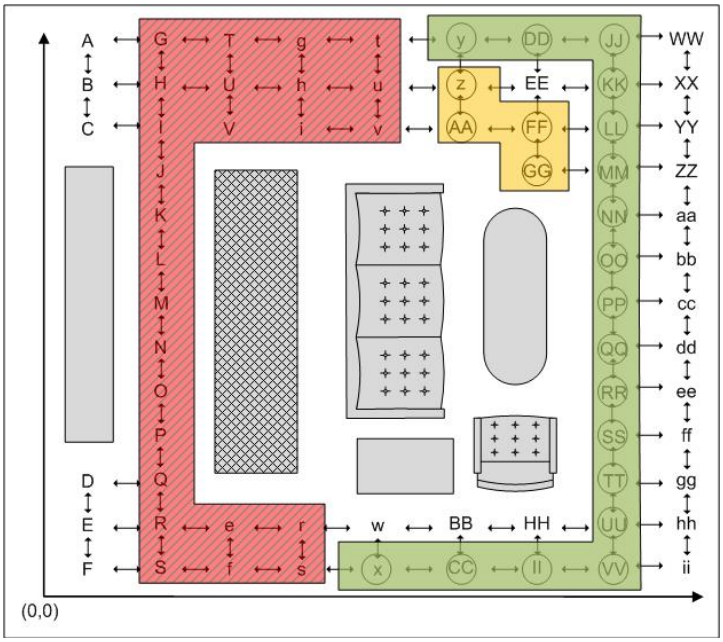


Figure 14 - Path Selection Example

The goal of the path selection is two-fold: find the shortest path, and, if more than one, select the one with less turns. The red section of the grid depicts possible paths from point “x” to point “y”. However, none of the paths would be short enough to be selected. The green section of the grid represents the ultimate path selected with seventeen (17) steps and two (2) turns. The yellow portion of the grid represents a leg of a shortest path that is also seventeen (17) steps. However, traveling through this yellow portion would cause a total of five (5) turns. Therefore, this path is not selected.

Multi-Space Path Finding

When a patient is located within a Space ID that is different that the robot, multiple paths must be combined to complete the delivery of the medicine. As discussed previously, adjacent spaces require the establishment of “hallway” points to allow a robot to travel from one space to another. If the patient is in a space adjacent to the robot, then two (2) sub-paths will be determined and combined to create a full path. Likewise, if there are three (3) adjacent spaces requiring travel, then three (3) sub-paths will be determined and combined into the full path from robot to patient.

Software System

MEDS' software system is set of programs written specifically for the various entities that makeup the system. The core of the system resides in a server application which instructs the various entities to perform based on certain criteria. The three entity types that communicate with the server application are the patients' handheld systems, the robots onboard processors, and the patient location processors. MEDS relies heavily on a database which contains the data necessary for proper system functionality. Additionally, in order to review processing that has occurred, research what is expected to happen, and maintain an overall understanding of the system's execution, a series of reports will be required. The details of these programs are presented in the sections below.

Server Application

Description

The Server Application is the main controller of the MEDS system. It is responsible for maintaining the patients' medication schedules, space and grid definitions, commanding the robots to perform when appropriate, receiving commands from the patients' handhelds, and communicating with the database to track all processing that occurs.

Maintenance Modules

Grid Creation/Maintenance

The Grid Creation/Maintenance program is responsible for defining the patient's home in terms of spaces and coordinates. The program allows

the user to define the Spaces comprised of beacons. Each beacon's unique ID, Space ID, height, and coordinate within the grid are defined in this program. The program also allows for the creation of traversable points within each space, as well as the adjacent points of each.

Robot Creation/Maintenance

The Robot Creation/Maintenance program is responsible for defining the robots that makeup the patient's system. For each robot, a home-base coordinate within the grid is defined. This includes which Space ID the point corresponds to and is where the robot starts from and returns to after delivering medicine.

