In Search of a Cost Effective Way to Develop Autonomous Floor Mapping Robots

Hung Nguyen, Akihiro Eguchi, Daniel Hooten
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR, USA
{hpnguyen, aeguchi, dhooten}@uark.edu

Abstract— **Simulation can be used to reduce the time and cost to develop a new technology. This paper describes the development of an autonomous floor mapping robot. In order to reduce the cost of building prototypes to test the program, we used the Simbad 3D simulator. To test in a more realistic environment, we established a way to control objects in a virtual world Second Life. Then, for the hardware part, we built a low cost robot with cheap but accurate Sharp IR sensors with a regular optical mouse.**

Keywords-Floor mapping, Simulation, Mobile robots, Navigation, Robot sensing systems

I. BACKGROUND

An updated and accurate floor map is important when we want a 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 an inefficient path because there is no information about the map. If we need to navigate the robot often within an area, for example, the Roomba vacuum cleaner, a robot has to keep track of the environment constantly because errors can accumulate, and the robot becomes lost. With a known map, the robot has a better chance of recovering its current position in case it gets lost when the wheels get stuck or when it is trying to go over 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 tedious for a human to do. Therefore, we 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. Besides, we want to explore a low cost system for developing new robots using a universal interface for both the simulated and real world robots.

II. 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 traverse the environment. Having a floor map beforehand 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. [1] Additionally, in order to see how the robot works based on the program we write, we have to build many different prototypes. This cost for building prototypes is also a problem.

III. COMMON 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 between hardware and software simple, we require both parts to share a common 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.

IV. SOFTWARE

A. 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]. Utilizing 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 encounters obstacles. One of the simplest ways that has often been used is a backtracking algorithm to map the floor [4]. However, one problem with this method is that the robot has to trace back the path used once it meets a

978-1-4577-0820-6/11/$26.00 ©2011 IEEE

dead-end, so it is not very efficient. Also, another problem of the backtracking algorithm is memory consumption. Since backtracking uses either recursion or stack, as the area to be mapped expands, the required memory size rapidly expands. Therefore, we developed a heuristic approach that uses breath first search algorithm to autonomously map the floor. The pseudo code is the following:

```
WHILE (unexplored grid exists)
    IF no block in front & not explored in front
        IF no block on left & not explored on left
            turn left
            go straight
        ELSE
            go straight
    ELSE
        IF no block on left & not explored on left
            turn left
            go straight
        ELSE IF no block on right & not explored on right
            turn right
            go straight
        ELSE
            Breath first search to find unexplored grid
```

Whenever the robot encounters an obstacle, it traces the shape counterclockwise and then expands the explored area spirally. Then, if it meets a dead-end, it uses breath first search to look for the closest unexplored grid and moves there. If there are no more unexplored areas, then the program terminates. With this method, we significantly reduce the cost of the mapping when compared to the previous backtracking method.

B. 3D Simulation

Even if people have an idea for developing a new technology or inventing a new product, the cost of implementation and building a prototype is always a problem. A software developer may not know anything about the hardware part, so smooth transition between the task of software developer and the hardware developer will be needed. 3D modeling technology helps this transition by making it possible for software developer to thoroughly test the program in a realistic environment. Since developing and testing in a virtual environment require less cost than in the real world, it could provide advantages over real world prototyping and testing.

1) Simbad 3D robot simulator

Simbad is a Java 3D robot simulator, which is developed for researchers who want a simple 3D environment to test algorithms for autonomous robotics. This tool provides a way to visualize a 3D environment and to retrieve sensory information from the environment through various types of sensors like vision sensors, range sensors, contact sensors, and etc [5].

We decided to use this freely available tool to test the algorithm we developed. 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 grids are used, the more mapping time and memory spaces are required. In order to solve this problem, we decided to use the grid only to navigate a robot and use some other sensors like the range sensors in Simbad and IR sensor in a real world to more accurately visualize the environment. In the figure, we can see both how the environment is put into a number of grids and how the environment is mapped. This way we can map the environment more accurately and much more efficiently (Figure 1).

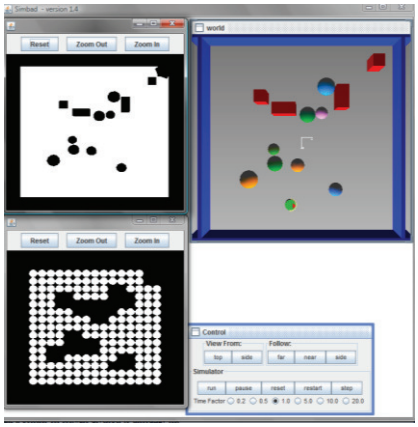

Figure 1. Simbad 3D robot simulator

One problem with this environment when we compare with the real world is that the virtual environment is too ideal because it always provides very accurate information. This is good to test the algorithm but it does not deal with a noise or error which we usually experience in a real world environment.

