Implementing Multi-Floor Intelligence within a Moving Elevator

Arnav Jain, Ashay Lokhande, Brahma Pavse

I. Abstract

The Building Wide Intelligence's primary mission is to design fully autonomous robots that can become a permanent part of the environment in the Gates-Dell Complex. In the process of becoming fully autonomous, these robots have to be able to navigate the interior environment of the main building. Currently, the robots are able to navigate a pre-configured map environment which is configured using a separate method. Ultimately, the goal is to have autonomous robots which can navigate variable environments (environments in which objects don't have a fixed location, but they move around dynamically as individuals interact with the environment).

A part of this mission is to implement functionality which would allow the robot to move in between floors. Currently, the robots have very little functionality related to interacting with the elevators. In this research paper, we explore an important components of this key functionality when interacting with the elevator - how the robot is to determine what specific floor it is on.

Using a secondary USB camera, we have programmed a Robot Operating System node which can take a clear video feed (presumably in an empty elevator) of the elevator button panel and detect which specific buttons are lit. This detection is specific to circular buttons with floor indicator lights in the center of the button. First, we detect the specific buttons and the numbers placed next to them. We use this detection of buttons to further work on enabling the node to detect the circles in the center and then map a specific button to the relevant floor number in real time.

II. Introduction

A prevalent issue that exists in the Building Wide Intelligence lab is the ability for a robot or an "intelligent" autonomous system to navigate itself inside a building via an elevator, specifically in the Gates-Dell Complex. As described in the abstract we aimed at trying to resolving this issue.

The current state of the robots in the Building Wide Intelligence lab is that they are able to maneuver themselves on the second and third floor, given that the map for that floor exists and is understandable by the robot. As of now, the robot must be manually placed in an environment that is "understands." A problem that gets rectified after working towards our goal in this project is that the robot will be able to navigate itself

from floor to floor via the elevator. When the robot is placed in the elevator, like a normal human, it faces towards to the buttons and based upon the lighting of the buttons it identifies where it currently is. When the elevator settles on a floor, the last recorded floor is the final floor on which the robot is on. Given that the map for the required floor has already been constructed the robot can pull up the map for that floor and successfully navigate itself on the floor.

The problem we will solve is the development of an automatic robotic system designed to identify which floor the robot is on and, hopefully, navigate itself on the detected floor with ease.

III. Background

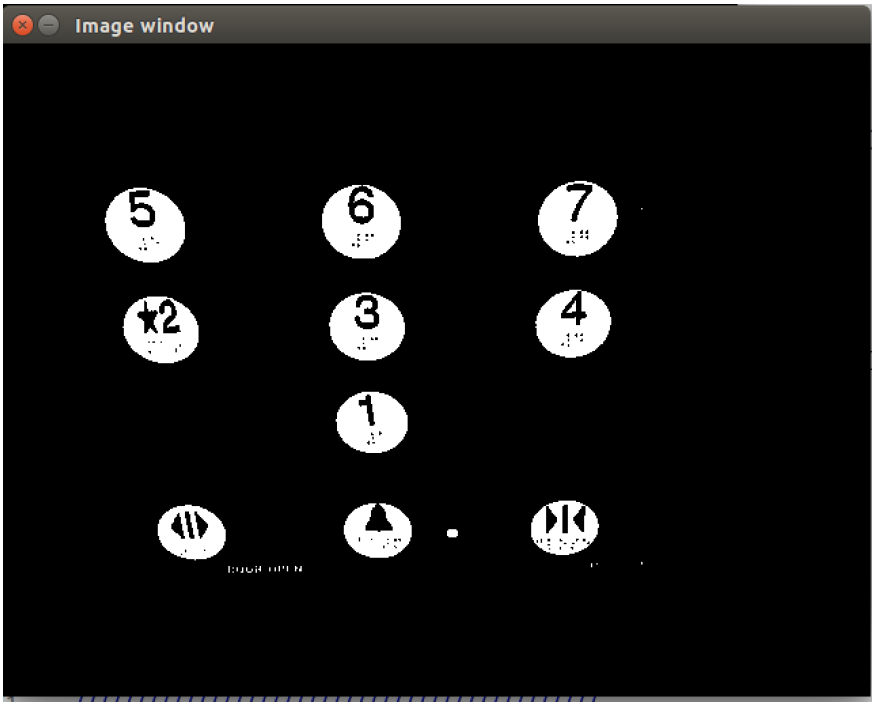
The development of this project was initiated after considering the several challenges that currently exist with the Building Wide Intelligence lab robots. The motivation to push towards this goal was also driven by the fact that this issue is prominent among other institutions as well. For example, among the numerous research papers that we studied and critiqued, a paper from Stanford University described a similar issue which we tackle in our project. The details about how we pursued this problem are discussed further in this paper.