Prescription Creation/Maintenance

The Prescription Creation/Maintenance program is responsible for defining the prescriptions that a patient takes, including the frequency and dosages. The program allows the user to add refill quantities to the prescriptions when filled. The program also defines which robot "holds" the particular prescription.

Scheduler

The Scheduler is a program that runs in the background and checks the prescription list. Comparing the current time with the prescriptions' frequency, the Scheduler determines if a delivery is due. Once a delivery time occurs, the Scheduler kicks off the Delivery program.

Delivery

The Delivery program is responsible for all aspects of the actual delivery of medicine. This includes the location of the patient within the system, the command to the appropriate robot to wake up, the determination of what path to travel, the travel of the robot, the delivery of medicine, the acceptance of the patient, and then return of the robot to its home point.

Interface Modules

Robot Interface

The Robot Interface is a program which receives a command from the Scheduler and informs the appropriate robot to deliver the medicine to the patient.

Patient Locator Interface

In order for the robot to deliver the medicine to the patient, it must receive the grid location of the patient. The Patient Locator communicates with the patient location processor to determine its coordinates and deliver this information to the robot as its travel goal point.

Handheld Interface

The Handheld Interface is a module within the Server program that sends and receives information to and from the patient's handheld device. The information, whether originating from the patient or the server, is stored in a table and processed accordingly.

Languages

The Scheduler, Handheld Interface, and Maintenance programs are written in C# in the .NET environment. The Robot Interface and Patient Locator Interface are written in C.

Database

The Database is Microsoft SQL Server 2008.

Client Programs

Patient Program

Description

The patient will have a handheld device (smart phone, iPad, tablet, etc.) in order to confirm that their medicine was delivered successfully. When a prescription is scheduled to be delivered and a robot is sent out on delivery, the patient is notified via their handheld of the pending delivery. Once the robot has located the patient's grid coordinates and traveled to the proper location, the patient retrieves the medicine and uses the Patient program on their handheld to confirm the medicine was delivered. This action will ultimately result in the robot receiving a command to return "home". Additionally, the patients will be able to view their schedule, see their current medicine counts, and be notified when prescriptions are to be filled.

Modules

The modules that will make up the Patient program are as follows:

- Actions Pending
- Reports

Language

The Patient program will be a mobile application written in C#.

Patient Locator Program

Description

The main objective of the Patient Locator program is to determine the (x,y) coordinate and Space ID that corresponds to where the patient is currently located within their home. The patient's wheelchair holds a wirelessly connected computer system that processes commands upon request. Additionally, the wheelchair has one (1) listener mote attached to the system for every Space ID that is defined within the MEDS system.

Modules

Record Distances

The first step is to record the distances from the patient's listener mote to each of the corresponding beacon motes. This process is accomplished through the *RecordDistances()* code described above.

Get Current Location

The second step in the process is, using the saved distances, to get the coordinate of the patient's listener mote through the *CurrentLocations()* code listed above. This process performs the

trilateration calculation and returns the patient's (x,y) coordinates. This information is reported to the Delivery program when requested.

Language

The modules are written in C++.

Robot Program

Description

The main objective of the Robot program is to determine the (x,y) coordinate and Space ID that corresponds to where the robot is currently located within the patient's home. Using this information, the Robot program then commands the robot top move as needed. Each robot has a wirelessly connected computer system that processes commands upon request. Additionally, the robots have one (1) listener mote attached to the system for every Space ID that is defined within the MEDS system.

Modules

The modules below are performed once a point is provided to the Robot program from the Delivery program. The Robot program repeats the modules below until such time that the goal point is reached. At this point, the Robot program returns success to the Delivery program, which then determines what steps to take next.

Record Distances

The first step is to record the distances from the robot's listener mote to each of the corresponding beacon motes. This process is accomplished through the *RecordDistances()* code described above.

Get Current Location

The second step in the process is, using the saved distances, to get the coordinate of the robot's listener mote through the *CurrentLocations()* code listed above. This process performs the trilateration calculation and returns the robot's (x,y) coordinates. This information is reported to the Delivery program when requested and used by the Movement Algorithm.

