Indoor Navigation for the Blind: Processing Digital Floor Plans Using OpenCV-Python to Determine Bluetooth® Beacon Placement*

Charlie Broadbent

***Abstract*— As the field of indoor navigation is further developed, an optimized method of implementing and deploying the system must be developed if it is to see widespread use. One of the necessary steps in this process is determining the placement of navigational beacons throughout a building and its floors. To do this in a timely, accurate, and efficient manner is a difficult task. In this paper, we explore the effectiveness and limitations of one possible implementation of OpenCV and its library of computer vision programming functions in a software tool that automates the beacon placement decision process. We find that it is possible to create a tool that is almost effective as pre-existing tools, though several improvements to accuracy, run-time, and usability must be made before it should see any practical use.**

I. Introduction

Indoor navigation with software such as GuideBeacon [1] has been shown to be useful in helping the visually impaired find their way throughout a building and its floors. Implementing this system is a multi-faceted procedure that requires efficiency, accuracy, and coordination between its working parts from start to finish. One of the steps in this implementation is determining the placement of Bluetooth beacons, which, when within a certain proximity, can transmit information to users their location, directions, and any other important information.

A useful place to put these beacons is at every door. One simple way to locate every door on a floor would be to do it manually—someone can look over an image of a floor plan and find the doors themselves, as well as find any other locations on the floor where a beacon may be helpful. This, however, is prone to human-error, and can be a painstaking process, especially for very large floor plans.

Fortunately, there exists ways to automate this process by using computer vision and machine learning. IBeaconMap [2] is a software tool that implements these. Given a digital floor plan image, it is able to locate not just the doors, but also find the path a user is able to take throughout the floor. This tool is written in MATLAB, however, which means it cannot be freely distributed for others to use. Luckily, other options for implementing computer vision exist, one of them being OpenCV.

II. Problem

OpenCV [3] is a library of programming functions used for implementing computer vision, and thus it should provide the necessary tools to develop a program similar to IBeaconMap. At the most basic level, we want a software tool using OpenCV-Python that can locate doors on a floor plan to a similar degree of accuracy to IBeaconMap. We find that developing such a tool is possible, though several improvements must be made before it can be a satisfactory alternative. The main goals were to have a tool that:

1. Marks as many doors on a floor plan as possible without missing any.
   i. At the same time, it should make as many *least* redundant marks as possible (i.e. marks that are not doors).
2. Is simple to use for a person of any level of computer experience.

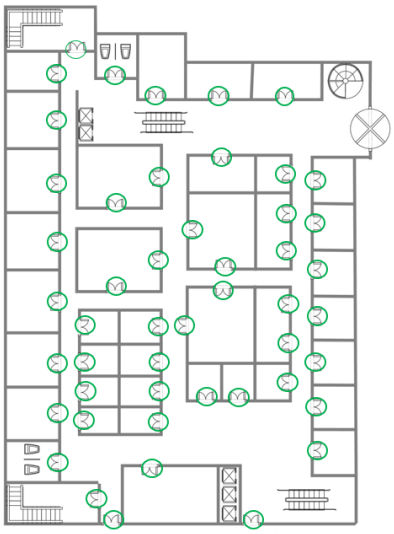


Figure 1. Example of a floor plan with its doors marked without redundancies. This would be the most ideal result.

III. Architecture

The tool takes two inputs: an image of a floor plan, and a cropped image of a door from that same image. It then takes those images, processes them using OpenCV functions, then returns an image with all the doors marked.

A. Feature Detection

The first step in identifying doors is through feature detection and matching. There are various algorithms available in OpenCV that allow us to take images and determine their features. If we run a feature detection algorithm on, for example, just an image of a door from a floor plan, we get back a set of keypoints the algorithm decided are important to the image, as well as descriptors for each keypoint.

*This work was supported by the National Science Foundation

Given a floor plan, we can run the algorithm on a cropped image of the most basic door to appear in the plan, as well as the floor itself. Using the keypoint descriptors, we then use another algorithm to match the features we found in the floor plan to the door's features, and remove everything we know with high likelihood is not a door. This tool uses the SIFT (scale-invariant feature transform) detection algorithm [4], as it provides the best results with the least amount of parameter tweaking needed. The keypoints for a sample floor plan and door can be seen in Figures 2 and 4.

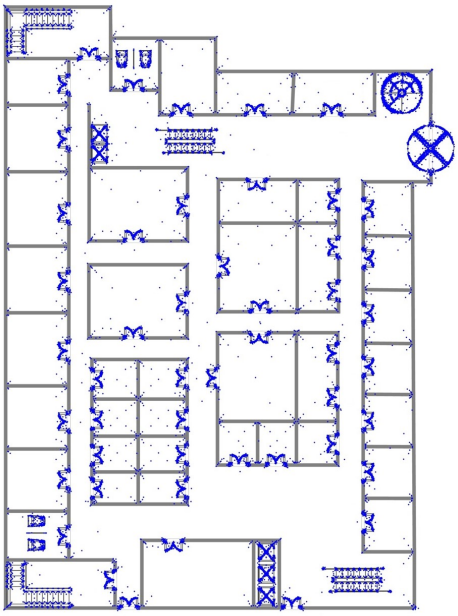
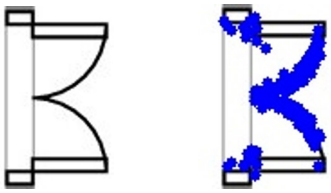


Figure 2. Keypoints of an example floor plan.