Although the technical details are discussed in detail below, an overall summarization of the tools we used to accomplish this goal have been the implementation of the Robotics Operating System, a framework that enables use to program in C++ and effectively communicate with the hardware of the current robot that is under observation. In terms of hardware, apart from the physical robot specimen we made use of a USB camera, which was used to as our viewfinder to capture the required video, called the rosbags, and data points in order perform the project in experiments, and real time. There are of course more details to the hardware of the robot used, but that those hardware specs are generic and contained no specialization for this project. Further, to accomplish the goal we made significant use of the OpenCV library such as HoughCircles, which enabled us to identify relevant circles in a given single frame of the rosbag.

One of the difficulties we encountered early in this lab was how we planned to obtain the live video feed of the elevator button panel. The Kinect sensor was placed too low for us to get an accurate depiction of the button’s form and the status of their lights. Thus, we utilized an external 720p Logitech USB camera as our viewfinder. Since this camera did not already have a node, we implemented a secondary node (called usb_cam) from the ROS Wiki, created by Benjamin Pitzer. In this node, the author integrated the Logitech camera drivers to publish the video feed to a common topic which we could then subscribe to similar to the Kinect's video feed topic.

IV. Technical Approach

After developing the required rosbags from the video stream. We use this as our data to study in order to accomplish the required goal. We perform this by converting frames of the given rosbag into a gray scale composite image. After we perform this task, we make a clone of this grayscale image in order to perform a thresholding method on it. We threshold the grayscale image to identify the black circles; these circles contain the floor number that the associated button represents. When we are able to successfully identify these black buttons we use this information to get the location of the buttons and map which buttons represent a given floor number.



After we perform the steps above, we blur the image in order to remove any unnecessary noise and interference, which might conflict with our result sampling. The Gaussian Blur assists our process tremendously because it maintains only the relevant information and data on the frame for the given rosbag. After we remove the noise, we use Hough-Circle transformation. The purpose of this transformation is to successfully locate the circles to the left of the actual buttons. The circles that we add to the vector is that circle that is within 3 radii of the x and y coordinates of the circle already present in the vector. These circles to the left of the actual buttons are the black circles that contain the floor number within them. These black circles next to the actual buttons are sequentially added to a vector. Each object in the vector consists of the three components - the radius of the circle collected, and its x and y coordinates of the center. Like this, we repeat this process until we collect 10 elements in the vector. In other words, we keep populating the vector until we get the 10 circles in the vector; we have a hard number of 10 because we know that there are only 10 buttons that are under observation and relevance to accomplish the required goal.

Once the vector of circles of size 10 has been acquired, we loop through the vector iteratively to find which circle contains the maximum radius; we try to find the maximum radius in order to normalize the data and reduce errors. Once we acquire the maximum radius, we apply this maximum radius to all the circles in the vector in order to, as mentioned before, normalize the data.