2) Virtual World Simulation

In order to test the program in a more realistic environment, we decided to use a virtual world environment, Second Life (SL), which can be accessed from anywhere in the world without any cost. SL provides a simple way to build complex objects, which can be controlled by Linden Scripting Language that has a similar syntax to C. One problem of using SL as a simulating environment is that even though it provides a way to communicate with the Internet outside of the SL, it does not provide a direct way to control the objects or avatars from outside of the client software. Therefore, in order to use the environment as a 3D robot simulator, it is required to establish a way to put every action in the environment under control.

Using a C# library provided by Open Metaverse Foundation [6], we developed a program which enables all the features of the common interface we listed in the section III.

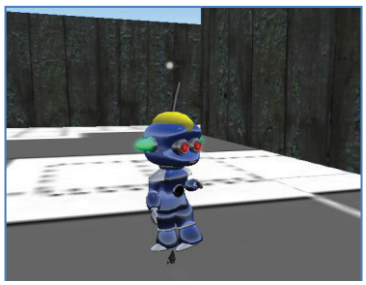

Figure 2. Avatar Bot with a sensor

We assume the avatar in SL as a robot we control (Figure 2); i.e., the avatar bot, controlled by our Java code, roams around the maze to build the map. The avatar wears a device that constantly feeds information of its location and facing direction to outside of SL, so that we can fully control the avatar by the Java code based on the information.

In order to implement the range censor, which is required to accurately map the floor in our program, we built a special object with the Linden Scripting Language. It throws a tiny particle to each of the 180 degrees around the avatar bot to determine if there are any objects around the avatar. If a particle hits any objects, it calculates the distance between the avatar and the object (Figure 3).

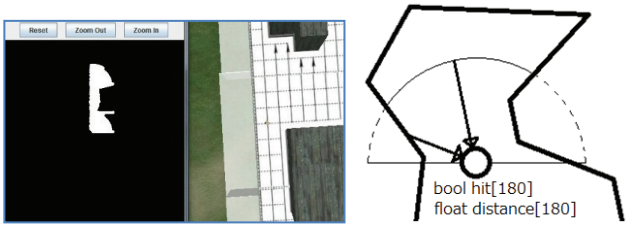

Figure 3. Range sensor in Second Life

Then, using the data obtained by the sensor, we tested our algorithm to map the floor (Figure 4).

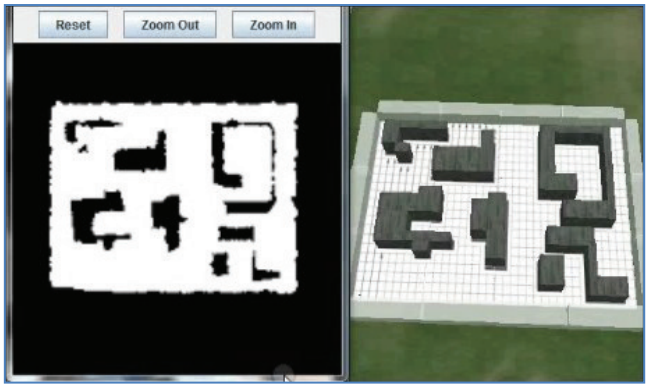

Figure 4. Floor Mapping in Second Life

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. As a result, we have not tried out our autonomous navigation algorithm in SL yet. Future work will be needed to address these kinds of problems.

V. HARDWARE

A. General Design

- The robot uses three Sharp Infrared Rangefinders (IR sensors) 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. This set up, 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 the robot turn 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 exclusively for keeping track of and updating the target position, which is the 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.

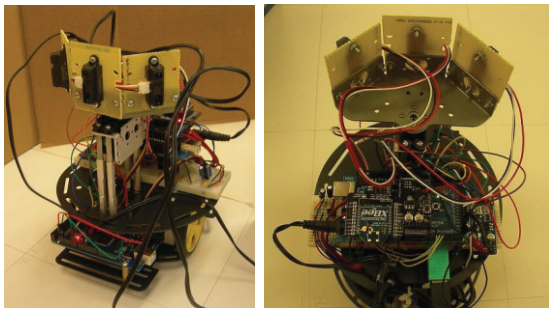

Figure 5. Floor mapping robot (front and rear views)

B. 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 easy. However, we encountered many problems when we tried to implement the interface in the real world.