Movement Algorithm

The third step that is processed is the movement algorithm. Using the algorithm pseudo code previously discussed, the movement algorithm determines the degree, direction, time, and speed of the robot's next movement. The robot then moves as directed.

Language

The modules are written in C++.

Reports

Prescription Report

The Prescription Report provides all applicable information pertaining to a patient's prescriptions. The information that is reported includes current prescription levels, history of every prescription delivery, detailed information about a specific prescription, and a history of all refills.

Path Report

The Path Report allows the system administrator to review the paths selected by the path algorithm. The report details the robot's starting point, the patient's location, and the individual points within the selected path for each delivery. Additionally, the user may drilldown into the robot movements that were taken to reach the intermediate path points. This information includes how many movements it took to move from a path point to the next path point. In this way, multiple paths that include the same from-to point combination can be compared against each other. Additionally, different from-to point combinations can be compared against each other to determine if some points are not receiving the most accurate transmissions. This could be due to deflections, low-batteries, the need to increase the number of beacon motes, or to adjust the beacons' locations.

Patient Locations Report

The Patient Locations Report provides the user with a list of the coordinates at which the patient was recorded. This information could be beneficial in determining whether the patient is active enough or not. Additionally, if there are

points where no location reading is recorded, then it can assist in determining whether there are locations the patient travels that are not reachable by the ILS.

Language

All reports are written in C#.

Database

Description

MEDS is extremely reliant on a database structure that maintains all aspects of the system. Below is an Entity Relationship Diagram depicting the database that is used by MEDS. Each table is displayed, listing their individual fields and data types. All Primary and Foreign Keys are included in the diagram to signify the relationships between the various tables. The following section provides additional details and explanations of each of the tables and their fields.

