University of Arkansas – CSCE Department
Capstone II – Final Report – Spring 2011

Autonomous Floor Mapping

Akihiro Eguchi – Daniel Hooten – Hung Nguyen

http://sdc.csce.uark.edu/projects/mapping/

1. Background

An updated and accurate floor map is important when we want the robot to go to a known location. Even though there are methods that will navigate the robot from one place to another around obstacles without a map, they are generally inefficient because the robot may choose the longest path because there is no information about the map. If we need to navigate the robot often and for a long time, for example the Roomba vacuum cleaner, a robot has to keep track of the environment constantly because errors can accumulate and the robot become lost. With a known map, the robot has a better chance of recovering its current position in case it gets lost because the wheels get stuck or it is trying to go over some bumps on the floor. However, the cost of mapping a floor is not cheap, especially when relatively good maps are required, and mapping a floor is very tedious for a human to do. Therefore, we tried to investigate a new and low cost robot that can do the mapping autonomously and efficiently. This is especially useful to acquire the map of a house.

2. Problem

A floor map is a record of an environment that details the boundaries of that environment and the boundaries of the objects in that environment. This description is in turn used by a robot like the Roomba to safely transverse the environment. Having a floor map before would enable the Roomba to better optimize the route it goes to make the route shorter so it completes its task faster. However, there is currently no low cost robot that can autonomously navigate through an unknown environment and map the whole area for us. The cost for one of the current laser scanners like SICK LMS-200 is $5,771.43. [4] Additionally, in order to see how the robot works based on the program we write, we have to build prototypes several times. This cost for building prototypes is also a problem.

3. Objectives

The objectives of our project are:

1. To build a low cost robot floor mapper

2. To develop a new algorithm used for autonomous floor mapping
3. To learn about robotics

4. Universal Interface

In order to test the robot in the real world, we must first test out our algorithm in a fully controlled environment such as a robot simulator. To make the transition for hardware and software simple, we require both parts to share a universal interface. Below is a description of the Interface class from which software and hardware inherits:

```
public abstract class Interface implements Runnable {

	public abstract Coordinate goStraight(float distance);
	public abstract float rotateAbsolute(float angle);
	public abstract float rotateRelative(float angle);
	public abstract boolean stop();

	public abstract float getDirection();
	public abstract void sweep();
	public abstract boolean[] getHits();
	public abstract float[][] getSensorReadings();
	public abstract float readAngle(float angle);
}
```

Since we use two software emulators and one real hardware platform, this interface makes the task of switching from one platform to another as easy as changing one line of code. As an example, we can change from

```
	Interface robotInterface = new SoftwareInterface();
```

to

```
	Interface robotInterface = new HardwareInterface();
```

for testing on the real robot platform after testing it with robot simulator.

5. Software

5.1 Autonomous floor mapping algorithm

There are many possible ways to implement autonomous floor mapping, and we developed the program based on the idea of occupancy grid mapping [2] [3]. In this approach, we divide the environment into small grids in order to keep track of the information of how the robot moves and where the robot encountered obstacles. A previous paper used a backtracking algorithm to map the floor [4], but this method was not very efficient. Therefore, we came up with more heuristic approach to autonomously map the floor. The pseudo code is the following.

```
WHILE (unexplored grid exists)
        IF (no block in front) and (not explored in front)
                IF (no block on the left) and (not explored on the left)
                        turn left
                        go straight
                ELSE
                        go straight
```