C. Hardware Problems Experienced and Solutions

1) IR sensor's readings are wrong with sharp vertical edges

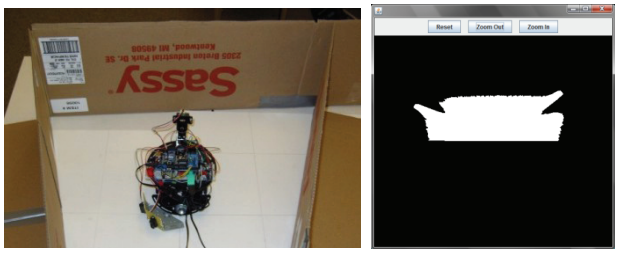

Figure 6. Error of IR sensor

When we did a full 180 degree scan with the IR sensors, we could see very strange spikes in figure 6 at the edges of the shape.

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.

One problem with this method is the assumption that the surface of the object to be measured is perfectly perpendicular to the IR beam. Thus, when used for mapping, the IR sensors report wrong readings when angled differently with the surface. In the extreme case, the IR beam does not return because the angle formed between the normal of the surface and the incident beam is too large. The two outward spikes seen in figure 6 have been formed this way.

Another problem with this method also seen in figure 6 is the two inward spikes right next to the outward spikes. We conjecture that the inward spikes are caused by multiple reflections as illustrated in figure 7. Instead of using angle 2 to measure the distance to A and A', multiple reflections cause the sensor to report distance to C and C' instead using angle 3.

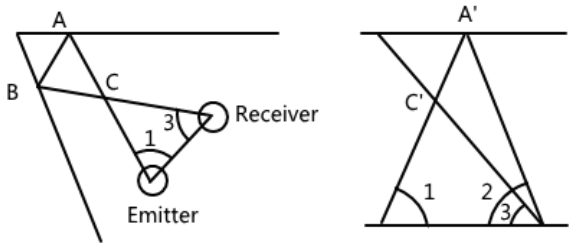

Figure 7. Sharp IR sensor error

Because we do not know exactly how the Sharp IR sensors work internally and rely on other sources [8], we hypothesize that the problem is because the beam width, which is parallel to the ground, of the IR sensor needs to be large (even though still very small compared to the cone shape of sonar sensors) in order to detect objects with unknown distances. This causes both the outward and inward spike problems because the surfaces we sweep form very different angles with the IR beam when we mount the IR sensor using conventional orientation: the axis going through the emitter and receiver is parallel to the ground. Since we assume the objects and the walls we map are perpendicular to the ground, we solved both problems with the spikes by mounting the IR sensors perpendicular to the ground instead (Figure 5). Figure 8 shows the result of the sweep.

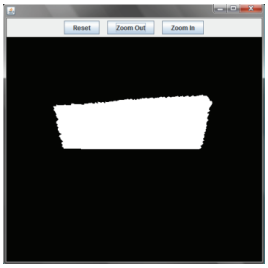

Figure 8. Result of modified IR sensor

This method provides a cheap alternative to the SICK LMS-200 since each sensor costs only about $12. The whole system with three sensors, one mounting bracket, and one servo cost less than $65, compared to the price of more than $5,500 for the SICK laser scanner. [1] The system works not only with rectilinear shapes but also round shapes (Figure 9).

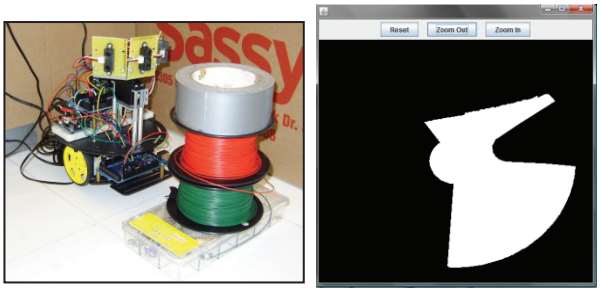

Figure 9. Mapping of a round duck tape

2) 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. We then used a new mouse with 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.

3) 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, and the wires can affect the direction of the robot. Furthermore, since low cost is our project's constraint, the wheel encoder we use can generate only 128 clocks per revolution, which makes turning to an angle with error rate of +/- 3 degrees impossible. 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 a table instead of the floor.

4) 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 to 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. Using the correction vector, the robot turns to the corrected angle and moves one micro 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 a high-end gaming mouse.