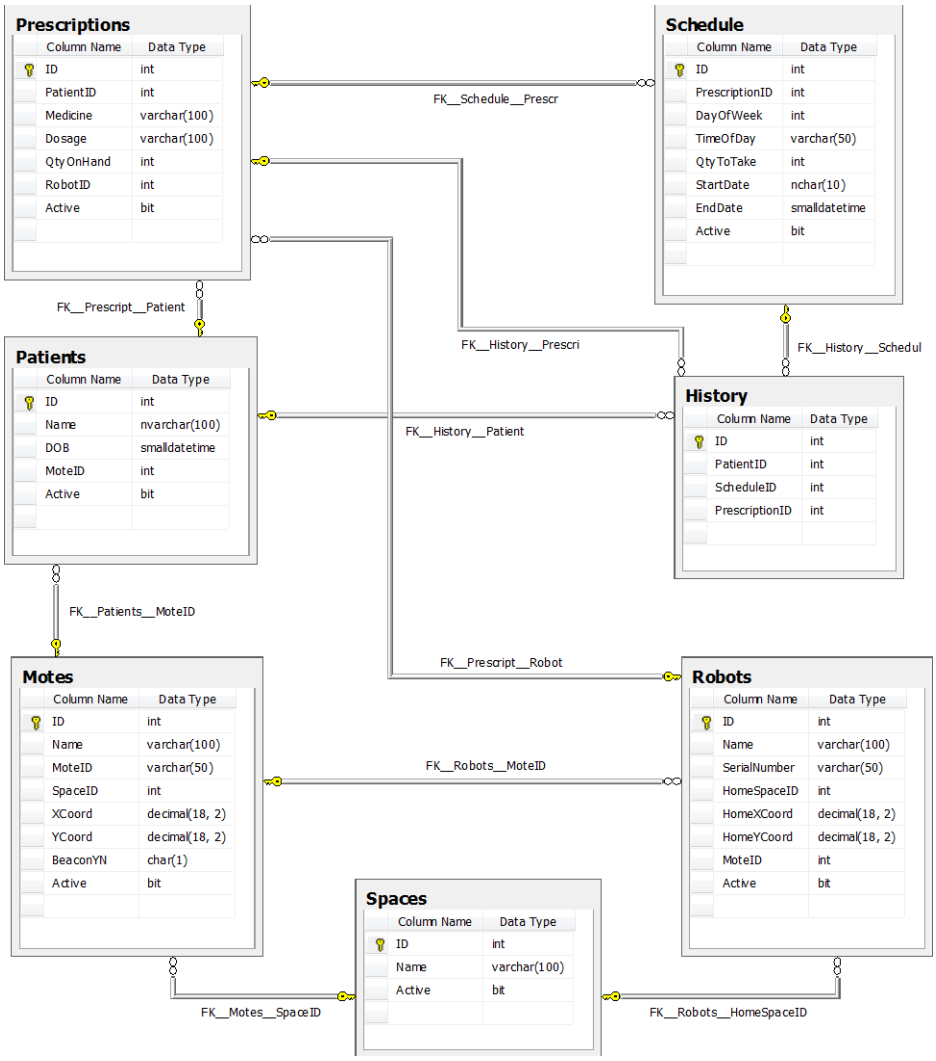


Figure 15 - Database ERD

Tables

Patients

The Patients Table holds the list of patients that are to receive medication through MEDS. The fields are listed and described below:

1. ID –The integer ID is the table's Primary Key (PK), a unique identifier defined by the database.

2. Name – The varchar(100) Name is the patient's name and used for descriptive reasons.
3. DOB – The smalldate DOB is the patients date of birth, used to further denote the patient.
4. MoteID – The integer MoteID is a Foreign Key (FK) to the Motes' ID PK. This integer defines which mote is placed on the patient's location device.
5. Active – The bit Active determines whether the patient is an active participant of MEDS at the current time.

Prescriptions

The Prescriptions Table holds the list of prescriptions that are recorded in MEDS for each patient. The fields are listed and described below:

a. ID –The integer ID is the table's Primary Key (PK), a unique identifier defined by the database.
b. PatientID – The integer PatientID is a Foreign Key (FK) to the Patients' ID PK. This integer defines which patient the prescription corresponds to.
c. Medicine – The varchar(100) Medicine is the name of the medicine that is being dispensed per this prescription.
d. Dosage – The varchar(100) Dosage is the amount to dispense of the medicine on this particular prescription.
e. QtyOnHand – The integer QtyOnHand is a number that represents how many of the particular medicine that is currently being held by

the robot. When the prescription is dispensed, the number equivalent to the amount dispensed is deducted from the QtyOnHand. When prescriptions are refilled and added to the robot, this number needs to be increased to represent the refill's quantity.

f. RobotID – The integer RobotID is a Foreign Key (FK) to the Robots' ID PK. This integer defines which robot is holding this particular prescription.

g. Active – The bit Active determines whether the prescription is an active part of MEDS at the current time.

Schedule

The Schedule Table holds the schedule of prescriptions that a patient is currently taking. The fields are listed and described below:

a. ID –The integer ID is the table's Primary Key (PK), a unique identifier defined by the database.

b. PrescriptionID – The integer PrescriptionID is a Foreign Key (FK) to the Prescriptions' ID PK. This integer defines which prescription this particular schedule represents.

c. DayOfWeek – The integer DayOfWeek is a number that represents which day of week this prescription is to be dispensed. 0=Sunday, 1=Monday, ..., 6=Saturday.

d. TimeOfDay – The varchar(5) TimeOfDay is a character field that represents what hour this prescription is to be dispensed. 00:00=12:00AM, 05:15=5:15AM, and 19:30=7:30PM.

e. QtyToTake – The integer QtyToTake is a number representing the number of pills to take at this particular scheduled date/time.

f. StartDate – The StartDate is an optional SmallDateTime field that represents a date that the prescription is to start. If there is no StartDate defined, then the prescription has an open-ended starting date.

g. EndDate – The EndDate is an optional SmallDateTime field that represents a date that the prescription is to end. If there is no EndDate defined, then the prescription has an open-ended ending date.

h. Active – The bit Active determines whether the schedule is an active part of MEDS at the current time.