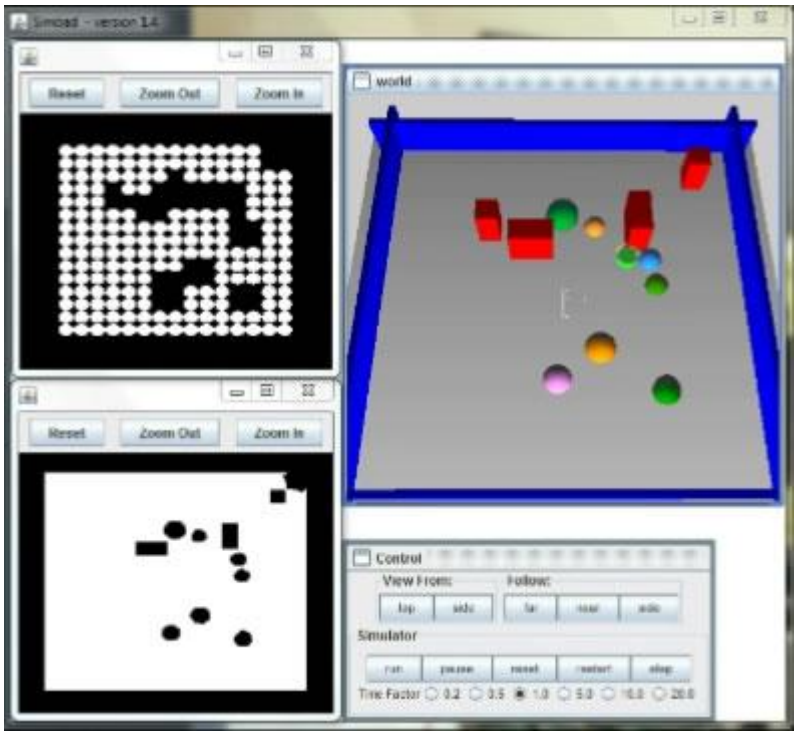
```
ELSE
        IF (no block on the left) and (not explored on the left)
                turn left
                go straight
        ELSE IF (no block on the right) and (not explored on the right)
                turn right
                go straight
        ELSE
                BFS to find unexplored grid
```

Basically, whenever the robot encounters an obstacle, it tries to trace the shape and then tries to map spirally. Then, if it meets a dead-end, it uses breath first search to look for the closest unexplored grid and move to there. If there are no more unexplored areas, then the program terminates. In this way, we succeeded in significantly reducing the cost of the mapping when compared to the previous backtracking method.

5.2 3D simulation

Even if people have an idea for developing new technology or inventing new product, the cost of implementation is always a problem. Also, a software developer may need a help debugging the hardware part, and this makes the process of the development slower. However, what if we could use a virtual world to simulate those new ideas as a virtual object controlled by code we write for real world. Since developing and testing in a virtual world may eventually require less cost than in the real world, this approach to prototyping and testing could provide advantages over real world prototyping and testing.

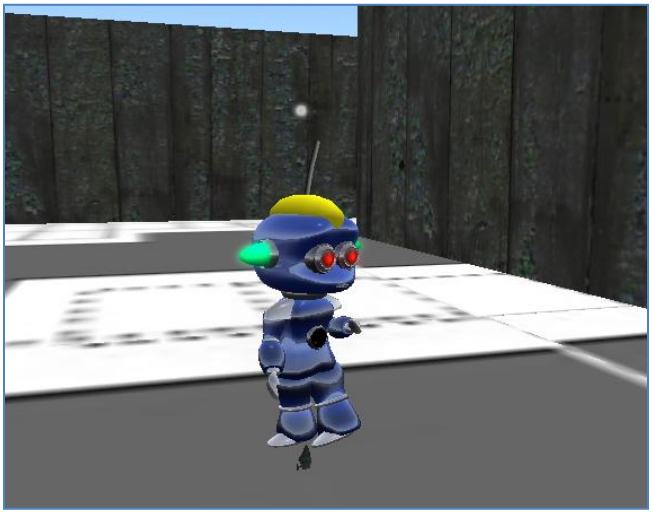
In order to test out the algorithm in a more realistic environment, we used a 3D modeling tool called Simbad robot simulator, which provides a quick way to simulate a robot control with sensors and test our programs in a 3D environment.