VI. Results

The result of the robot mapping a 4×4 grid is satisfactory (Figure 10). Even though there is much room for improvement, the robot has mapped the 4×4 grid well. Because of errors when turning coupled with the electro-magnetic interference which causes wrong compass readings, the resulting map has some unexpected spikes. However, we plan to deal with correcting errors by fusing multiple sensors in the future to obtain a much better map.

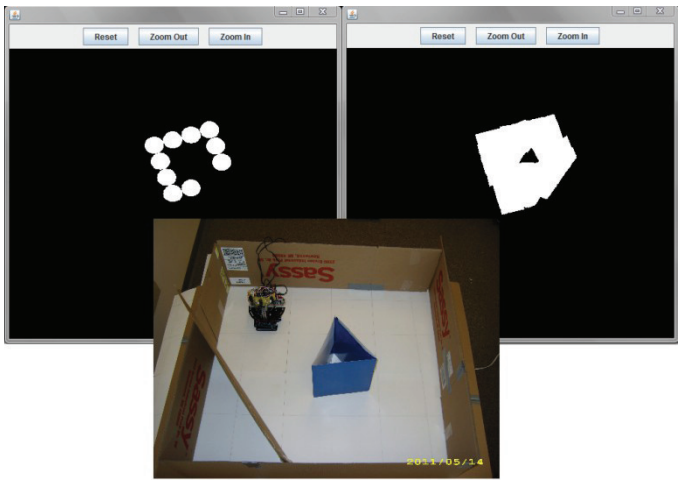

Figure 10. Real world result

VII. Potential Impact

Roomba robots will be smarter and will efficiently vacuum the whole house or only a designated area of 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 retrieve a soda from the refrigerator.

In addition, these robots could be useful in a real world situation in a retail store. The robot roams around the store and detects any changes of the layout of the product display to build a map which can then be used to traverse through the store. Coupled with RFID technology, the robot can autonomously inventory the products in the store including identifying if there are any expired items that need to be replaced.

VIII. Conclusion

In this paper we explored a way to minimize the cost to develop robots used to map an unknown environment. We initially tested our algorithm using the robot simulator Simbad. Our algorithm combines the grid-based approach for navigation and accurate sweeping to build highly detailed maps, which results in simpler navigation than with the latter alone and less time consuming than the pure grid-based approach. We then assembled the robot and implemented the algorithm in the real world. The transition from a perfect and controlled software environment to the real world robot was not as smooth as we had hoped. Trivial tasks in the simulation such as making the robot go straight, executing some arbitrary turn, and obtaining readings from the sensors had to be overcome using methods such as using an optical mouse for tracking, adjusting the robot's headings incrementally, and mounting the IR sensors in an unconventional way. These initiatives enabled us to obtain highly accurate tracking feedback from the mouse, and a very cheap and reliable sweeping system. Costing a little over $400, our robot has

demonstrated some promising initial result (Figure 10) despite having lots of room for improvement.

IX. Future Work

Our algorithm can be improved by using the 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 number of turns and amount of distance to the desired location. We can expand this further to allow the robot travel in and arbitrary path instead of a grid-based one.

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

Currently the mouse needs to be in contact with the floor to track the robot movement. In the future, we would like to modify the focus lens so that the mouse can be mounted higher with no contact to the ground so that it does not affect the robot's movement and also works with bumpy surfaces since the main technology behind optical mice is optical flow, which tracks movement by comparing multiple images taken at different times. Also, we could use a Kinect sensor mounted on the robot to map the 3D environment.

Furthermore, we can apply this work to everyday life situations by combining it with the concept of smart objects [7]. While the robot roams 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. Using the map and communicating with smart objects, robots will be able to travel from floor to floor using smart elevators, opening rooms with smart doors, and operating all other nearby smart objects in the room. The future of a housekeeping robot may not be too far.

X. Acknowledgment

We would like to extend our gratitude to the Department of Computer Science and Computer Engineering at the University of Arkansas for making this project possible.

References

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

[2] T. Sebastian. *Probabilistic Robotics*. Cambridge, Mass: MIT Press, 2005.

[3] 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, pp. 151–155.

[4] 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, pp. 842 - 843.

[5] "Simbad 3d Robot Simulator," Simbad Project, Internet: http://simbad.sourceforge.net/, 2011 [May 11, 2011].

[6] Open Metaverse Foundation. Internet: http://www.openmetaverse.org/, 2010 [May 11, 2011].

[7] A. Eguchi, 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.

[8] "Sensors – Sharp IR Range Finder." Society of Robots, Internet: http://www.societyofrobots.com/sensors_sharpirrange.shtml, 2011 [May 26, 2011].