Spaces

The Spaces Table defines all the spaces that makeup the indoor location grid. The fields are listed and described below:

a. ID –The integer ID is the table's Primary Key (PK), a unique identifier defined by the database.

b. Name – The varchar(100) Name is the space's name and used for descriptive reasons.

c. Active – The bit Active determines whether the space is an active part of MEDS at the current time.

Motes

a. ID –The integer ID is the table's Primary Key (PK), a unique identifier defined by the database.
b. Name – The varchar(100) Name is the mote's name and used for descriptive reasons.
c. MoteID – The varchar(50) MoteID is the mote's unique ID that is programmed into its firmware. This ID is used when determine locations of listeners based on beacons in range.
d. SpaceID – The integer SpaceID is a Foreign Key (FK) to the Spaces' ID PK. This integer defines in which space this particular mote is located.
e. XCoord – The decimal XCoord is a number which defines the mote's x-coordinate within the system's grid.
f. YCoord – The decimal YCoord is a number which defines the mote's y-coordinate within the system's grid.
g. BeaconYN – The bit BeaconYN is a flag stating whether the mote is a Beacon (giving off signals) or a listener (receiving signals).
h. Active – The bit Active determines whether the mote is an active part of MEDS at the current time.

Robots

a. ID –The integer ID is the table's Primary Key (PK), a unique identifier defined by the database.
b. Name – The varchar(100) Name is the robot's name and used for descriptive reasons.
c. SerialNumber – The varchar(50) SerialNumber is the robot's unique ID that is assigned to it by the manufacturer.
d. HomeSpaceID – The integer HomeSpaceID is a Foreign Key (FK) to the Spaces' ID PK. This integer defines in which space this robot's home base is located.
e. HomeXCoord – The decimal HomeXCoord is a number which defines the robot's home base location's x-coordinate within the system's grid.
f. HomeYCoord – The decimal HomeYCoord is a number which defines the robot's home base location's y-coordinate within the system's grid.
g. MoteID – The integer MoteID is a Foreign Key (FK) to the Motes' ID PK. This integer defines which listener mote is on this robot.
h. Active – The bit Active determines whether the robot is an active part of MEDS at the current time.

History

a. ID –The integer ID is the table's Primary Key (PK), a unique identifier defined by the database.

b. PatientID – The integer PatientID is a Foreign Key (FK) to the Patient' ID PK. This integer defines which patient this history record references.

c. ScheduleID – The integer ScheduleID is a Foreign Key (FK) to the Schedules' ID PK. This integer defines which schedule this history record references.

d. PrescriptionID – The integer PrescriptionID is a Foreign Key (FK) to the Prescriptions' ID PK. This integer defines which prescription this history record references.

Database Type

The Database is Microsoft SQL Server 2008.

Example Program Flow

This section describes the flow of steps that are taken to determine that a prescription is due and to deliver the appropriate medicine to the patient. This example assumes that the system has been setup with motes, robots, spaces, grids, and prescriptions.

1. The Scheduler determines that a prescription is needed to be delivered.
2. The Scheduler notifies the Delivery program of what prescription to deliver.
3. The Delivery program activates the Patient Locator program.
4. The Patient Locator program returns the patient's Space ID and coordinates point within the home.
5. The Delivery program determines which robot holds the prescription.
6. The Delivery program determines the appropriate path to take in order to deliver the prescription.
7. The Delivery program sends the robot's next point to the Robot program.
8. The Robot program maneuvers the robot to the next point and informs the Delivery program when achieved.
9. These steps are repeated through the path list until the robot reaches the goal point of the patient's location.

10. The Delivery program sends a notification to the Patient Locator program noting that delivery has occurred.
11. The Delivery program waits for a notification from the Patient Locator program that the patient received the delivery.
12. The Delivery program reverses the path and sends the robot home using the same process described above.
13. Each step of the way, the programs write their actions taken to the database

Below is a Message Sequence Diagram visually displaying the program flow and communication that occurs when a prescription is determined ready for delivery.

