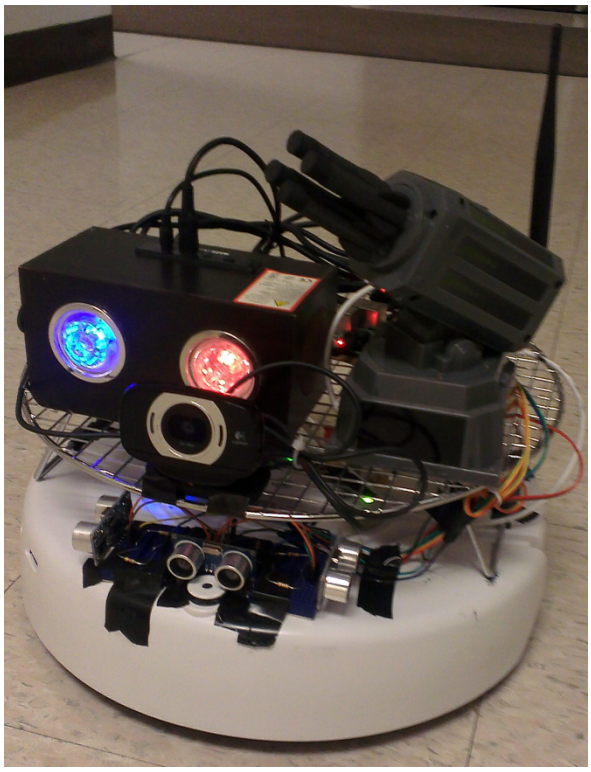
Autonomous Indoor Mapping

WALL – E

Eric Yong
David Roper
Kassem Nabha
Christopher Jackson
Ryan Carroll



Executive Summary:

This report provides design and analysis of autonomously mapping a building with panoramic images. Methods include using a robot to navigate a structure by using sensors and taking pictures. Algorithms and methods are listed further in report. Results showed that this is possible using relatively inexpensive equipment.

This report finds that mapping can be done reliably.
Achievements discussed include:

- Having robot follow hallways and take pictures at set distances.
- Integrating an embedded computer system with the robot to control a High Definition Web Camera.
- Interfacing all components so that the system is completely mobile.
- Uploading coordinates to a web database
- Integrating panoramic pictures onto user friendly website

Table of Contents:

Project Summary:

Our goal is to use an iRobot Create platform to explore a building and map it with panoramic images. The robot would estimate its own position, take and store images in order to stitch all the pictures together on a computer. Depending on accuracy of position measurements, the pictures could then be programmatically placed in a database corresponding to map points.

The project would be successfully completed if panoramic images and their measured positions correspond to the actual building in an accurate 2D map with precise "street view" areas.

Project Statement:

When exploring new areas, it's advantageous to view the new surroundings ahead of time. This concept is infinitely more critical in dangerous areas, such as a burning building. The WALL-E team has surmised a way of viewing foreign terrain, in a cost effective manner.

We began this project with the goal of providing a safer environment for first responders. With the indoor mapping robot, we hope to instantly provide Armed Military Personnel or Emergency Services the information they need to safely enter buildings to perform search and rescue missions. The robot would be controlled externally via the Raspberry Pi, which would communicate orders from a human-monitored computer

Our product could be used in various high stakes, dangerous situations. For example, the robot could be used to inspect a house with a gas leak. Firefighters could have a full layout of debris and any possible injured civilians, before jumping into the setting. The robot could also be used to inspect or locate explosives, without endangering soldiers. The robot was designed on a limited budget. The true value of the product is the potential to save lives.

The product could also be marketed commercially for civilian use. The robot's collected data would be compiled into a smartphone application to provide walking directions for pedestrians. The directions would include indoor maps. For example, new students at a university could find the way to their classrooms prior to physically entering the building.

Objectives / Goals / Requirements:

Our goal is to construct and program an independently-operating robot that can take panoramic pictures at a specified rate. The Raspberry Pi would determine rates of image-taking depending on the distance from an object, which is measured by sensors. We then will upload the pictures to an image stitching program on a separate computer. The stitched images will then be put onto a website similar to Google's "Street View". To accomplish such a comprehensive task, we split up the goals into separate objectives for each team member and created deadlines for each task.

During the Fall Semester we determined a set of goals and details we want to accomplish by the end of the year. We decided that a "wall hugging" robot, reliable image stitching programming, and a logic controlled webcam would be suitable goals for the Fall Semester. This would leave us with the tasks of programming an autonomous controlled robot, uploading pictures and coordinates to a database, and possibly create an Android application GUI.

Over the Winter Break and the Spring Semester we redesigned the robot's control systems from the ground up. We determined that moving all control to the Raspberry Pi increased the versatility and development speed of the robot. We changed our sensors to sonar sensors to help the robot navigate the building in a more intuitive way. We also decided to

organize all the data into a database so that we could easily order and store data we recorded in an efficient manner.

Schedule:

Fall Semester:

1) Find an accurate and reliable image stitching program
2) Experiment taking spherical (panoramic) pictures
3) Design a mechanism to turn the camera, or program the robot to turn accurately
4) Test sensors and functionality
5) Create a program to have the robot independently navigate a building
6) Research image recognition, with the goal of recognizing room numbers
7) Program Raspberry Pi to make the camera take pictures at a specified rate
8) Implement Wi-Fi to upload pictures wirelessly

Spring Semester:

1) Implement Raspberry Pi as sole source of control
2) Combine all mechanical, programming and Wi-Fi elements
3) Perfect independently traveling robot by calculating distance traveled
4) Rough mapping of a building floor
5) Design website modeled after “Street View”
6) Map a building floor

Design Constraints:

Our design will utilize the Raspberry Pi, employing three external sonar sensors. Time and budget constraints could limit extra features and more accurate and informative data. For example, external sensors could gather more data. Total weight of equipment will not exceed 15lbs because size of the robot platform is limited. We utilized an HD web camera to obtain high quality images, which was powered through a USB hub. The robot's memory will also determine the extent of our mapping system. We initially used an external power supply, but ended up powering all the electronics directly through the iRobot. Total run time with a full charge was limited to four hours.

Success:

We then set up a list of requirements to define a success for our project. We decided that the final product should include:

-Program robot to roam hallways autonomously
-Robot must take pictures with sufficient overlap
-Stitched pictures will display an accurate panoramic representation of building
-Coordinates are precise and recorded to database
-Robot path and panoramic pictures are displayed on website

Approach:

Our first step for this project was to decide on what type of robot we want to utilize. We chose to use the iRobot, which is more commonly known as the Roomba, but without the vacuum mechanisms. The iRobot proved to be sufficient in meeting our design requirements. The robot's three wheels and low center of gravity provided the stability for decent quality pictures. The iRobot was also large enough to carry all the electronics we required, such as, the Wi-Fi antenna, USB hub, web camera, voltage divider circuit with heat sink, and optional nerf missile launcher. We initially used the built-in touch sensor and iRobot Create Platform that came included with the robot. Although we didn't utilize

Considering the amount of work this project requires, we sought to tackle each task as efficiently as possible. We're trying to avoid any pitfalls in the future while using the programming the language that is best for the task.

-Individually test components and sensors

-Design method to take panoramic pictures

1) Image Stitching through the use of Microsoft's Image Composite Editor or comparable program.
2) Rotating web camera by programming the robot to turn with 18° increments
3) Time picture taking with rotation
4) Store and upload pictures via Wi-Fi

-Program robot to navigate a hallway while keeping track of its location

- Combine robot and camera system to take pictures and link with location
- Map a building floor
- Determine if image recognition is feasible
- Create basic application to view images

Challenges:

We face two difficult tasks of programming the Raspberry Pi to control the camera, and control the iRobot Create Platform to perform the tasks. We decided to use Python on the Linux platform with the Raspberry Pi. Although we weren't as experienced with the Python language, we were informed by experts that it would alleviate the task of controlling the camera. Originally we had chosen to use the iRobot Command Module to control the robot. We found that most of the native libraries on the robot were written in C, so we decided to continue with that language. We found that controlling the robot through a serial communication line was much easier when developing and debugging, but it presented the challenge of translating the C library into Python. To power the robot while it was mobile, we used the iRobot battery to supply power to the Raspberry Pi and camera. We used a 14.4V, 3.3Ah battery with several 5V voltage regulators. This ensured the Raspberry Pi and camera received a clean signal and were able to function and not be tied into a stationary power supply.

