CS 1567 Final Report

Kyra Lee and Michael Appel

- ## Introduction

Our project uses computer vision and infrared sensors in order to safely navigate a user through the fifth floor of Sennott Square to room 5804, the robotics lab. By combining these two methods, we were able to design a system that guided the user through the center of the hall, identified room number panels in order to determine the user's current location, directed which path the user should take through intersections, and warned the user of nearby walls.

- ## Design

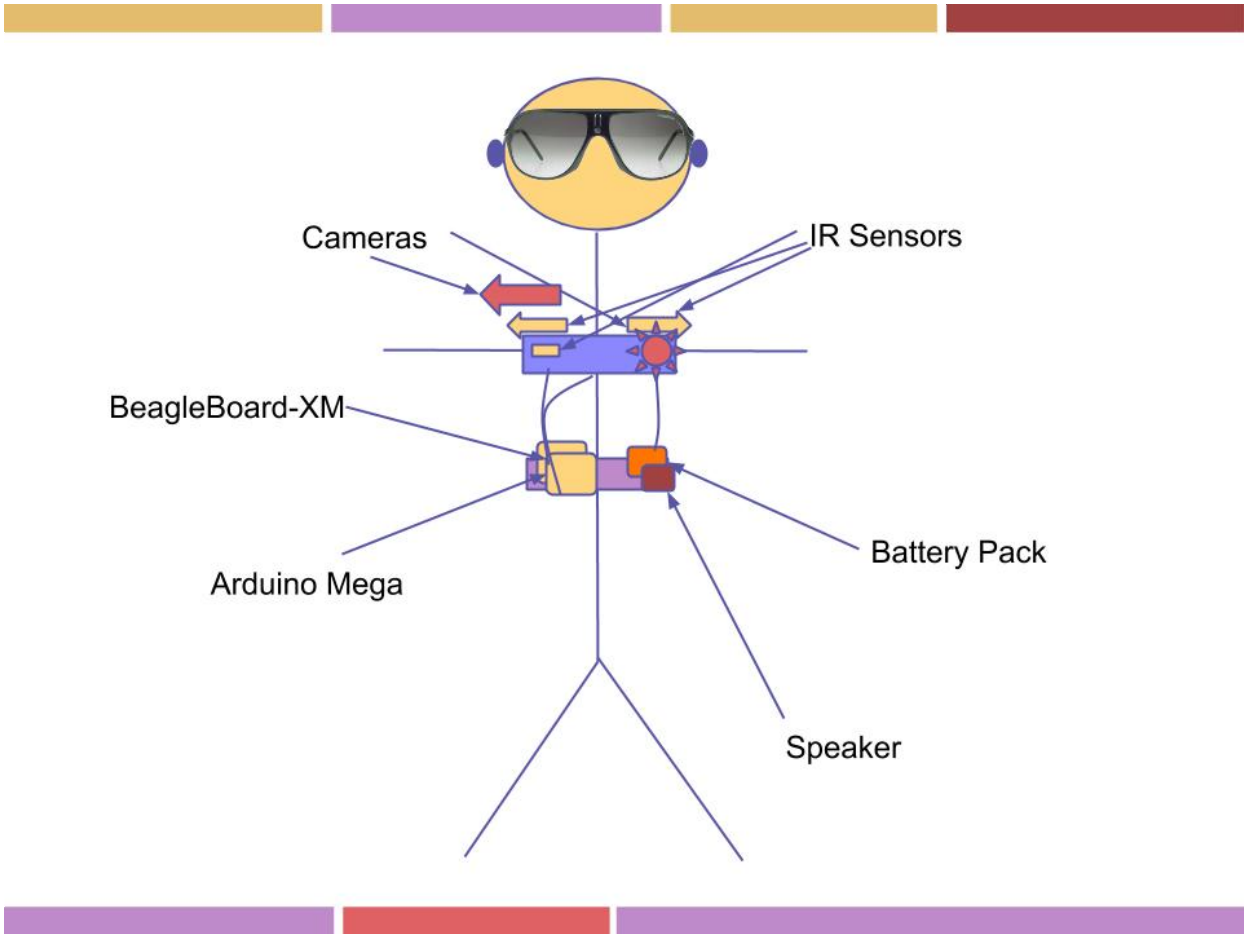


Figure 1: Artist's Rendering

- Hardware
  - BeagleBoard-xM
  - Arduino Mega
  - 3 IR Sensors (Sharp GP2Y0A21YK)
  - 2 Webcams (Logitech C210 and Kinobo Origami)
  - 10,000mAH Portable Battery
  - Portable Speaker

Our system's primary processing takes place on the BeagleBoard-xM, a single board computer (SBC) with a 1GHz ARM Cortex A8 processor and 512MB of RAM, which runs Angstrom linux. Video from the webcams comes into the BeagleBoard via USB. The analog IR sensor data is processed by the Arduino, which sends alerts to the BeagleBoard via a USB serial link. We chose the BeagleBoard-xM for its increased RAM and processing power compared to the regular BeagleBoard, and opted against the smaller Raspberry Pi SBC due to poor image processing performance in initial tests. The portable speaker is a standard 3.5mm audio out. The BeagleBoard runs on a 5V output from the battery.

The two camera implementation was a late design change. Initially the cameras had been purchased separately as two possible solutions. The Kinobo Origami is a cheaper webcam with lower resolution and less features. A beneficial side effect of this is it only supports UYVY output. Nicer higher resolution webcams, like the C210, will use MJPG compression to bypass usb bandwidth limits and output data in an RGB format. Raw UYVY data can be passed directly to the Texas Instruments DSP on-board the BeagleBoard-xM for accelerated h.264 encoding, which can be useful for recording or transmitting video streams. While it is possible to offload certain OpenCV algorithms to the the TI-DSP, the complexity of the task proved to be beyond the time constraints of this project. For future developments, such as pathfinding recording, live feeds over IP, video recording, or playback specifications, we would use the Origami.