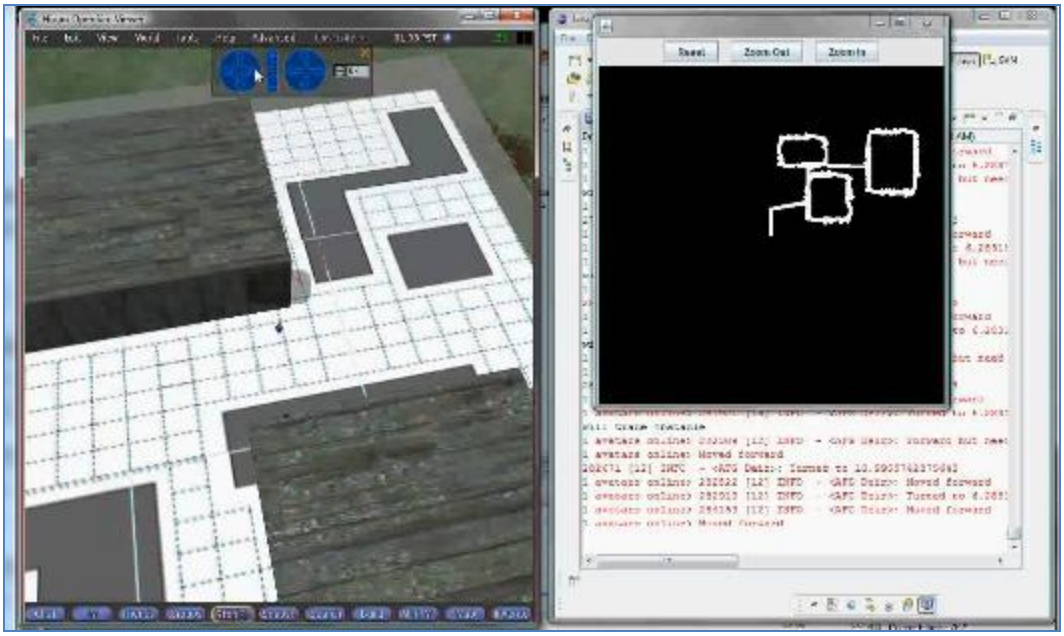
One problem with the method of occupancy grid mapping is that in order to map the floor accurately, it is required to divide the grid into small parts. However, the more numbers of grids used, the more mapping time required. In order to solve this problem, we decided to use the method only for navigating a robot, and use some other sensors like a the sonar sensors in Simbad and IR sensor in real world to more accurately map the environment. As a result, we successfully map the floor really accurately.

5.3 Virtual world simulation

In order to test the program in an even more realistic environment, we decided to use a virtual world environment, Second Life. It can be accessed from anywhere in the world without any cost, and the environment simulates physical low like in the real world. One problem of the Second life is that it does not provide a direct way to communicate with objects in the environment from outside of the client software. Therefore, it is required to set up the environment in the second life with Linden Scripting Language, and a C# library provided by Open Metaverse Foundation to connect the virtual world and our Java code for the mapping.

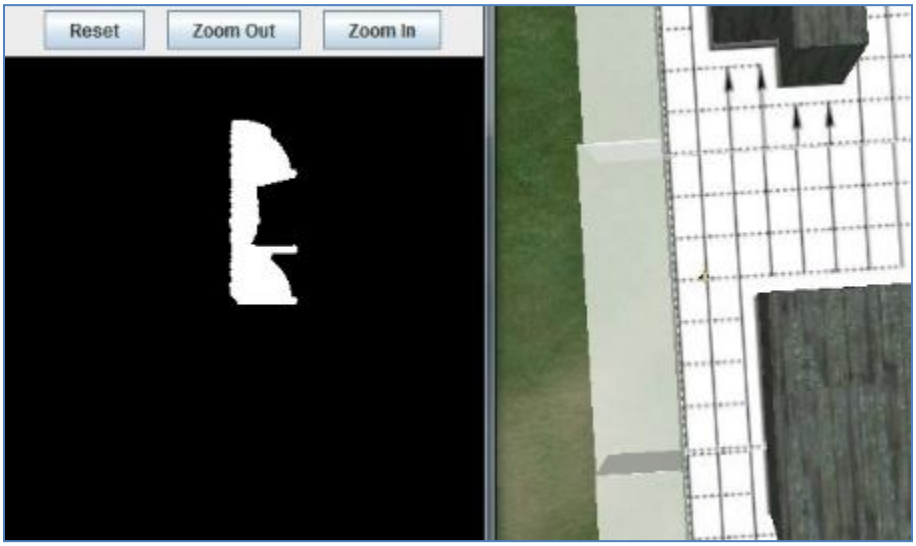


This avatar bot, controlled by our Java code, roams around the maze building the map. The avatar wears a device to send his location and information of the collision with a wall back to the Java code outside of second life, and the avatar is fully controlled by the Java code. Therefore, whenever the bot detects any change of its position, it sends the information about the distance traveled back to the program.