Our team is relatively experienced in programming, but we face a few obstacles. The Python language is fairly new to us, so controlling the camera and iRobot with it has been an obstacle. We also face the challenge of processing the pictures and creating a cohesive user interface program to display the "Street View".

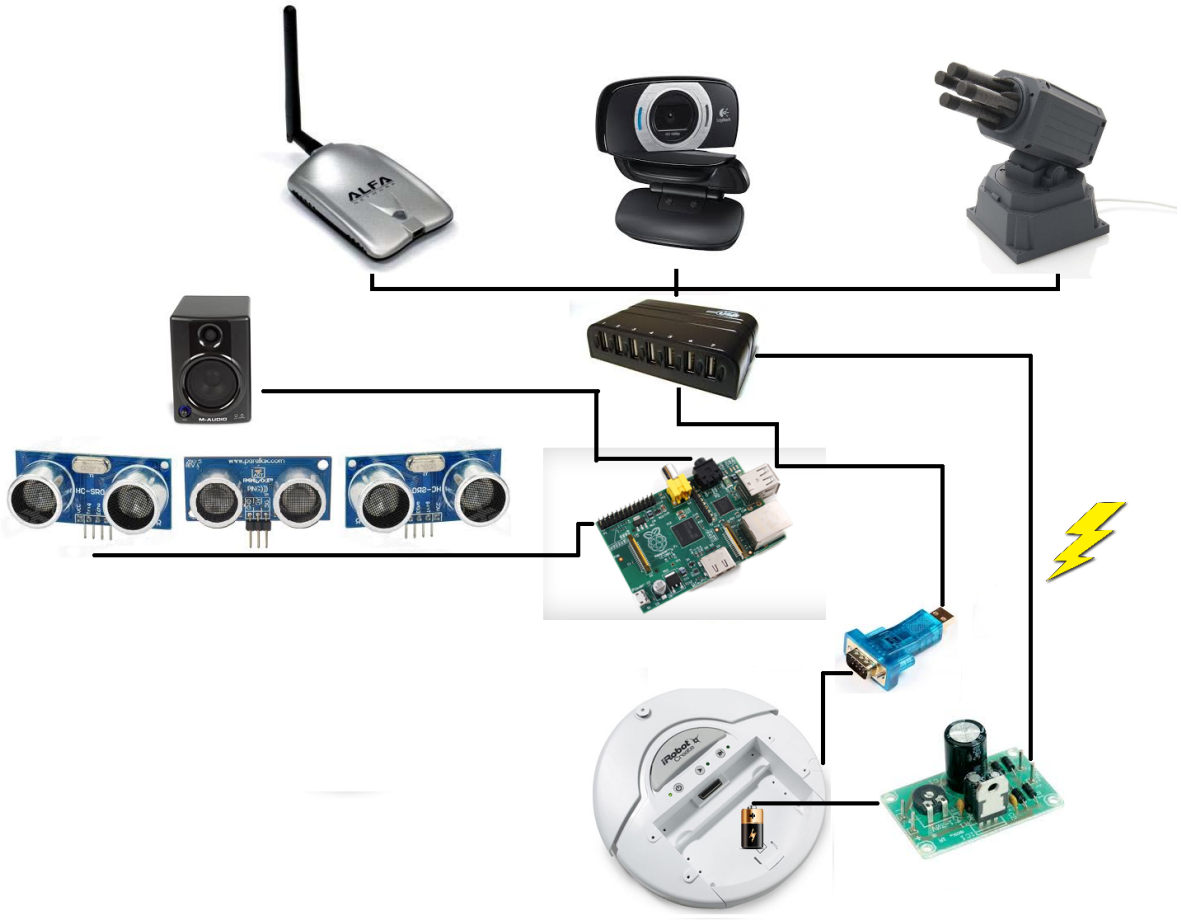
One problem we faced when programming the robot was our discovery that the iRobot's wheel encoders were somewhat inaccurate. For example, we would write in the code to turn 90

degrees but the robot would instead turn about 150 degrees. A possible solution may be to invest in a digital compass for more accurate angle measurements.

Design:

Our project involves extensive hardware and software elements. It's crucial that both systems are fully functional and can be incorporated into a single operable unit.

Wiring Diagram:



Overall Design:

The iRobot is controlled solely by the Raspberry Pi via a serial communication link. A USB-to-Serial convertor allows the Raspberry Pi to use its USB 2.0 ports send signals directly to the iRobot's communication port. The Raspberry Pi uses its General Purpose Input/output pins to utilize the sonar sensors to determine the range of the walls in front, left, and right.

The iRobot Battery is used to provide power to the entire platform. From the battery, a line goes to two 5V Voltage regulators. Two were used to ensure that we maintained a steady, low noise, 5 Volts to the Raspberry Pi. One Voltage regulator directly feeds a USB hub which then supplies power to the other devices, and one voltage regulator supplies power directly to the Raspberry Pi.

The Wi-Fi adapter, the web camera, and the (optional) missile launcher are powered via the USB hub. The USB hub also serves as a central communication core as well allowing all the devices to talk to the Raspberry Pi.

Once data and pictures have been collected, it is shipped to another computer over Wi-Fi. That pictures are then stitched together and associated with the location data with where they were taken. The data and pictures are then accessible through a website.

Hardware:

- iRobot Create Platform
- iRobot Create Battery 14.4V, 3.3Ah
- 4 5V Voltage Regulators
- Raspberry Pi
- USB-to-Serial Connector
- USB Web Camera (Logitech C615)
- USB Missile Launcher
- USB Hub
- USB Wi-Fi adapter
- 3 Sonar Sensors
- Speaker

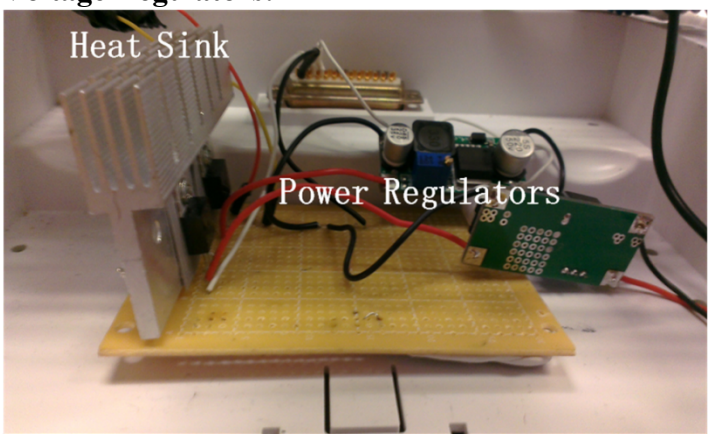
iRobot Create Platform:

The iRobot Create Platform provided us with a cost effective and reliable platform for our project. Its three wheels and low center of gravity allowed it to traverse most indoor terrain with ease while providing a stable platform for sensors and computing equipment.

The iRobot Create's main purpose was to transport the equipment throughout thee building so that we could effectively map it.