The system harness is a backpack, with the Arduino and BeagleBoard mounted on a pegboard frame on the inside, and the sensors and cameras secured to the straps with velcro. The three IRs are facing one each to the front, left, and right. One webcam is mounted facing forward, and one is facing out from the right shoulder. The backpack's straps and the velcro mountings make it possible to customize the fit of the system for individual users.

- Software
  - OpenCV (Open source computer vision library)
  - eSpeak (Open source text to speech software)
  - Tesseract (Open source OCR software)
  - Arduino analog input and serial output

Our program operates in two main stages: first localization and then navigation. During the localization phase, the right-facing camera searches for plaques outside of rooms, based on checking for boxes defined by bounding lines. When one is identified, we send the image of it to Tesseract for processing, which returns any recognized text. We parse this output for four-digit room numbers. If we are able to identify one, that will give us our current location. Once we have localized to outside a particular door on the map, we move on to navigation. The map defines a route back to the robotics lab, a series of directions to take at encountered intersections, for each plaque. We then direct the user through this route, checking off one intersection at a time, until the user successfully arrives at the goal.

Throughout both stages, the forward facing camera searches for lines defining the perspective down the hallway. By tracing these to a goal point and that comparing that point to the user's current trajectory towards the center of the frame, we are able to direct the user to "bear left" or "bear right" in order to continue walking forward in the center of the hallway. Additionally, we can query the Arduino in order to obtain any current IR sensor warnings, and warn the user to stop and step away from the wall if they come within a dangerous range.