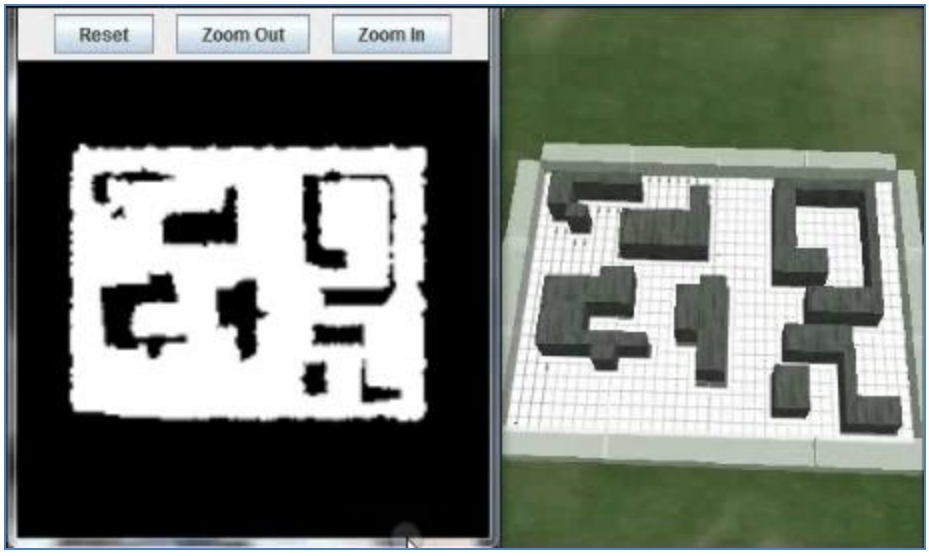


In order to test the program that controls the objects in second life, we first built a simple code to control the robot. Once it detects an object, it traces the shape of the object by moving around the object. Based on the input the program gets, it visually builds a map.

Then, we developed a sensor like IR beam sensor in a second life, which will be needed for our autonomous floor mapping program. It throws tiny particles to each180 degree to determine if there are any objects around the avatar. If particles hit any objects, then it calculates the distance between the avatar and the object.