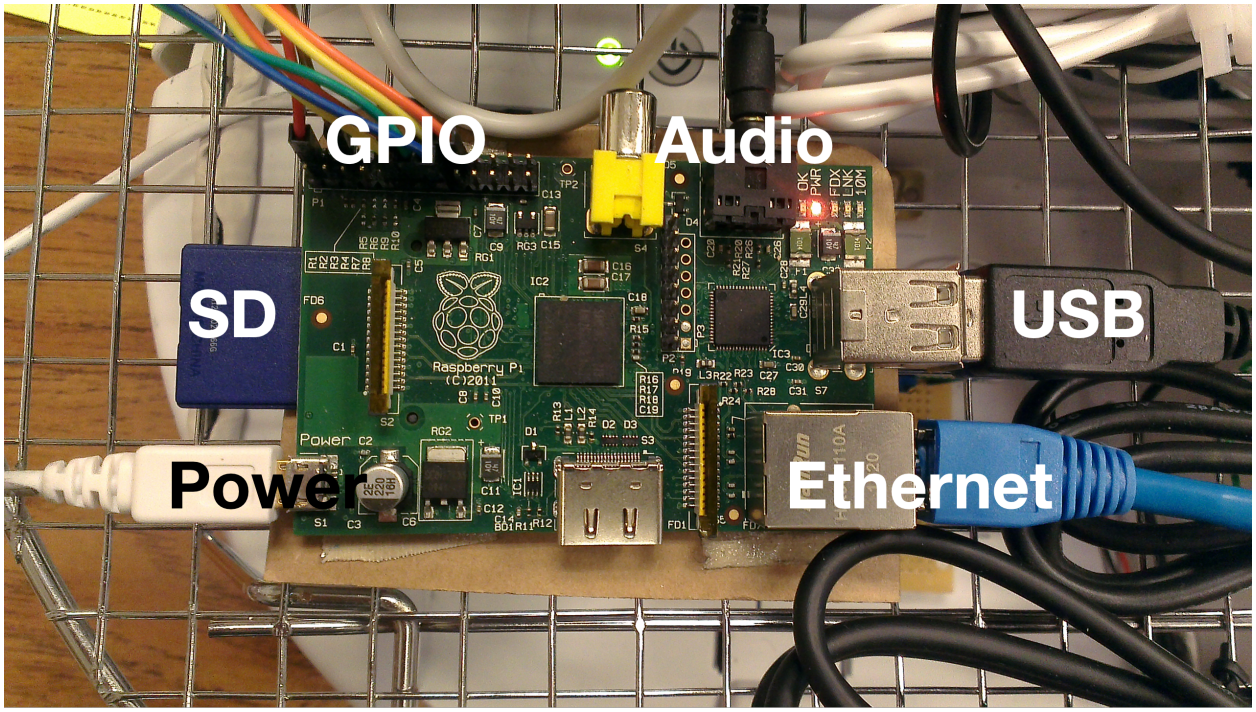
The iRobot Create Battery pack provided power for the iRobot Create as well as all the peripherals and computing hardware. It is rated at 14.4V, 3.3Ah. It is guaranteed to last for 100 minutes but testing yielded it was effective up to 4 hours.

Voltage Regulators:



To power the rest of the robot, the iRobot Batter pack is used but with the addition of 4 power regulators. The power regulators are hooked in pairs that are in series, to ensure a stable 5V signal for the Raspberry Pi and other devices that required that signal.

Raspberry Pi:



The Raspberry Pi acts as the command and control center of the robot. It runs the Raspbian - Linux operating system and the Python programming language is used extensively to control the iRobot and its peripherals. The SD card holds the operating system as well as the data. The GPIO (General Purpose Input / Output) are used to communicate with the sonar sensors. The USB ports are used to connect the Raspberry Pi to the Wi-Fi adapter, the Web Camera, and the (optional) Missile Launcher. The Ethernet port is used to tether a computer to the Raspberry Pi to expedite debugging and development.

USB-to-Serial Adapter:

The USB-to-Serial Adapter was used to allow the Raspberry Pi to easily communicate with the iRobot. The iRobot had a serial communication port which coupled perfectly with the adapter.

Logitech C615 - Webcam :

The Logitech C615 webcam was easy to integrate with the Raspberry Pi, even though no official drivers exist. It is capable of 1080p video and still images as well.

Wi-Fi Adapter :

The Wi-Fi Adapter was used to let the robot communicate wirelessly with the computers and access the internet.

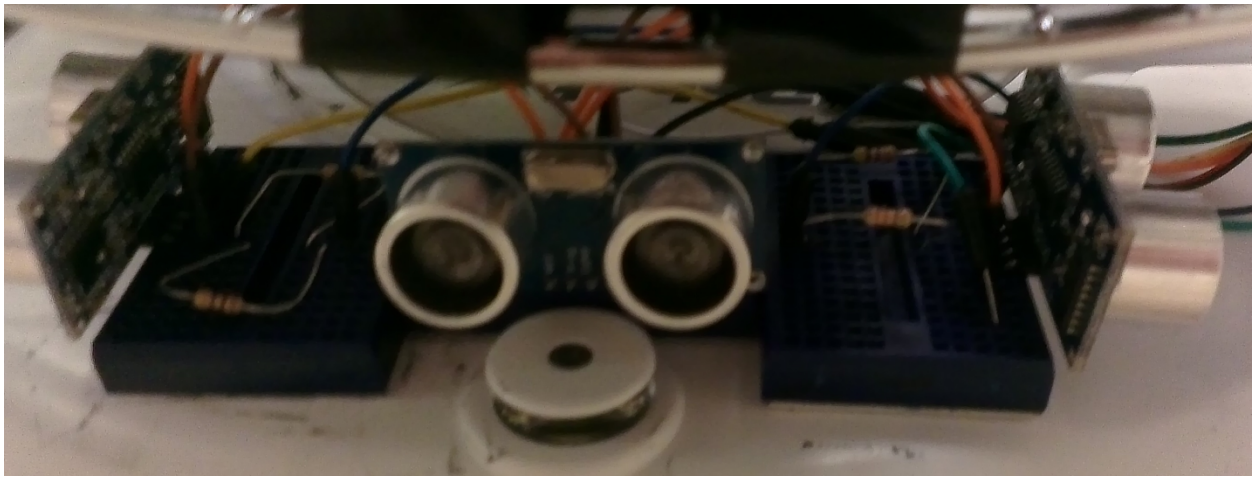
Missile Launcher :

Utilized strictly for self-defense situations and stress relieving actions. Controlled via custom drivers written in Python.

USB Hub :

The USB Hub serves two purposes it distributes power as well as handles all communication to and from the Raspberry Pi.

Sonar Sensors:



The Sonar Sensors are the main sensor on the robot. They look left, center, and right to help determine the surroundings of the robot. They are accurate up to 2m and lose accuracy as distance increases. At 4m, they are unreliable. They emit a high frequency pulse every 0.07 seconds and measure the time it takes for the sound to reflect back. Knowing the speed of sound and the time it took to go to the target and come back allowed us to determine the distance.

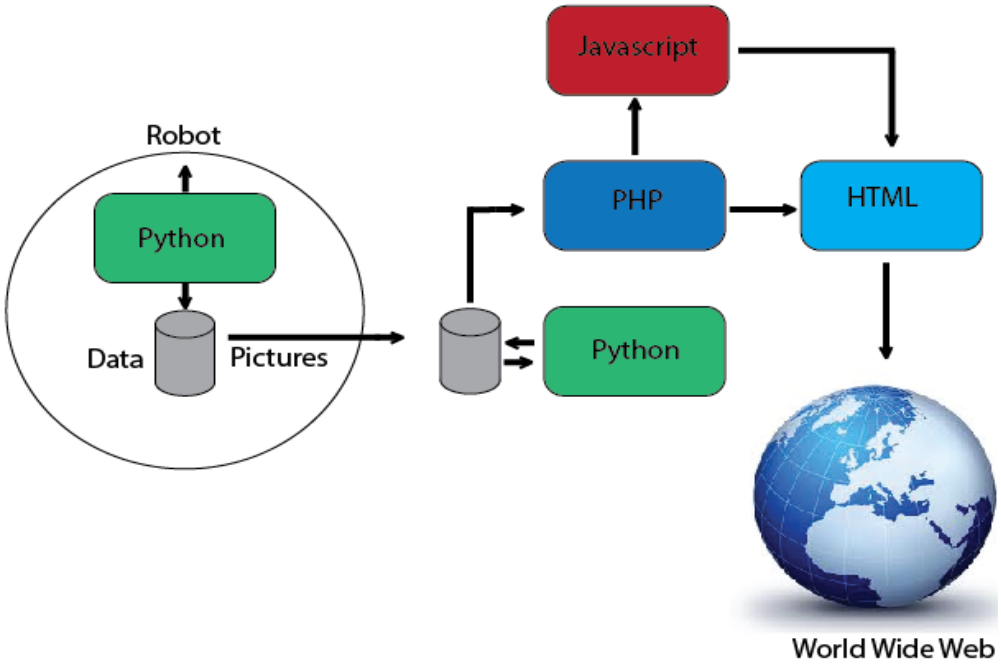
Speaker:

The speaker was used as a audible warning device to let others know the robot was there. It also served as an audible debugging device. Having the robot play specific sounds during specific actions allowed us to see if the robot skipped a function or was running amuck.