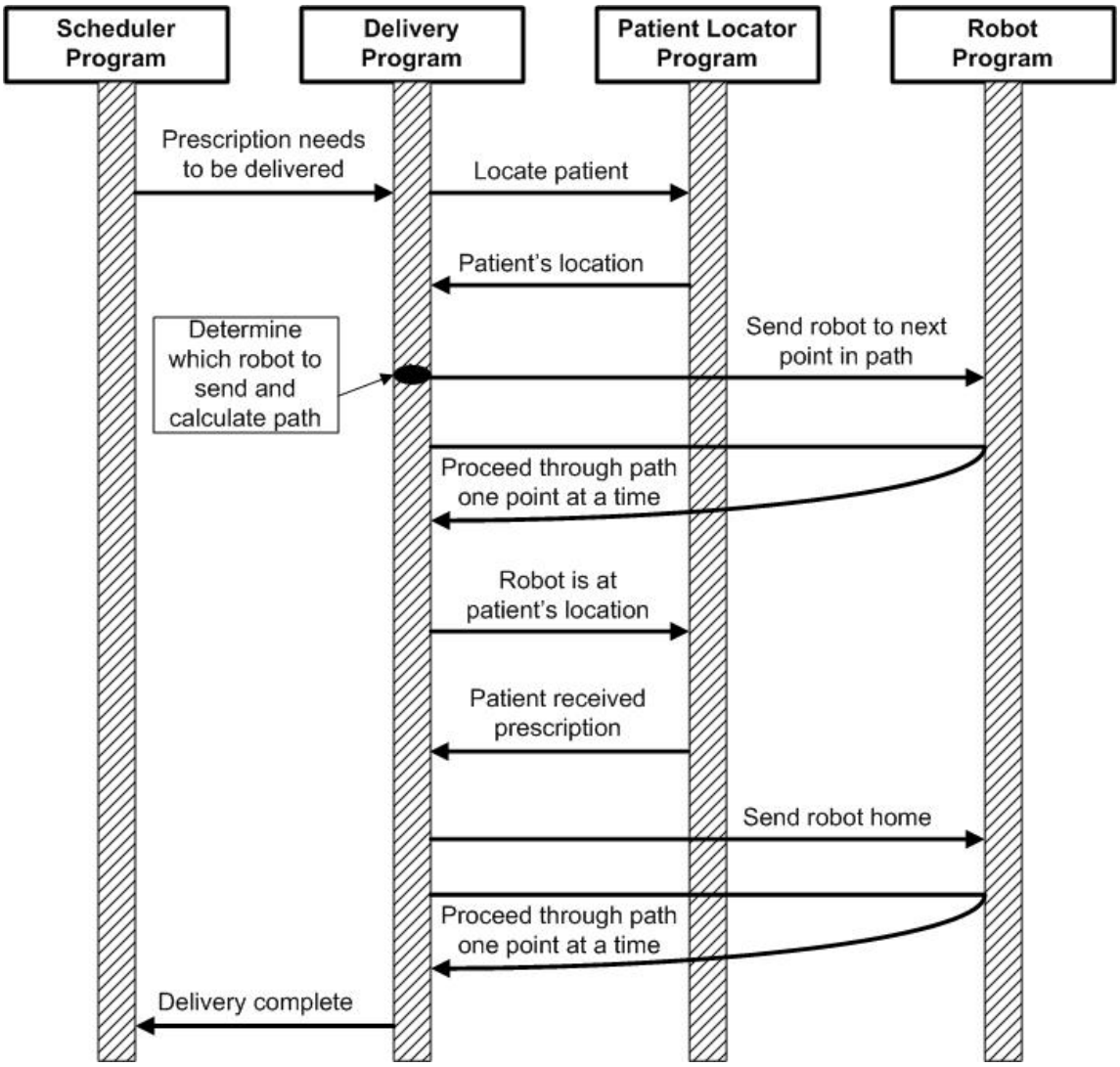


Figure 16 - Message Sequence Diagram

Future Enhancements

The section describes features that are not presented in the initial MEDS plan, but would be viable and applicable enhancements to add to future implementations.