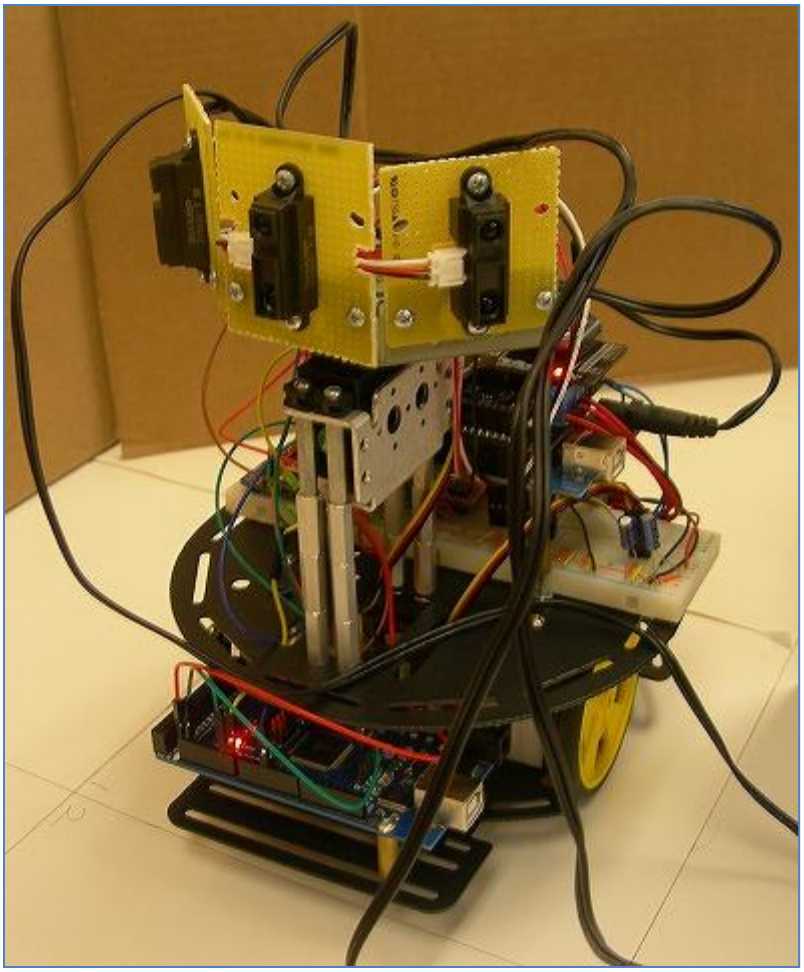
Then, using the data obtained by the sensor, we tested our algorithm to map the floor.



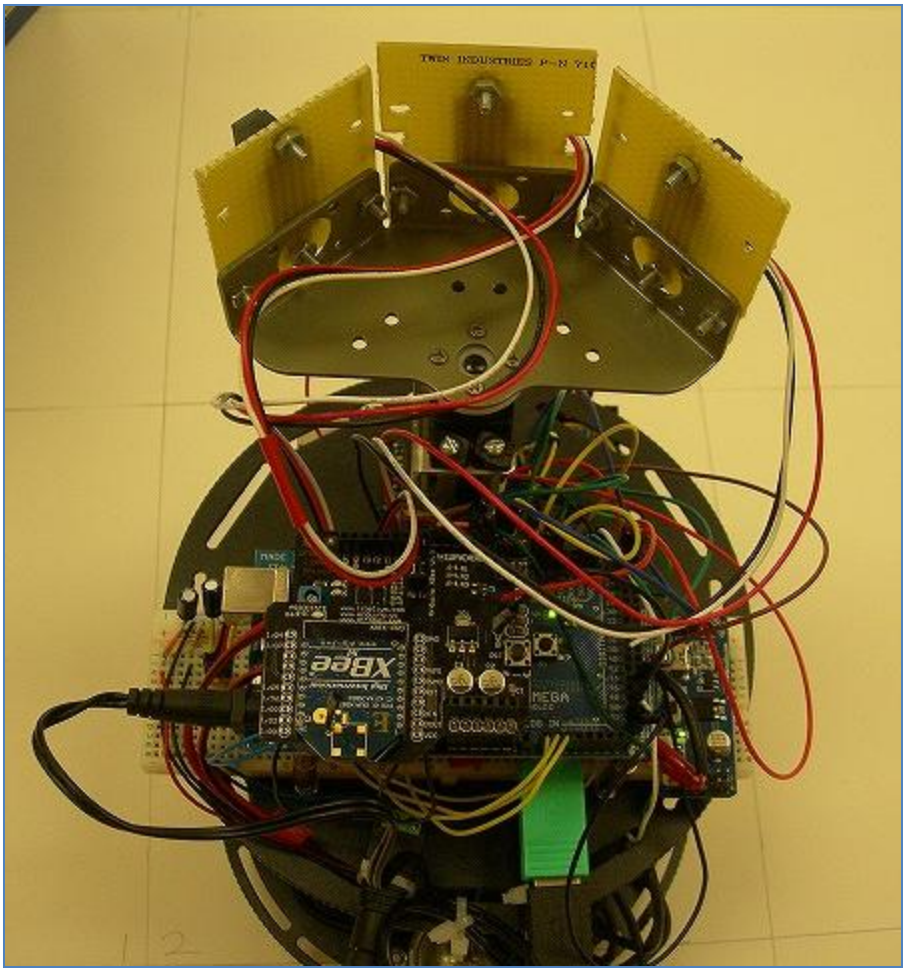
When compared to the simbad 3D robot simulator, the shape of the map is not very accurate. This is because we physically measure the distance between avatar and objects using the sensor we developed. It could be influenced by many possible noises like wind, gravity, network lag, and etc. However, it is also true that in the real world simulation, we will have to deal with many environmental factors that bend the input data we would obtain. Future work will be needed to work on these kinds of problems.