Software:

- AI Simulation
- Raspberry Pi Linux system with Python Implementation
    - General Robot Control
    - Artificial Intelligence
    - Control System
    - Database System
    - HTML, PHP, JavaScript Webpage on Robot
- Image stitching program
- Webpage for user to access data

Software Flowchart:



Raspberry Pi Linux System with Python Implementation

General Robot Control:

To every mechanical part in the world there is a brain that runs it controls it or monitors it. Wither the brain is human or human made, a brain has to be there. The Raspberry Pi, a powerful computer-on-a-chip, was chosen to fulfill the role of the brain. The Raspberry Pi allowed modules to be added to the Robot that were not possible before One example is Wi-Fi which allowed full human control and monitoring , plus the transfer of data and pictures.

The Raspberry Pi contains a lot of components that are user friendly. The Raspberry Pi contains all the code that talks to the hardware connecting to it, which include the robot, sonar sensors, Wi-Fi, speakers, missile launcher, and a camera. The Raspberry Pi allowed us to take efficiently handle input and output data and to have full control over all functionalities. An example on input transition is the pictures gathered from the camera, or the distance that the robot is far from the wall gathered from the sonar sensors. An examples on output transmission of data is the commands sent to the Robot using serial communication, or wav files to be played by the speaker to ease debugging and make the system more real and fun.

Each piece of hardware connected to the Raspberry Pi works together in our software algorithm. To start, the robot is powered on when a user clicks a button on the Robot that will power the system on through a battery located on the Robot. The battery feeds power to a USB hub and the hub distributes the power to the peripherals of the system, but the robot is fed straight from the battery. Voltage regulators that drop the 14.4V supplied from the Robot's battery to a steady 5V.

On startup, the Raspberry Pi boots up and connects to Wi-Fi. Connecting to Wi-Fi was important as it gave the user remote access to the Raspberry Pi. The Raspberry Pi could also be communicated with through an Ethernet cable for debugging and testing purposes. When the user connects to the Raspberry Pi, a python script runs that controls the Robot, cam, sensors, missile launcher and speakers. The script first asks the user to input the speed to drive the Robot

and the Robot's location and orientation of the Robot. Once the input from the user has been given, the AI on the Robot takes control. Note that the system is also capable of being controlled by the user instead of the AI. The user in charge can control the Robot, missile launcher, and camera using different keys on the keyboard.

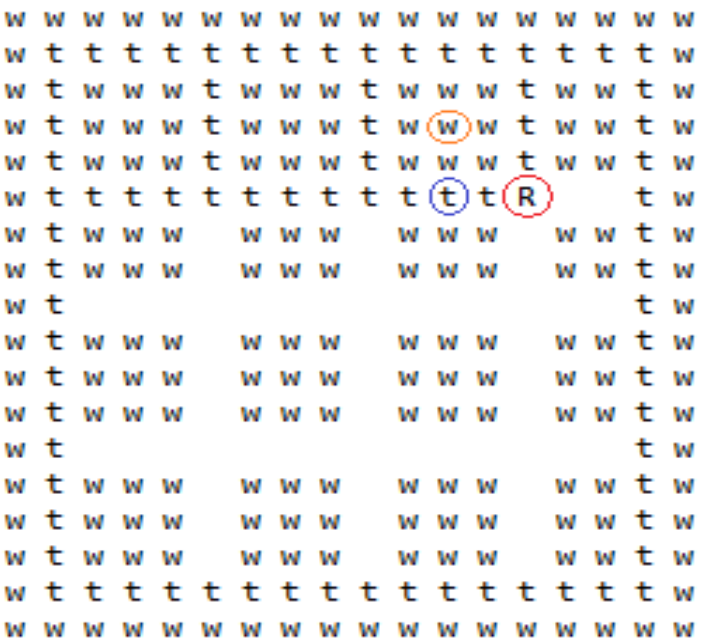
When the AI is in control, the first thing it does is check its distance from the left and right of the wall using the sonar sensors. The sensors send a pulse of sound and receive it. By taking the difference of time of when the sound pulse was sent and received, the AI is capable of knowing the distance from the wall. If the data returned back shows that the Robot is closer to one wall then the other the AI will tell the Raspberry Pi to give a command to the Robot to turn towards the middle of the hallway. If the difference is small, the Robot will speed up or slow down its left or right drive wheel accordingly so that it moves to the center of the hall. If the difference is too large, the robot will turn 90 degrees in the direction of the further distance and move to the center, the rotate back towards the direction it was originally facing.

Another benefit of the sensors is that the measurements collected provides a way to determine when is the right time to make a turn. The three sonar sensors installed on the Robot on the front, left, and right gives the AI a measurement that is too large, AI would decide that the Robot is at a corner and command the robot to turn in the proper direction.

During this entire process, the AI is collecting location data, pictures, and wall measurements and storing it in a database. This database is used for later processing and to post data to a website hosted on the Robot. More in-depth discussion of how the AI works, more specifically algorithm will be mentioned later.

.

AI Simulation:



The picture shown above is the Path Finding Simulation. The Path Finding Simulation is a simulation written in java code which eventually became our robot's AI. The letter "w" circle in the orange circle represents a wall, the letter "t," circled in blue, represents the trail of where the robot has been. The letter "R" is the robot that is walking the hallway.

For the robot to walk the entire hallway without taking picture, it took about 15 min to 25 min. Due to this very time consuming testing process, it was decided to write the Path Finding Simulation to save time on testing the robot. It also allowed testing in almost all kinds of hallways and navigate through them in a very short amount of time.

The Path Finding Simulation priority goes with left as top priority, then straight, lastly right. It will also check if it has been to the location in front of it or next to it. It has a function to turn around and go backwards if it sees there is a dead end in front of it or it has been to the location in front of the robot.

Artificial Intelligence:

Although code was developed that allows for manual control of the robot, an integral part of the WALL-E system was the development of an autonomous artificial intelligence system to navigate unknown indoor terrain. This was accomplished in two major stages: a simulation stage and an implementation stage. The simulation stage, as described in the "AI Simulation" section above, was designed to create a logical algorithm to theoretically enable the robot to navigate through most buildings. The implementation stage, described in detail below, translated the basic logical algorithm to the real world.

The AI simulation algorithm used a left-straight-right priority system whenever the robot encountered a decision point such as an intersection. When the robot would reach a dead end, it would turn around and go back the way it came. Moreover, it featured the ability for the robot to keep track of all of the locations it had been. This information was then used along with the left-straight-right priority system to prevent the robot from exploring the same area twice.

Transferring the simulation AI to the real robot's python code required all of the above features to be realized using the limited capabilities of the robot. Due to early testing, it was observed that the sonar sensors were accurate only up to about 2 meters, and extremely unreliable beyond 4 meters. As a result, the any distance greater than 4 meters was considered "infinite" in the python code implementation. An infinite sonar reading on the left or right would then represent a direction that the robot could possibly turn. This was then used to define the left-straight-right priority. For example, if the left sensor read infinite, the robot would go left; if the left and straight sensors were finite but the right infinite, it would go right. The straight sensor was slightly different, as it only checked if the sensor reading was greater than 1 meter (not 4 meters). This was done so that the robot would stop if anything passed in front of it within 1 meter to prevent collisions.

However, the real world presented several challenges not covered by the simulation. The robot had a finite, nonzero size and the sensors were mounted on the front. As a result, if the robot were to turn as soon as a sensor read infinite, it would not be able to go around a corner (see Figure 1). This problem was solved by having the robot move forward a small distance after a sensor reads infinite. It would then move forward in the new direction while reading its sensors until the sensor that previously read infinite was finite again. Then the robot would continue with its normal routine (see Figure 2).