- Implementation
  - OpenCV Perspective Lines

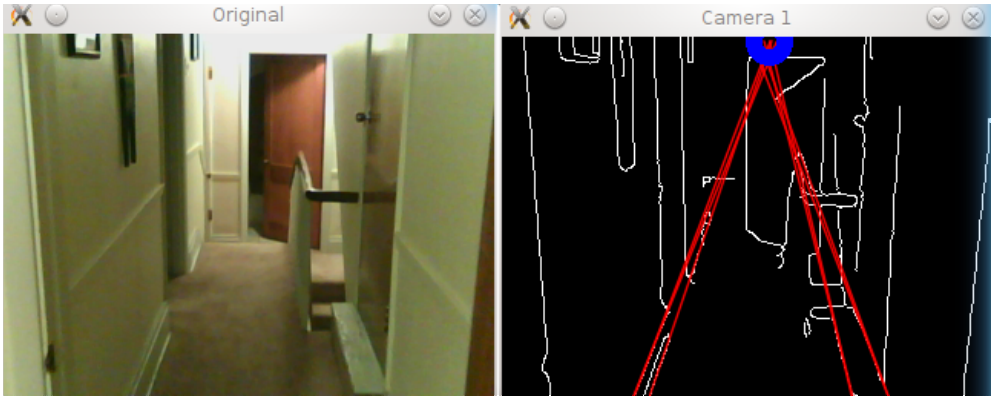
Our initial design specifications called for tracking the overhead lights to maintain centering through the hallway, and to recognize the round lights and fire exit signs at the intersections to aid in navigation. However, there were difficulties between our preferred webcam (a Sony Eye, chosen for its wide field-of-view) and Angstrom's older UVC kernel module. As a result the camera was not able to adjust exposure, and the interior of the building was extremely washed out. The lights were indistinguishable, and line detection suffered greatly from the lack of contrast and definition. Because compiling a new kernel and drivers would have been an excessively time-consuming exercise, we moved to the dual webcams, which worked well under the older drivers. However, because of the narrower field of view of these cameras we were unable to see lights closer than 15 feet away, which would make it infeasible to navigate by them.

After researching solutions used in other computer vision navigation research, we implemented the vanishing point orientation technique.

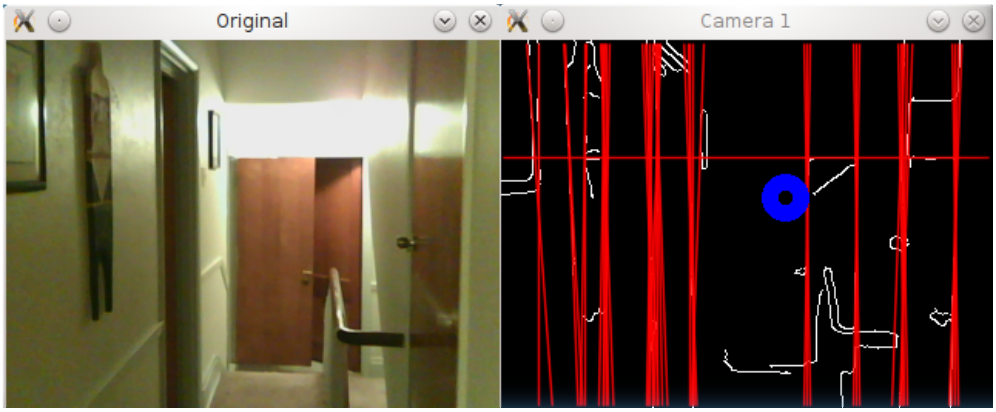
The image processing begins by capturing a frame from the camera. We convert the image to grayscale to reduce the image from three channels to one. The normal approach is to then apply a blur to reduce noise, and improve line detection by making pixel value changes more uniform. However, in our testing this reduced the straightness of the lines, and consequently decreased the effectiveness of the Hough transform. Using a pyramid style resizing, we downsize the image by half, and then grow it back to original size. This gives acceptable noise reduction, and maintains the sharpness of the lines in our system. Next, we apply the Canny function to the image which calculates the variations from pixel to pixel looking for sharp variations in one direction, and continuity in the other. This is the hallmark of a line. Canny will then return a binary image to us, which consists of the white lines detected on a black background. The binary image allows us to run the Hough transform, which gives us a set of polar coordinate described infinite lines. By separating these lines into their respective slope categories (negative, positive, horizontal, and vertical) we were able to create a more robust perspective detection algorithm to false positives. We initially check negative and positive lines against each other; if that fails we test the set against itself. Ideally, we find lines on the left and right of the hallway, and extrapolate their opposite slope values to an intersection. If there is an obstacle blocking the line on one wall, then it will not be found. If that happens, we then check the negative and positive slopes against themselves. However, this second approach is more prone to false positives due to bulletin boards, mailboxes, lights, carpet noise, etc.