6. Hardware

6.1 General Design



- The robot uses three Sharp Infrared rangefinders (IR sensor) to speed up the process of sweeping an area and to achieve larger range. The sensors are mounted on top of a servo which rotates the sensors to a specific angle. The leftmost and right most sensors are mounted at a 45^{o} relative to the middle one, combined with the 180^{o} range of the servo gives us a maximum sweeping range of 270^{o}.
- The mouse is mounted at the bottom of the robot to keep track of the robot movement. We assume that the floor the robot operates on is smooth and works with a regular optical mouse. In addition to the mouse, we also have one wheel encoders to help turning to an angle faster.
- The robot has a digital compass with 0.1^{o} accuracy to keep track of the heading angle.
- The robot and the computer exchange data using two XBee modules, which provide wireless serial communication.
- The robot has two Arduino boards: one is used exclusive for keeping track of and updating the target pose, which is a vector (x, y, θ) indicating the x and y coordinates and heading angle, as the robot moves; the second board is for all the remaining tasks: sweeping the area with the IR sensors, turning the motors, and communicating with the computer via the XBee module.



6.2 The hardware interface

Since both software and hardware has the same interface, and the algorithm only relies on this universal interface, the only change that needs to be made to switch from software simulation to real world hardware can be as trivial as changing only one line of the code. The abstraction of the underlying architecture that we have studied is first applied successfully to the project to speed up the development process and makes the transition from software simulation to real world run easy. However, we encountered many problems when we tried to implement the interface in the real world.

6.3 Problems encountered in hardware implementation

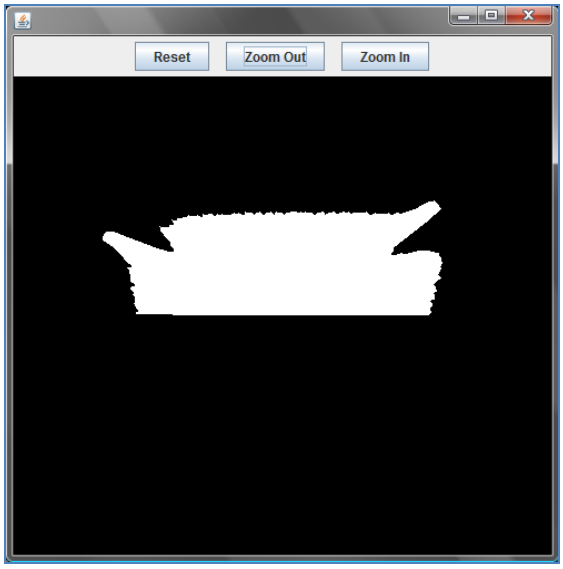
6.3.1. Erroneous wiring of IR sensor cables

Two of the sensor cables that we ordered from *trossenrobotics.com* were incorrectly wired. The colors of the VCC and Ground pins were swapped. Therefore, the VCC wire is black, and the Ground pin is red, completely opposite from normal convention. This resulted in the immediate destruction of two of our IR sensors. We did not know the cause of the problem and tested the third IR sensor with the same wires and damaged the third one also. This error is the first one

that we encountered and was the fault of the vendor. Because of this, we had to wait until after Spring break to for new IR sensors to arrive to work on.