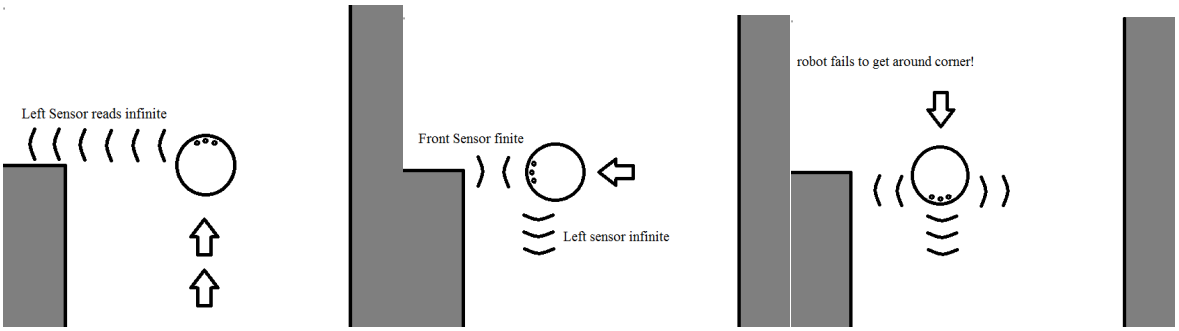


Figure 1

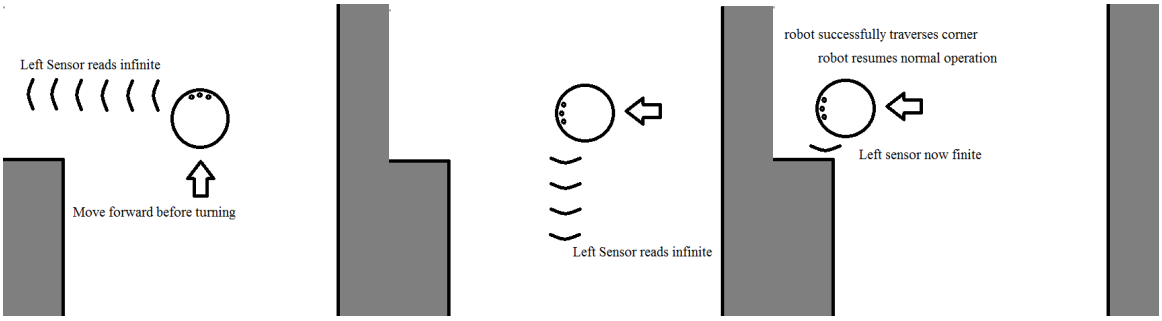


Figure 2

Another challenge was the discovery that the sonar sensors would sometimes give erroneous infinite readings, likely due to bad sonar reflections. This was very problematic, because an incorrect sonar reading would cause the AI to treat a wall or obstacle as if it were open space. This was solved by implementing a double-check into the code, so that when an infinite reading was detected on a side sensor, the robot would turn and check it again with the front sensor to confirm the reading before proceeding. These modifications to the basic algorithm allowed the robot to implement the simulation's priority system.

In order to fully implement the simulation algorithm, the robot needed a means of storing every location it had been as it went. By using the robot's speed and measuring time, the python code was able to accurately provide the distance travelled every time the robot stopped. The robot would stop whenever it took a set of panoramic pictures (usually every 5 meters, but changeable in the code), and whenever it reached a decision point. This yielded a set of discrete points at various (x,y) locations, which were then stored into arrays. However, it was necessary to have all of the points in between for the simulation AI to be effective. To solve this problem, when the robot stopped and measured the distance, it would fill up the appropriate array from its previous point to its current point in increments of a fraction of a meter which was adjustable in the code (usually 0.1 or 0.2 meters). For example, suppose the robot traveled 5 meters in the x-direction from the origin, and then stopped to take panoramic pictures. Its starting position would be (0,0) and ending position (5,0). The x-array, however, would be filled with values from 0 to 5 in increments of 0.2 meters: [0, 0.2, 0.4, 0.6, …, 4.6, 4.8, 5.0]. This data was then later used to be able to tell where the robot had previously travelled, with an accuracy of 0.2 meters. The full version of the robot artificial intelligence not only checked the sensors, but also checked whether or not the robot had already been to a given location before applying the left-straight-right priority.

Control System:

Programming the control systems on the robot proved to be a difficult task. The algorithm we used would tell the robot to move every 4 meters, and take series of pictures over 360 degrees. The robot would then continue in a straight path along the hallway for another 4 meters. However, the Wall-e robot would tend to veer slightly left as it roamed down the hallway. Consequently, our robot would perform a series of panoramics until it ran into a wall. Fixing this problem was critical to the overall success of the project.

Firstly, finding a way of controlling each wheel individually was our primary task. This was accomplished by accessing the separate motor controls through the raspberry pi. Then, we programmed the sonar sensors to measure the distance of the left and right walls every .07 seconds. The distance between the opposite walls were then compared to each other. If the distance to the right was greater than a tenth of an inch greater than the left, then the left wheel would increase it's speed 5mm/s. The wheel velocity would return to its original velocity when the difference between the left and right walls was less than a tenth of an inch.

The newly implemented algorithm was a great success. Our robot could migrate every hallway with ease. The only issue we encountered was that our robot would drive slight a sinusoidal path down the hallway. This was a minor problem because the wavering route did not show up on the web database, and it didn't affect our panoramics. Overall, the new control system was very beneficial.

Database System:

To effectively store all the data in an organized and efficient fashion, a database was created using sqlite3. sqlite3 is very computationally light and has restrictions such as only having 5 data types. The database was only being used to store strings and integers, so this restriction did not affect the project. As the robot traversed the halls we would store data in 3 tables: Current Robot Data, Pictures Data, and Travel Data. Current Robot Data just stored the current position and speed of the robot so that if the program crashed, the last location was stored on the robot. Pictures Data recorded which pictures were related to each location. Travel Data recorded travel distances, wall distances, and orientation relative to start at set distances and whenever the robot stopped.

By simply transferring the database file, all the data would be available on any computer. This is especially useful when it comes to using the data for websites and batch scripts to access and analyze the data.

HTML, PHP, JavaScript Webpage on Robot:

In order to view what the robot was doing at any given time and to look at its recorded data, a website was created. The website was formatted using HTML which set the basic layout. PHP was used to pull data from the database and make it accessible to HTML and JavaScript. The JavaScript was used mostly for making the website more appealing and allowing a basic map to be drawn tracking the robot's distance. As the robot would roam the building and write to the Travel Data table of the database, the JavaScript, pulling info via PHP, would then drawn lines to visualize the robot's movement. Below is an image of the webpage that was created to be broadcast on the robot.

WALL-E Dashboard:

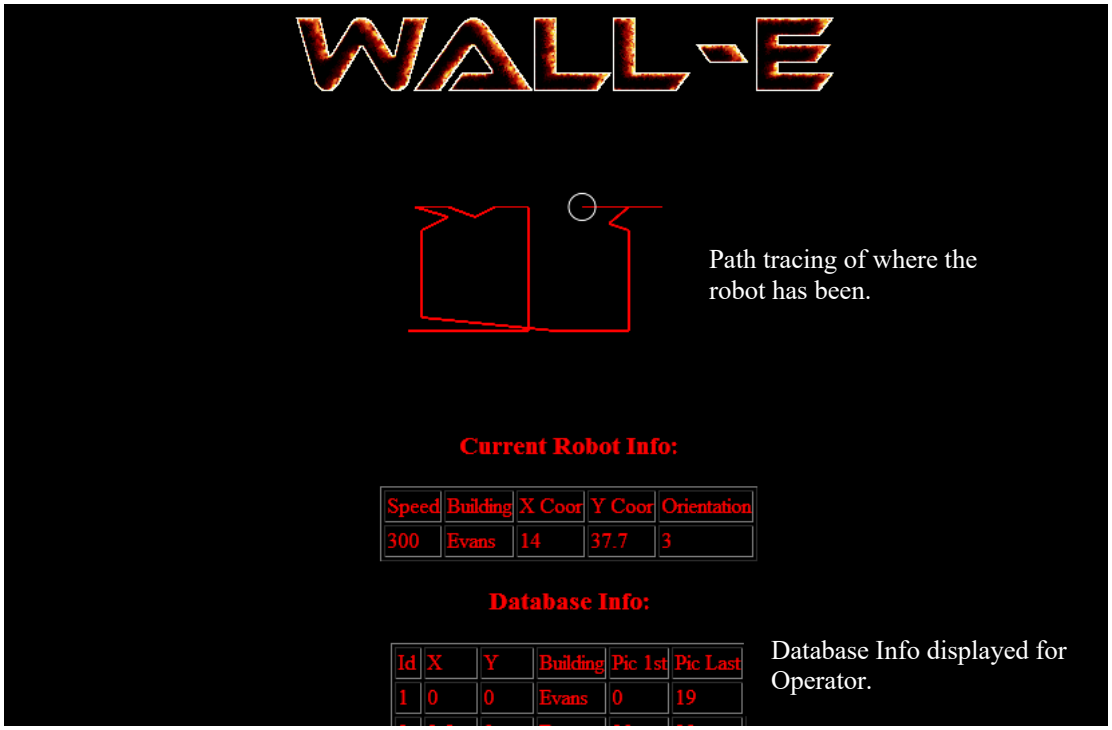
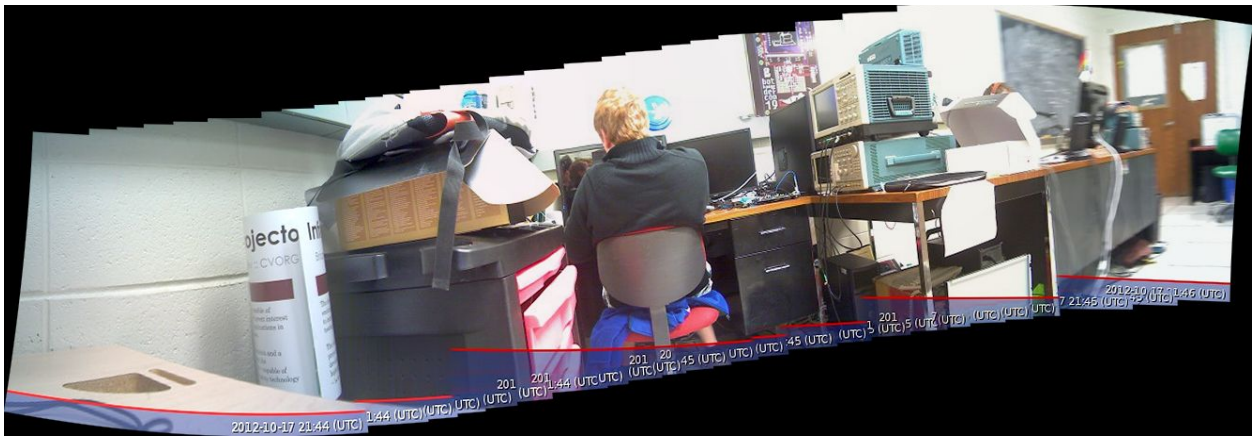


