**Abstract**

After natural disasters strike buildings can be left in an unsafe state for people to enter. When these buildings still need to be searched it would be useful and much safer to send a robot in. Making the robots autonomous would allow for a much greater area to be searched simultaneously with as many of them as you can get. This project was designed as a small beginning step towards that goal.

**1. Introduction**

I decided to do my own project for the senior capstone instead of a project for a local company or professor. I knew I wanted to work with a robot and the Intellibrain2 was available through the department, so the idea was to have it roam around a room taking measurements and use those measurements to construct a layout of the room.

Though this sounds like an extremely easy problem as soon as real hardware issues are factored into software design things become difficult. For this project the hardware issues that had to be worked with were; inaccurate range finder (radar), inconsistent servos, and a power jolt at the end of a movement cycle to one of the servos.

**2. Device Overview**

The hardware used was the Intellibrain2 distributed through Ridgesoft. They also provided the software to build the programs in java and upload them to the robot.

**2.1 The Robot**

IntelliBrain 2 Controller

* Java programmable
* 2 Serial ports
* 7 analog input ports
* 13 digital input/output ports
* 38 kHz infrared receiver/transmitter
* 8 servo ports
* 2 pulse width modulated DC motor parts
* 16 x 2 character LCD display
* Thumbwheel
* Buzzer
* 2 push buttons
* 6 LEDs

Nubotics WheelWatcher

* Prebuilt kit that just needs assembled
* Will provide more accurate coordinates than just time based servo commands

Parallax Ping Ultrasonic Sensor

* Already assembled
* Mounted on front end of Intellibrain 2 Chassis
* Accurate on straight shots up to at least 60 inches

Air Cable USB-Serial

* Wireless cable for connecting robot to computer
* Frequency determined with GUI and on controller



**2.2 RoboJDE**

RoboJDE is an extremely simple IDE. The only thing special about it are the 4 buttons towards the right. The wrench looking button is the compiler. The circuit looking button next to it loads the program to the robot and the play and stop button start and stop the program running on the robot.

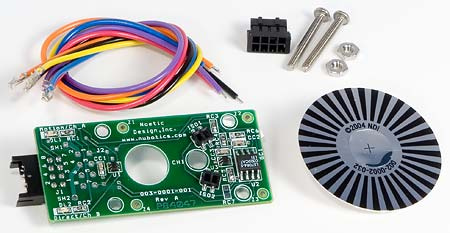


**3. Design**

When I began this project I broke it up into three steps; movement & measurement, data filter, and mapping. I decided to do them in that order as well since it was the logical process, but I never got past movement due to difficulties with the servos so I will only have basic design information for the other parts.

**3.1 Movement**

The initial plan for keeping track of movement was to use the Nubotics WheelWatcher hardware.



The problem with that was as you can see in the kit above there is only the connector for 1 side of the wires and I didn’t have any connectors available at the time. So instead the photo-resistor sensors mounted behind the wheels were used to count the number of wheel turns. Then using the following equations the robots position could be determined.

Dist = #slices \* (distance/slice)

X = Last\_X + cos(Ang) \* Dist

Y = Last\_Y + sin(Ang) \* Dist

Angle = Last\_Angle + (#slices \*(angle\_change/slice))

This all worked out fine except for one major problem. At the end of a movement cycle one wheel will usually just suddenly jolt forward or backwards a random amount. This problem was where at least 75% of the work time was spent, because there was no reason to move onto other problems or phases until this one was working correctly 100%. The reason this issue couldn’t be resolved at the software level is because there was no rhyme or reason as to when or to what degree it would happen. Sometimes it would jolt backwards 50 times in a row and sometimes it wouldn’t jolt for an entire run.

The main error with this problem was when it did the jolt and passed into a different slice and made the count increase faster than it should have. Since each slice counted for over 5 degrees of a turn the error added up very quickly. When it happened early in a run the robot would have coordinates in the program that were nowhere near the actual coordinates by the second spot.

The only solution I came up with for this problem would be to have a wheel similar to the wheelwatcher, but to have colors instead of just black and white. Using the color wheel I think you could overcome this problem, but this idea would also require other hardware to watch the color wheel so it would probably be easiest just to get a better robot.

**3.2 Measurement**

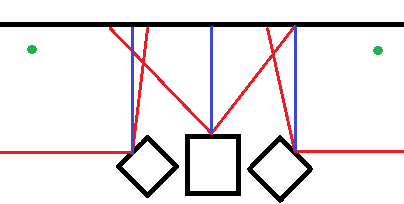
****

The measurement’s take place in step one, but aren’t handled until later. So these were seen, but not taken care of yet. The problem with radar is that it goes out at an angle so you aren’t always getting the measurement of what’s exactly in front of you. Just in the small test area used for this project (a small piece of hallway with 2 doors) the door frame would catch enough of the radar signal to determine the wall was there instead of 5 feet further.