6.3.2. IR sensor's readings are wrong with sharp vertical edges

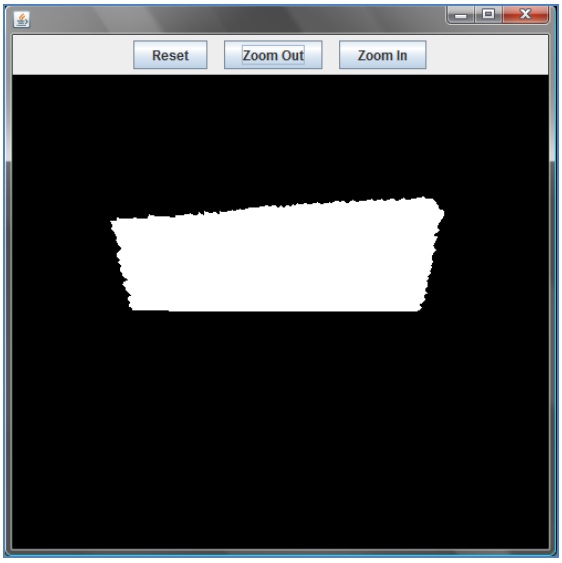
When we did a full 180 degree scan with the IR sensors, we could see very strange spikes as in the picture below at the edge of the object.



We experimented multiple times with different objects and shapes, and the problem always occurred at the vertical edges of the objects. To understand the cause of the problem, we had to research how an IR sensor works. It works by triangulation: an IR beam is emitted from one window and received at the other. The measured angle of reflected beam indicates how far the object is. We conjectured that the problem is because of the beam width of the IR sensor being

large (even though still very small compared to the cone shape of sonar sensors) in order to detect objects with unknown distances. If the edge of the object is within the beam width, then the IR sensor is trying to measure two different surfaces (or points) at the same. Therefore, the readings near the corners or edges of the objects are wrong because there is a sudden sharp change in the orientation of the surfaces.

If the beam's width is perpendicular to the edge, we have wrong readings. However, we hoped that the beam's height is very small compared to the beam's width, and it will detect the edges correctly. Therefore, instead of using the IR sensors as recommended by the datasheets and all online tutorials, we chose to rotate them 90 degrees so that the line going through the two windows was perpendicular to the ground, and the result is very satisfying.



6.3.3. Mouse's update speed could not keep up with the robot's movement

Initially, we used a mouse with 400 dots per inch (DPI) resolution. No matter what we tried, even speeding up the update rate or switching from remote mode ,which lets the mouse keep track of the movement counters and only reads them when needed, to stream mode which can interrupt the microcontroller when the mouse moves, the mouse's readings were still 2-3 cm short of the actual movements and became worse as it moved further. Realizing that the mouse's DPI was not high enough for our project, we ordered a new mouse with an advertised 1000 DPI and still using PS/2 interface, the Logitech M110. We then ran into the problem of the movement counters overflowing even though the mouse only moved 3-4 cm. Therefore, we opted to use stream mode with interrupts for the new mouse. The initial test with only the mouse was promising, but when we tried to adjust the headings and distance of the robot inside of the mouse's interrupt routines, it took quite some time for the function to complete. Since we did not want to nest the interrupts, all timer interrupts were paused while the Arduino board processed the mouse readings and tried to compute the new direction and distance. This interfered severely with our motors and made them act erratically since they work based on Pulse Width Modulation (PWM). Therefore, we had to use a second Arduino board just to process the mouse readings at the rate of 200 Hz and compute the new heading and distance to get the robot to a preset target.

This board then uses a pin to interrupt the first Arduino board to stop the motors once it determines that the target location has been reached.

6.3.4. Motors do not have necessary torque

The original motors were high speed, but low torque. They had problems getting over rough spots in the floor. We decided to switch to motors with lower speed and higher torque. We decided that operating at lower speed was acceptable. The original motors had a gear ratio of 120:1 while the new motors had a gear ratio of 224:1 meaning that the new motors had twice as much torque as the original motors.