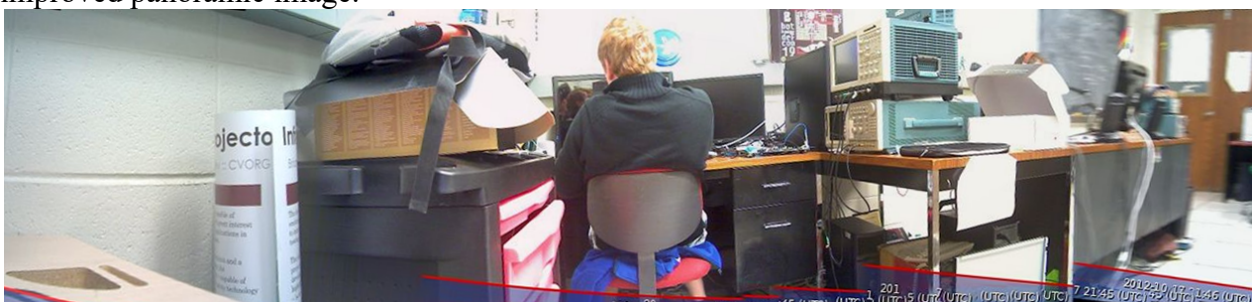
Image Stitching

Our first test was to test Microsoft's Image Composite Editor (ICE), and learn the different techniques that can be applied. We didn't have our robot or web camera operational at the time, so we used a team member's cellular phone camera to take pictures. A series of 10 pictures were taken in an Evans Hall hallway.

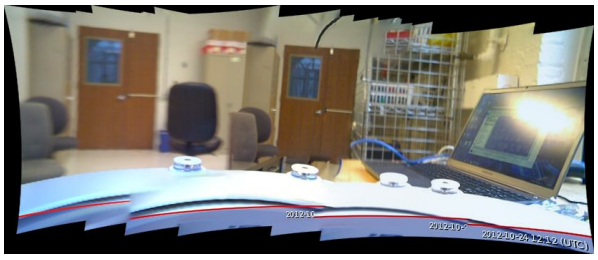
As shown above, our first test with ICE was only marginally successful. After further research, we learned that there wasn't a significant amount of overlap for the program to properly stitch the images together. Also, we were concerned with image distortion and lack of horizontal alignment between pictures. We encountered a similar distortion while testing the web camera.

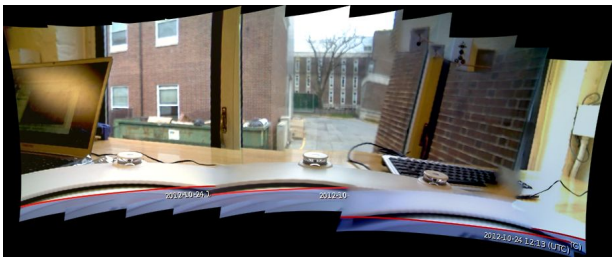


After modifying, rotating and cropping, our stitched result, we were able to get a greatly improved panoramic image.



The next step in the trial process was to have our web camera intermittently take pictures with the Raspberry Pi, while attached to our rotating iRobot Platform.



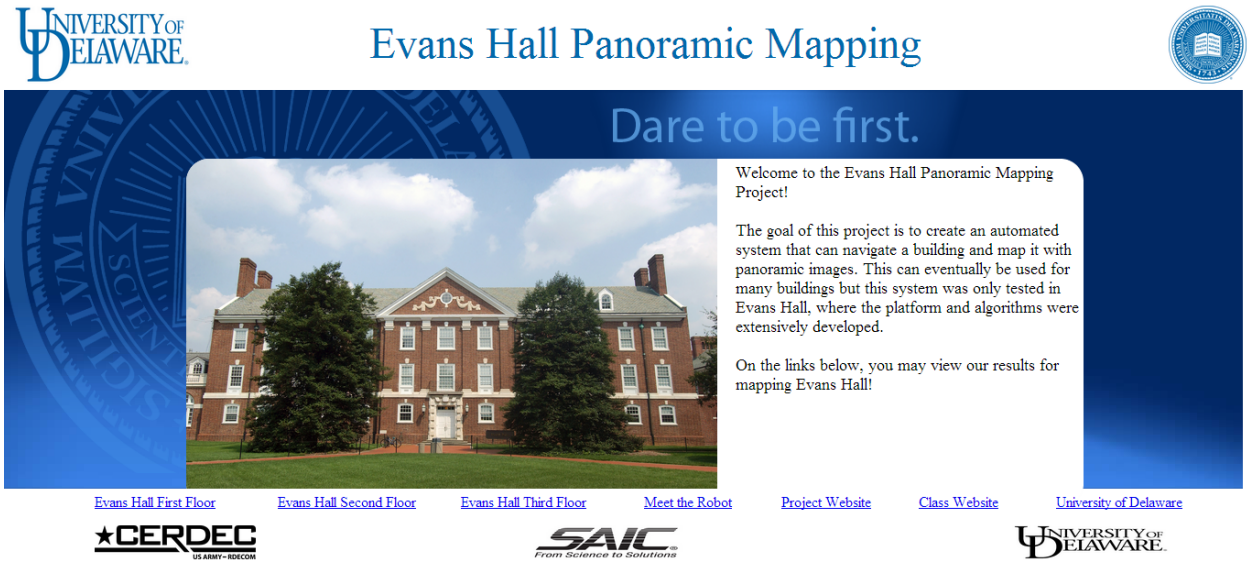


Several issues were immediately apparent with this trial-run. For one, we didn't have enough overlap for the pictures to properly stitch together. Second, the Raspberry Pi had the camera taking pictures at a slightly faster rate than our rotating robot. The timing offset caused a blurring effect along with uneven overlap between pictures. Finally, the Raspberry Pi was synced with the robot to only have pictures taken when the robot was still, this produced nice, still images. After playing around with the frequency to take pictures for the panoramas, it was determined that 18degrees was most appropriate. Taking photos at 18deg intervals allowed for nice smooth panoramas to be stitched together such as the one below.

Microsoft ICE was a great program but was Graphical User Interface based and cumbersome to use. This system worked well for testing, but a more robust and adaptable method needed to be created. To fix this, we took advantage of the database. When the pictures were copied over to a computer, the database came with it. A Python script would then parse the database and see which images were part of the same panorama. It would then create a DOS command for the command line that would call a program named Auto Stitch (which did an equivalent job to Microsoft ICE) to put the pictures together. This would then stitch all the pictures together and create a server side database with panoramic images and the locations they were taken at.