After normalizing the data, we have data to work with it because it has been standardized. We then sort the vector of circles based on the y coordinate of the center of the circle objects. When

we do this, we allow ourselves to identify which index in the vector corresponds to which button. We know that the last three buttons are meaningless to identify the floor numbers, so we sort the vector to bring them together in order to ignore them altogether. After we get the sorted circles, based on the y coordinate, we sort the vector as groups of three based on their x axis. The reason we consider a group of three is because the elevator panel consists of three columns of buttons. Once the circles have been sorted in all the required manners, we know which index in the vector corresponds to which button on the panel. To identify floor numbers we do not need the last three buttons; rather than ignoring them entirely, we erase them from the vector in order to have the vector contain relevant buttons.

At this point in the process, we have filtered the vector to contain only relevant circles. We loop through the vector and find buttons that have a y coordinate within 2 radii of where the last deleted button was. After applying this further filtering, we reach the stage where we can map an index of buttons to the floor numbers:

Index	Button Floor Number
0	5
1	6
2	7
3	2
4	3
5	4
6	1

Using these relevant indices, we can calculate the amount of white pixels (the pixels that pertain light from the buttons). If this amount crosses a certain established threshold, we can decide if a button is lit or not. Based on lit or unlit we draw the appropriate circle, to track the lit button.

Overall, every fifth frame we count the total white pixels again and compare to the average count through the array. If there is a significant change in difference of the amount of white pixels calculated we know that the light has been turned off; in other words, the button is unlit. Through this procedure, we can identify if a button is lit or unlit.

V. Evaluation

We ran our tests with two variables to test upon. Our method of testing involved collecting rosbags for the information published by the external USB camera. The rosbags we collected recorded data from the /usb_camera/image_raw topic. In our collection of the rosbags, we placed the robot at three different distances from the elevator button panel.

The first placement of the robot was at 0.5 feet from the elevator panel. The second one was at 1.5 feet from the panel. The third one was 1.0 foot from the panel. Our purpose for seeking three different placements was to see if the distance from the elevator panel distorted our detection algorithm. In doing so, we were able to first test our basic code on the farther detection algorithm and then increase the accuracy of the node to detect the light change from different distances. Furthermore, we recorded the rosbags with

sequentially pressed buttons in different arrays to simulate the random nature of buttons on a typical trip up or down the floors in the GDC.

One of the biggest variables we anticipated in our testing was the reflection of natural light in the elevator. We tried to control for this variable, however, we found it extremely

difficult to narrow the natural light reflection since the elevator is made of metal (reflection) and the robot's main hours of operation are during the day (natural light). Hence, we took our rosbags with the reflection in order to mirror real usage conditions.

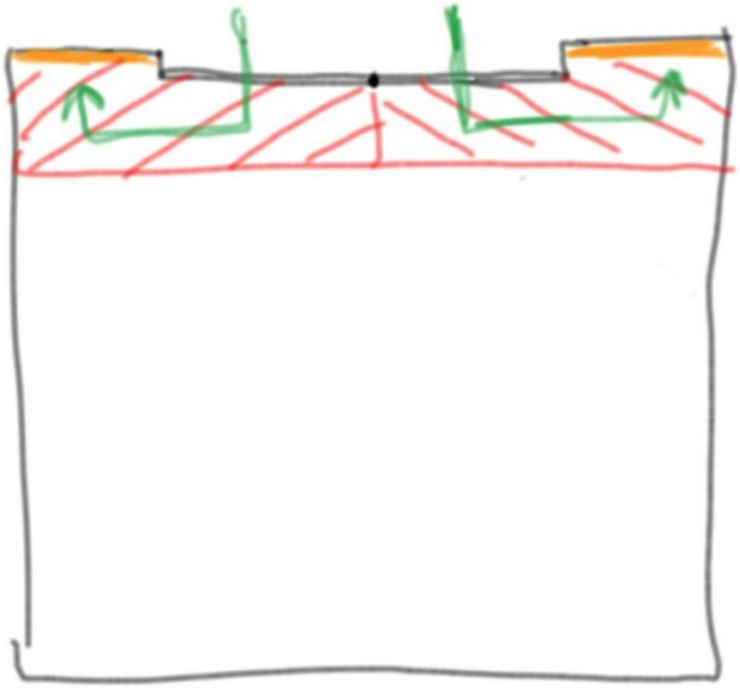
Another primary challenge that existed, specifically to the Gates-Dell Complex, is making the distinction between the buttons in the elevator. The construction of the button panel in the Gates-Dell Complex elevators is made in a manner such that the button circumference is extremely light, which makes it hard to distinctly detect a button. This challenge escalated further when we incorporated the Gaussian Blur; we used this feature in order to avoid unnecessary noise and interference. However, in addition to avoiding noise the blur faded out the already light circumference of the buttons, which added onto current challenge that we were presented.