6.3.5. Turning to an arbitrary angle is difficult

The problem of turning to an arbitrary angle is hard because the rubber wheels skid when the robot turns. Besides this the robot does not have a battery and has to be powered by wired adapters, the wires can affect the direction of the robot. Therefore, instead of a continuous turn with the wheels spin at normal speed and counting down the ticks the wheel encoders generates, we use the following to get the robot turn quickly to the approximate range of the target heading and then used small wheel rotations to turn the robot to a very accurate angle. The small wheel rotations are short bursts of 15 ms with the motors set at nearly the maximum speed. This has worked really well for us except for the inherent inaccuracy of the compass. We tested turning the robot on the table and on the ground. The robot works fine on the table, but behaves strangely when on the floor until the compass is recalibrated. Even worse, the robot does not work consistently for every spot on the floor. When near lots of metal frames or power ports, the compass has to be recalibrated to work correctly. Therefore, we choose to let the robot run on the table instead of the floor.

6.3.6. Going straight

Due to the two motors not perfectly synchronizing with each other, it is difficult to get the robot to go straight for some distance - in our case the distance is 20 cm. The current API assumes that the goStraight function works flawlessly like in the Simbad simulator, so all error corrections have to be done on the Arduino boards. Similar to turning, we use small wheel turns to get the robot move a few centimeters forward. Then the main Arduino board asks the second one which keeps track of the robot's movement using the mouse to send back the vector that will correct the robot's path. Then the robot turns to the corrected angle and moves one more step forward. This process continues until we can reach the desired location with error rate of +/- 3 mm. This has worked really great for us even though the mouse we are using is just a regular mouse and not some high-end gaming mouse.

7. Potential Impact

Roomba robots will be smarter and will efficiently vacuum the whole house or only a specific area in the house. Future home robots can use the map to communicate with each other to accomplish household tasks such as feeding the dog, vacuuming, sweeping, and mopping the floor, finding lost items, moving heavy objects such as beds or dresser, or just to get us a soda from the refrigerator.

Also, it can be useful in a real world situation in a store. The robot roams around the store and detects any changes of the layout of the product display to build a map. By adding an RFID

sensor to the robot, it can keep track with information like what kind and how many product are at any certain location in a store. This system reduces a cost of initial investment for installing RFID to a store.

8. Future Work

The algorithm can be improved by using A* search instead of breadth first search to find the closest unexplored area. Currently we compute the cost of going from one square to another by counting the number of squares needed to accomplish that. However, with A* we can further consider adding turning as part of the cost, so the problem is to find the path with least amount of turn and distance to the desired location. We can even expand this further to allow the robot travel in arbitrary path instead of a grid-based one.

The immediate future work for the hardware part is to use gyroscope and accelerometers to help with the turning because they are not affected by magnetic fields like the compass. Another improvement is to improve the speed of the robot by using better motors and wheel synchronization methods such as a mechanical instead of electrical wheel syncing.

Furthermore, we can apply this work to everyday life situations by combining it with the concept of smart objects [5]. While the robot roam around to build a map, an RFID reader, which could be embedded on the robot, can detect the information about all nearby smart objects to overlay onto the map the robot autonomously built. Then, a smart phone can show the map with the location of all the smart objects you have in your house. This idea puts all smart objects in a house under the control of owners everywhere with a smart phone. From your bed room on the second floor, you could turn on the air conditioner on the first floor. If you forget to turn off the AC, you can do so from your place of work later. All of this will be possible in the near future.

References

[1] T. Sebastian. *Probabilistic robotics*. Cambridge, Mass: MIT Press, 2005.

[2] M. F. McNally, "Walking the grid: robotics in CS 2", in *Proceedings of the 8th Austalian conference on Computing education - Volume 52*, Darlinghurst, Australia, Australia, 2006, p. 151–155.

[3] S. Albers, K. Kursawe, & S. Schuierer, "Exploring unknown environments with obstacles", in Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 1999, p. 842 - 843.

[4] K. Sevcik. "Interfacing with the Sick LMS-200" internet: http://www.pages.drexel.edu/~kws23/tutorials/sick/sick.html, 2006 [May 11, 2011].

[5] A. Eguchi and C. Thompson. "Towards a Semantic World: Smart Objects in a Virtual World." *International Journal of Computer Information Systems and Industrial Management*, vol. 3(4), 2011.