On-Command Patient Requests

The “On-Command Patient Requests” feature would enable a patient, using their handheld device, to activate a robot through a direct command. This would enable the patient to receive certain medical attention as needed. Examples could include items such as Ibuprofen, insulin, and cough syrup.

Connections to Outside Systems

Enabling MEDS to connect to and communicate with outside systems would be an extremely valuable enhancement to include in a future release of MEDS. The communications could be manually initiated from a patient’s handheld device or programmatically generated from the Server program.

Alerts to Emergency Contacts, Departments

MEDS could notify a patient’s emergency contacts as well as emergency personal (such as Fire, Police, and EMS) in case of an issue. The notifications could be generated manually by the patient through their handheld device. Additionally, the system could generate alerts based on their findings within the system. For instance, the “Tracking Patient Movements” enhancement defined below could detect that a patient has been immobile for an extended length of time and alert the proper person or department of a potential issue.

Updates to Pharmacies, Doctors

MEDS continually monitors the prescription on-hand levels as they are dispensed and restocked. Using the on-hand values combined with the schedule of future dosages, MEDS can determine when a prescription will need to be refilled. The patient's pharmacy could be notified within a certain time cushion of a desired refill. This would ensure that the patient's medicines are maintained at a safe level. Additionally, if the prescription refills have been exhausted, a notification could be sent to the prescribing physician requesting additional refills.

Tracking Patient Movements

An enhancement to include in a future release of MEDS would be to track a patient's movements throughout the ILS. One benefit of this feature would be to ensure that the locations traveled by the patient are covered by the locationing system. Additionally, in connection with the "Connections to Outside Systems" feature, lack of movement by the patient could inform an outside entity of a potential problem.

Upgraded Dispensary Robot

The "Upgraded Dispensary Robot" would be a fully designed and developed robot with separate sections for the various prescriptions required. The robot would maneuver to the patient, open the appropriate compartment, and dispensing the exact level as prescribed.

Conclusion

This paper presents an indoor delivery system entitled Mobile Electronic Dispensary System (MEDS). MEDS is a complex system which combines a database, programs, algorithms, and hardware to achieve a very unique home medical service. Using robots and an indoor location system, MEDS locates a patient within their home and delivers their medicine or other medical supplies based on their prescription schedule.

A great importance of MEDS is its ability to overcome the lack of GPS accuracy within an indoor location. This enables the patient to, in spite of their physical or mental disabilities, receive timely and accurate medical deliveries. Creating and implementing MEDS for a patient within their home would be an incredible benefit to their independence and a relief of worry to their loved ones.

The MEDS project utilized important Software Engineering concepts and procedures in order to design, develop, and test a quality system. Requirements Analysis was performed to determine the needs of a patient who lived at home and was on an explicit prescription schedule. A System Architecture development process was completed in order to determine the most effective hardware and communication system to create based on the requirements gathered. The next step in the process was to design and develop a highly modularized and efficient software and database system. A Test/Validation process was created in order to ensure all intended features of the system performed as expected. By employing the Software Engineering methods and methodologies, the MEDS project resulted in a highly effective and unique system.

References

1. "The Cricket Indoor Location System", http://cricket.csail.mit.edu/.
2. "iRobot Create", http://www.irobot.com/en/us/robots/Educators/Create.aspx.
3. "GPS.gov", http://www.gps.gov/.
4. "Cricket v2 User Manual", http://cricket.csail.mit.edu/, Jan 2005.
5. "Trilateration", Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Trilateration.
6. "Shortest Path Problem", Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Shortest_path_problem.
7. Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L. 1990. Introduction to Algorithms. MIT Press and McGraw-Hill.
8. "A* Search Algorithm", Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/A*_search_algorithm.