User Webpage:

The final task was to make the data collected available to the average user. It was decided to do this with a well designed webpage implementing the advantages of the database, much like the WALL-E Dashboard. On the server that stitched the images together, a web host, Apache Server, was used to broadcast a webpage. The webpage used HTML for formatting, PHP to access the database, and JavaScript for graphics. An image of the front page of the website is displayed on the next page.



From the front page, the user could access which floor they desired to look at. Each floor had its own webpage with an accurate drawing of it. On the drawing would be small buttons with a picture symbol on it. These buttons were programmatically placed to be at the same place on the floor plan relative to where panoramas were taken in Evan's Hall. Clicking on the button would produce a pop-up window with the appropriate panorama that the user could then pan to look left, right, and zoom. Further details are shown in the Results section of this report.

Standards Used:

- IEEE 802.11	-Wireless Protocol
- IEEE 802.3	-Ethernet Protocol
- RS232	-Serial Cable Protocol
- USB 2.0	-Universal Serial Bus

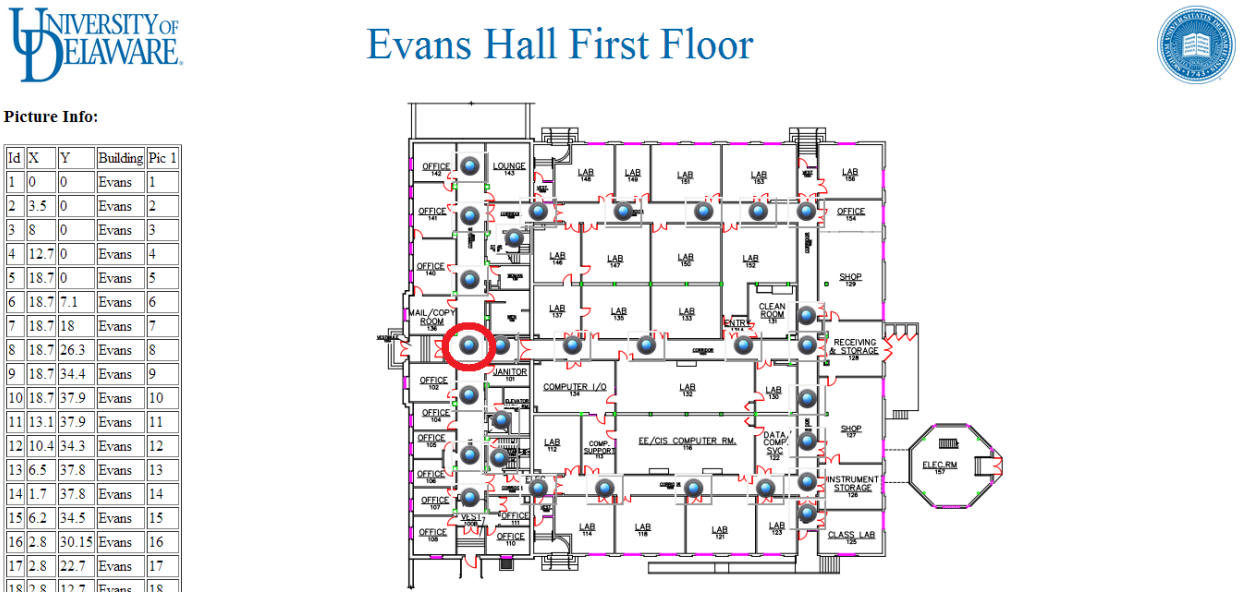
Safety Issues:

Our team faces very little danger in designing this robot platform. However, while testing the web camera with the Raspberry Pi, we use an external power supply. We chose to use a 5V, 5000mAh power supply. Our power supply has a max discharge rate of 500mA, which can cause defibrillation and paralysis. The most dangerous part of this battery is charging it. Charging it at the wrong rate could cause it to explode and cause injury. Caution is needed when handling and storing the power supply.

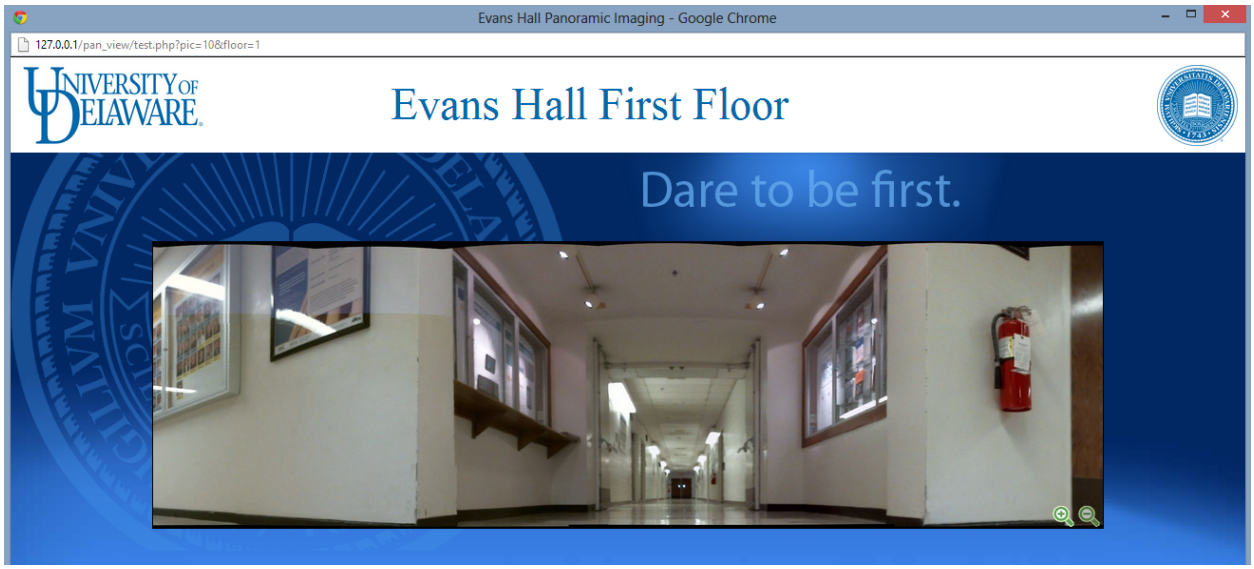
The robot also presents a tripping hazard when it is moving through the hall. Audible noises from the robot have been implemented to help warn others that it is there.

Project Results:

Our team created a full prototype of a product from backend to front end. We were able to take our creation and perform a full panoramic mapping of Evan's Hall. A fully functional website gives an enjoyable user experience when they access the data. Accessing the First Floor on the website yields:



Clicking on the picture button highlighted in red results in the following pop-up image:



The image is presented in a mask so the user can only see what they normally would when looking down a hallway. This masking helps prevent the hallway from looking distorted to the user. The entire picture is shown below for comparison.

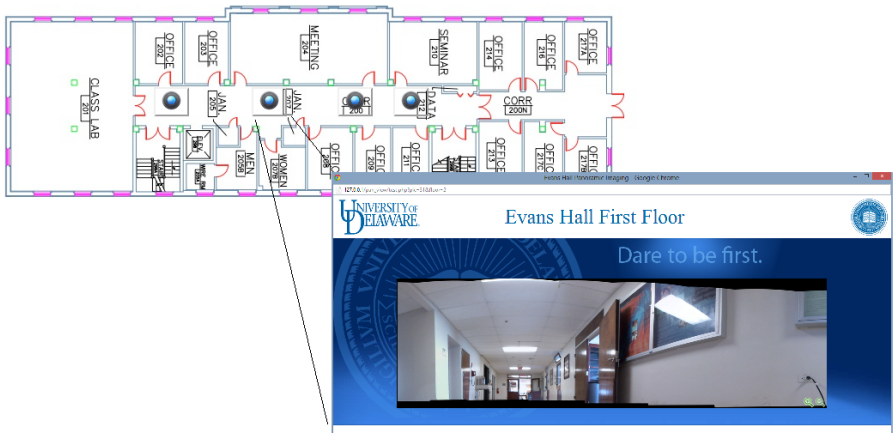
Other floors of Evan's Hall were accurately mapped as well. The Second and Third floor maps with sample location pictures are shown below.