**3.3 Data Filtering**

When the radar was doing measurements on a flat wall, only when it’s straight on would it give an exactly right measurement.

Red is radar edges and Blue is the measurement that would be obtained. Green dots are approximately where the program would think the walls were.



The main idea of filtering the flat areas would have been to just use the furthest out of the semi-circle. Much more advanced filtering would have been needed for corners and actual advanced room layouts, but since I never got to it I didn’t spend a lot of time figuring that part out. These problems could have once again been eliminated simply by using better hardware, in this instance a laser rangefinder.

**3.4 Mapping**

Mapping would have been just a simple, extra step at the end. Once the coordinates were determined, using OpenGL, you could simply build lines, or polygons for 3d, where the walls were determined to be and use the keyboard or mouse to navigate through the display. Since the project didn’t get near this far no example was made.

**4. Scheduling**

The plan was to go this sequentially. That plan was followed, but with the problem being at a very early stage I quickly fell behind and never caught back up.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Weeks  **Tasks** | 01/29 to 02/05 | 02/05 to 02/12 | 02/12 to 02/19 | 02/19 to 02/26 | 02/26 to 03/05 | 03/05 to 03/12 | 03/12 to 03/19 | 03/19 to 03/26 | 03/26 to 04/02 | 04/02 to 04/09 |
| Initial Planning | Proposed |  |  |  |  |  |  |  |  |  |
| (actual) | Actual |  |  |  |  |  |  |  |  |  |
| Initial Design | Proposed |  |  |  |  |  |  |  |  |  |
| (actual) | Actual |  |  |  |  |  |  |  |  |  |
| Initial Testing | Proposed |  |  |  |  |  |  |  |  |  |
| (actual) | Actual |  |  |  |  |  |  |  |  |  |
| Write Proposal | Proposed |  |  |  |  |  |  |  |  |  |
| (actual) | Actual |  |  |  |  |  |  |  |  |  |
| Proposal Presentation |  | Proposed |  |  |  |  |  |  |  |  |
| (actual) |  |  | Actual |  |  |  |  |  |  |  |
| Build Wheel Watcher |  | Proposed |  |  |  |  |  |  |  |  |
| (actual) | http://associate.com/gantt/images/008000.gif |  |  |  |  |  |  |  |  |  |
| Final Robot Build |  | Proposed |  |  |  |  |  |  |  |  |
| (actual) |  |  | Actual |  |  |  |  |  |  |  |
| Movement Code |  |  | Proposed |  |  |  |  |  |  |  |
| (actual) |  |  | Actual | Actual | Actual | Actual | Actual | Actual | Actual | Actual |
| Coordinate Code |  |  |  | Proposed | Proposed |  |  |  |  |  |
| (actual) | http://associate.com/gantt/images/008000.gif |  |  |  |  |  |  |  |  |  |
| Visualization Code |  |  |  |  | Proposed | Proposed |  |  |  |  |
| (actual) | http://associate.com/gantt/images/008000.gif |  |  |  |  |  |  |  |  |  |
| Testing |  | Proposed | Proposed | Proposed | Proposed | Proposed | Proposed | Proposed | Proposed | Proposed |
| (actual) |  | Actual | Actual | Actual | Actual | Actual | Actual | Actual | Actual | Actual |
| Code Review |  |  |  |  |  |  |  | Proposed |  |  |
| (actual) |  |  |  |  |  |  |  | Actual |  |  |
| Final Fixes |  |  |  |  |  |  |  | Proposed | Proposed |  |
| (actual) | http://associate.com/gantt/images/008000.gif |  |  |  |  |  |  |  |  |  |
| Final Writeup |  |  |  |  |  |  |  |  | Proposed | Proposed |
| (actual) |  |  |  |  |  |  |  |  |  |  |
| Final Presentation |  |  |  |  |  |  |  |  | Proposed |  |
| (actual) |  |  |  |  |  |  |  |  | Actual |  |
| **Tasks**  Weeks | 01/29  to  02/05 | 02/05  to  02/12 | 02/12  to  02/19 | 02/19  to  02/26 | 02/26  to  03/05 | 03/05  to  03/12 | 03/12  to  03/19 | 03/19  to  03/26 | 03/26  to  04/02 | 04/02  to  04/09 |

The green blocks are things I didn’t get to because of the servos.

**5. Summary**

While I still think this project idea was a good one, the way I went about it once I came up against the problems didn’t work at all. I had hoped that I could work around the hardware issues in the software and that just wasn’t feasible for this scenario. If I were able to do it all over again I would just build my own robot instead of using a prebuilt system that has problems in it I don’t know about.

Though it was a completely different type of learning experience than most people get out of a capstone project I did learn a lot from this. Mostly I learned about some of my own limitations and how much I still have to learn when it comes to hardware and software interaction in robotics. You can’t just expect to work around a major hardware issue in software because it will never work as well as making sure everything works properly on both sides.