Once we do have an intersection point, we compare it to the center of the frame, and orient the user towards it. If it is already in our centering range we issue a “Step forward” command. This solution is excellent because it is both general and robust. This algorithm would work anywhere with relatively good lighting (or contrast), and could be adapted to handle curves by examining the derivatives of a set of similar lines.

Our second camera uses the same vanishing point technique in order to identify intersections during the navigation phase. With a more complex, specifically tested algorithm, it might be possible to find intersections and doors with the vertical and horizontal slopes of a single, forward-facing webcam’s image. However, the dedicated camera simplified the algorithm, and allowed us to reuse the same algorithm as for hallway centering.



A rather noisy example image. Demonstrates the robustness, and generalized nature of our solution



The separated vertical and horizontal lines found

- Sign recognition and Tesseract results

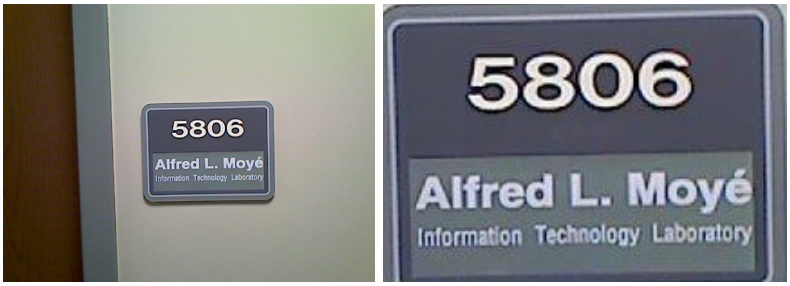
During the localization phase, when the user's location is not known, the shoulder mounted camera is constantly scanning for room signs on the walls.

Because the walls are relatively plain and lightly colored, while the room signs are dark, we are able to use a more efficient version of the Hough Transformation to find the lines in the image. First we convert to grayscale (single channel), then we apply a Gaussian blur to reduce noise (acceptable result here, and faster than resizing twice), and pass the image to the Canny function to find lines before running the Hough transform. The probabilistic Hough Transform will not locate infinite lines, but finite units which will help us construct our boxes (or regions of interest). A Gaussian distribution of points is used to check for edges in the binary image. It will create lines once a threshold value of points is reached that are connected, and it will then return the line. This is more efficient than naïvely checking every value in the frame. Once the lines are known we calculate the intersection of each line with the others, and then create a bounding box

based on the minimum point (upper left) and maximum point (bottom right) that were found. Once the box is found, a pointer to the region of interest that matches the box dimensions is created and this is written to disk. We then execute Tesseract OCR on the output image, read back its results, and parse out the discovered room number.

The sequence below shows the raw image, the binary Canny result with probabilistic hough lines overlaid, and finally the corners highlighted with blue circles found as intersections with the bounding box describing the region of interest. The final image is an example of the cropping done to improve Tesseract's accuracy.

Sequence showing steps to find rectangles in an image



Tesseract reads this sign as:
5806 W
Alfred L. Moye
InInrmlliflll Tldmdflly Llbulllnly

- Arduino Serial Communication

The Arduino continually monitors the analog voltages it reads from the IR sensors, keeping track of the current and previous measurements for each of them. At appropriate stages, and at least once in every iteration of the main system processing loop, the program queries the Arduino, which responds with any current warnings. Warnings occur when both the current and previous measurements of one sensor are above a threshold voltage of approximately 1.5V, which, for our sensors, corresponds to a distance of approximately 18 inches from the wall. When such a warning, if any exists, is received by the program, it directs the user to take appropriate evasive action.