UNIVERSITY OF DELAWARE

Evans Hall Second Floor

Picture Info:

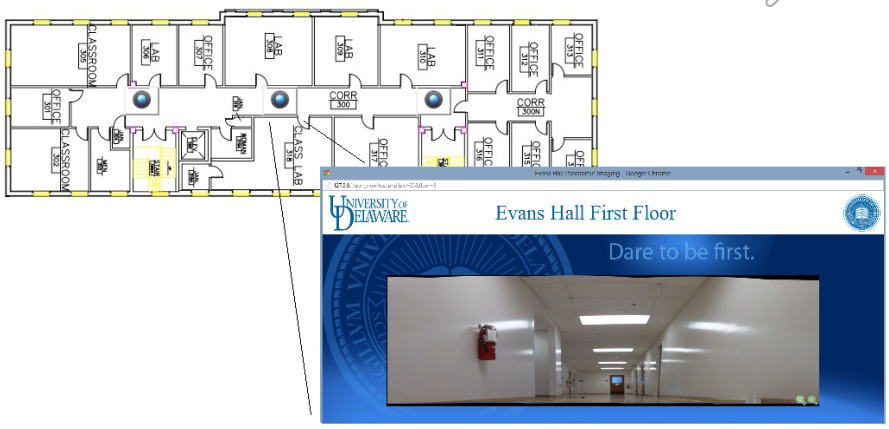
Id	X	Y	Building	Pic 1
1	1.85	0	Evans	30
2	13.5	0	Evans	31
3	23.75	0	Evans	32
4	30.5	0	Evans	33



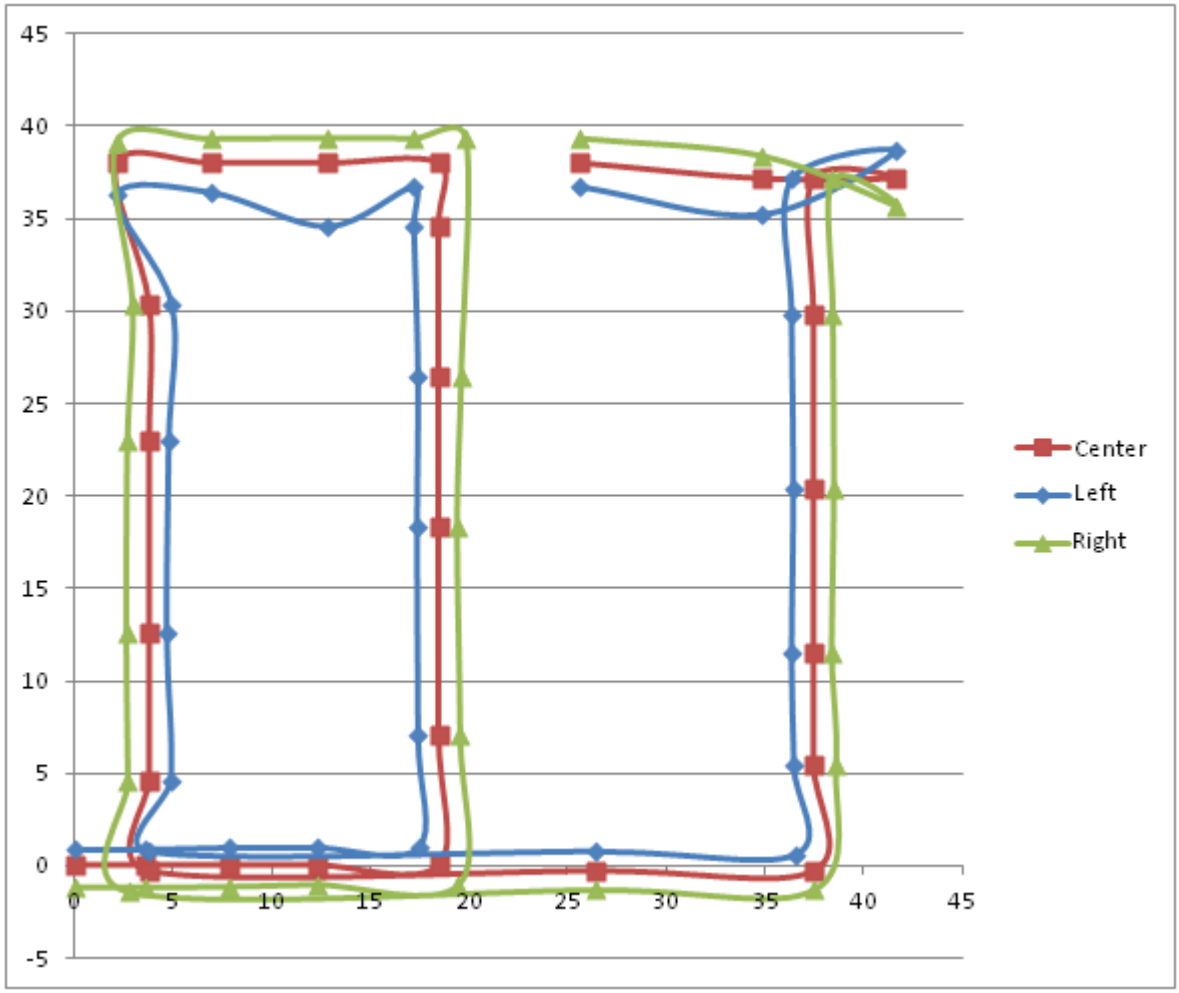
Evans Hall Third Floor

Picture Info:

Id	X	Y	Building	Pic 1
1	6.8	0	Evans	34
2	23.4	0	Evans	35
3	41.9	0	Evans	36



An accurate map of Evans Hall was also created from the data points recorded and is displayed below.



This map is measured in meters and displays the robot's process throughout the building. The red line represents the absolute path of the robot throughout the building. The blue line shows the left sonar sensor's readings of the wall, and the green shows the right. All it all it created a rather accurate map. The oddities are from specific phenomenon, such as in the top right, the robot did 180deg turn thus flipping the left and right sonar sensors. On the top left, the dip is from the sonar sensors picking up the closet where the soda machine is. One important point about this map is that no assumptions are made on turn angles.

Recommended Future Work:

As with any project, there are always things that can be improved and expanded upon. The robot AI, while functional and efficient could have been expanded upon so that it could navigate buildings such as Gore Hall. Another thing to improve would be the sonar sensors. The sensors used were cheap and, while accurate for our uses, would not be suitable in a different environment. The image process could also be tweaked to decrease waves in the panoramas and make them more natural. One process that was discussed but our team was unable to implement due to time constraints was OCR recognition. Using OCR recognition would allow the pictures to be indexed not only by location, but rather actual human symbols such as door numbers or room names.

Appendix:

Equipment Needed:

- iRobot Create Platform
- iRobot Create Battery 14.4V, 3.3Ah
- 4 5V Voltage Regulators
- Raspberry Pi
- USB-to-Serial Connector
- USB Web Camera (Logitech C615)
- USB Missile Launcher
- USB Hub
- USB Wi-Fi adapter
- 3 Sonar Sensors
- Speaker

Budget:

- iRobot Create platform $129.99
- iRobot Battery $32.99
- Logitech C615 HD Webcam ~$40.00
- Raspberry Pi ~ $39.99
- USB-to-Serial Connector ~ $14.99
- USB Hub ~ $19.99
- USB Wi-Fi ~ $19.99
- USB Missile Launcher ~ $14.99
- Voltage Regulator Circuit ~$1.99 x 4
- Sonar Sensors ~ $8.88 x 3
- Speark ~ $5.00

Total Spent Budget – $352.53

References:

Apache Web server
httpd.apache.org

AutoStitch.exe
http://www.cs.bath.ac.uk/brown/autostitch/autostitch.html

Database, HTML, PHP, and JavaScript Tutorials
http://www.w3schools.com

JavaScript code for Image manipulations -
http://www.dynamicdrive.com/dynamicindex4/imagepanner.htm

iRobot Create homepage for manuals, tutorials, and simple example code:
http://store.irobot.com/shop/index.jsp?categoryId=3311368

Microsoft Image Composite Editor
http://research.microsoft.com/en-us/um/redmond/groups/ivm/ice/

nginx web server
nginx.org