After taking our rosbags, we ran our node side-by-side with the rosbags. We did this multiple times with the rosbags we collected at 0.5 feet. We then recorded the accuracy of the robot based on if it detected the light correctly (which we can observe is lit or not).

VI. Conclusion

Overall, we learned a good deal of handling a task with such a high variability of success given our current knowledge and skill level. At the end of this project, we can place a mobile robot in the elevator facing the required button panels and navigate the Gates-Dell Complex building elevators, and hopefully in unknown elevators. From there, we can use the lighting of the buttons to identify which floor we currently on. The technicality of this information is described above in the paper. After performing several tests, we have a fairly decent success rate for finding the buttons and detecting the lights. We found through our testing process, we could detect the buttons with an average 96% success rate and detect the lights with an average 56% success rate with most errors originating from extra detections due to the reflecting light.

The reason we have such a low success rate for the light detection comes back to our previous problem, discussed in our evaluation, of light reflection within the elevator. The natural light, when reflecting towards our camera, appeared similar to the lights that turn on when an elevator is pressed, thus fooling our algorithm. One of the biggest challenges we foresaw entering this project was that this interference could make our node less optimal. Even though we have optimized this node for real usage scenarios, we were able to make substantial development towards the goal and we hope to reach a higher success rate in the future to account for the reflecting natural light.



For further research, we look to the human-elevator interaction paradigm. Your typical human presses the button of the direction he/she wishes to go, detects which elevator opens, enters the correct elevator, locates the elevator button panel, presses the button he/she wants to get off at, and then detects the floors changing as the elevator moves up or down. In this paradigm, we have created a node which can detect the elevator

button panel and detect which floor the elevator is currently on. To improve upon the robot's role in the above paradigm, we could program nodes that can navigate the robot inside the door once it has detected the specific elevator door that has opened (there are two elevators). Once it enters the elevator, it could navigate to the correct position in front of the elevator button panel, as indicated by the figure above. Similarly, we can program the robot to exit the elevator once it has detected the elevator has reached its floor.

VII. References

"Feature Detection." *Feature Detection — OpenCV 2.4.13.0 Documentation*. Opencv. <http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles>.

J.-G. Kang, S.-Y. An, and S.-Y. Oh, "Navigation strategy for the service robot in the elevator environment," in *International Conference on Control and Automation Systems*, 2007, pp. 1092–1097.

Klingbeil, Ellen, Blake Carpenter, Olga Russakovsky, and Andrew Y. Ng. "Autonomous Operation of Novel Elevators for Robot Navigation." <http://ai.stanford.edu/~olga/papers/icra10-OperationOfNovelElevators.pdf>.

Pitzer, Benjamin. "Usb_cam." *Usb_cam*. ROSWiki. <http://wiki.ros.org/usb_cam>.

S. K. Divvala, D. Hoiem, J. H. Hays, A. A. Efros, and M. Hebert, "An empirical study of context in object detection," in *Computer Vision and Pattern Recognition (CVPR)*, 2009.