The communication takes place over USB as a serial port. On the Arduino's side, all communication takes place using the built-in Serial.read() and Serial.print() commands. On the BeagleBoard, the Arduino mounts to the Linux device file /dev/ttyACM0. To use it from our program, we use open(), then initialize the file descriptor to take into account the peculiarities of the serial format, and then can use the standard C read() and write() functions for all future communications. The details involved in initializing the serial communication, including opening the file in non-blocking mode and reading the data in raw form, are listed in our source code and at our Arduino Serial Communication reference. A query request to the Arduino simply consists of any single byte, and the warning response byte is one of the characters 'F', 'R', or 'L', to represent whether the warning is from the front, right, or left IR, respectively.

- eSpeak User Interface (including list of commands)

Using the open source text-to-speech software eSpeak allowed us to generate as many user commands as needed as easily as typing in a string. This made it possible for us to have several different types of audio output, tailored to all of our system needs.

- Hallway Centering and Guidance
  - "Step forward"
  - "Bear right"
  - "Bear left"
- Wall and Obstacle Avoidance
  - "Slow down"
  - "STOP"
  - "Wall ahead"
  - "Wall on right"
  - "Wall on left"
- Specific System Guidance
  - Full guidance for turning around 180 degrees, pausing to locate a room sign, and stepping down the final hallway to room 5804.
  - System status updates for loading directions file, initializing the camera, and any errors in camera inputs, Arduino connections, or Tesseract file errors.

Results

As a proof of concept, our system was a rousing success. Our software successfully navigated back to the room running on a laptop, with only a small hiccup in noticing the last intersection. The final run with the complete, integrated system was only stymied by easily fixable bugs in our source code. Our approach was sound, and required only slightly more testing and debugging work in order to result in a complete computer vision navigation solution.

In future work, we will begin testing much earlier, in order to address all bugs and glitches sooner in the development process and leave more time for system testing. Our experience with Tesseract is an example of insufficient system testing. We downloaded Tesseract version 3.02 onto the BeagleBoard very soon after adding OCR to our solution. However, when it failed to perform during final testing, we were forced to rollback to version 3.00 at the last minute. When testing on the BeagleBoard itself, we also encountered known bugs with USB crashes and out of date kernel modules and drivers. We each developed and tested individually, emulating parts of the system on our own laptops, but in the future we will be sure to begin integrated testing on the actual system hardware as soon as possible. This will ensure that we are able to present bug-free solutions at the conclusion of our development.

- # Conclusion

Computer vision is an amazing solution to our navigation problem. With video processing approaching 5 fps, we navigated down hallways faster than individual step-based pedometer or accelerometer solutions. We were also able to gather more information (such as room number and name) about our environment than would be possible using only IR or ultrasonic distance-sensing devices. Even with our high-performance SBC, the BeagleBoard-xM, our inexpensive webcams kept our budget very reasonable. Further refinement of the design could even reduce our design to a powerful smartphone app, and could easily expand into new maps for different applications.

The value of the design we demonstrated is impressive. The reduced costs of image sensors and falling costs of computing hardware indicate video based real-time processing guidance is still only in its infancy. Though our system may take some more time and development to become more useful to a blind person than their own adapted senses, it will eventually prove an indispensable step forward for both human and robotic navigation.

In terms of personal development, this project was also a great success. We had great opportunities to learn about computer vision, the C++ language, hardware and Arduino programming, and team software development. The experience was very rewarding, and we look forward to applying the skills we have acquired in the future.

- References
  - Arduino Serial Communication: http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino/
  - eSpeak: http://espeak.sourceforge.net/
  - OpenCV: http://docs.opencv.org/
  - Tesseract: https://code.google.com/p/tesseract-ocr/
  - EyeDog research paper: http://heracleia.uta.edu/projects/cplay/paper/eyedog.pdf
  - Computer Vision Capabilities for Semi-Autonomous Wheelchair: http://www.ai.uga.edu/Theses/tarver_jeremy_l_200808_ms.pdf