Figures 3 and 4. A cropped door image from the floor plan and its keypoints.

One factor that needed to be accounted for was the resolution of the floor plan. In general, the less high-def an image is, the lower number of keypoints that will be found, and with less descriptive descriptors. This creates many undesired matches between the floor and the door. This is slightly fixed by fixing the sigma parameter of the SIFT algorithm function to *sigma=0.5*, versus the default of *sigma=1.6*. By doing this, we increase the number of features detected in low resolution images, and also gives us less pronounced, thinner features, which in turn provides us with more accurate matches. This does have a drawback of also increasing the number of features across all images, sometimes more than is needed, which significantly increases the run time. This becomes a serious problem in high resolution images with long curves present in the floor plan that are not doors, as the feature matcher will interpret them to be doors which results in many redundant points, thus in creasing the run-time.

Figures 4 and 5. Difference in number of keypoints detected with *sigma=.5* (Figure 4) versus *sigma=1.6* (Figure 5).

B. Feature Matching

Once the floor plan's and the door's keypoints and descriptors have been found, a brute-force matcher is used to match the floor plan's features to the door's, leaving only points which are most likely part of a door, as seen in Figure 6.

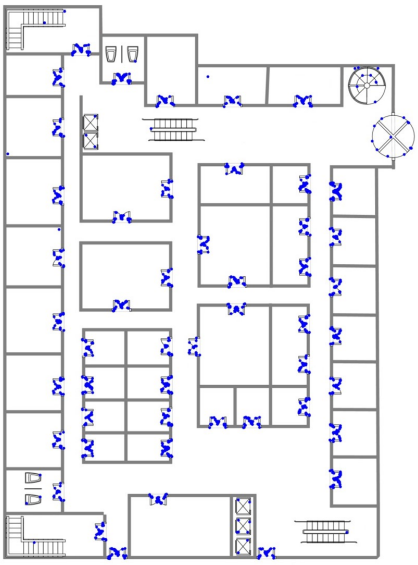


Figure 6. The matched features from the floor plan.

Note that feature matching algorithms will find and return matches that are not doors, leaving us with more keypoints than what is ideal. See Figure 7 for an example of a floor plan with long curved features that are matched as doors. Regardless of the number of redundant points that are remaining after feature matching, we then have to narrow the set of points down such that each door only has one point on top of it, or in its near vicinity.

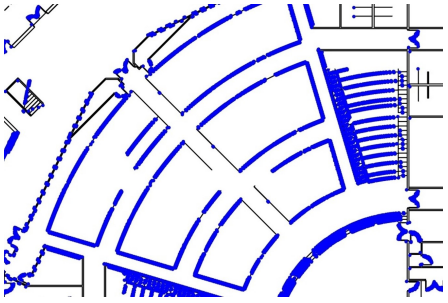


Figure 7. A floor plan with curves that are detected as doors.

C. Clustering

We accomplish this narrowing by first using k-means clustering. At each door on the floor plan, there are dozens of keypoints, all in a cluster near each other. If we assume nicely that there are 20 doors on the floor, we could run the k-means clustering algorithm on the set of points where *k=20*, and we would be left with 20 points, each which are at the center of each of the door clusters. We can then simply place a mark at each of those centers and return that to the user.

In reality, using this algorithm is not that simple, as 1. we need to run it on an arbitrary floor plan, meaning we have no prior knowledge how many doors, or clusters, should be expected and 2. the feature matching does not give us perfectly desired results, meaning there will be redundant points scattered throughout the image that are not doors.

We can somewhat resolve the first issue by assuming a k-value that is large enough to account for any realistic floor plan. Rather, no real world building likely has more than 5000 doors, so we can set *k=5000* as an upper limit and still drastically decrease the number of points, as well as improve the run time.

Figure 8. Remaining features after k-means clustering.

After the initial clustering is finished, we are still left with many points. Now, must use a separate brute-force algorithm that is many times slower than k-means clustering. Since we provided the program a cropped image of the door, we have a general idea of how far away from a door keypoints will be from the center of the door. Thus, we can effectively find the center of every remaining cluster by, for every point, checking if there exists any points within a certain distance (the minimum of the width and length of the door image) from it. If there is a point within that distance, we calculate the midpoint of those two points, make that a new point, and then remove the two old points.

We repeat this process until all points are a sufficient distance away from each other. As expected, this algorithm takes an extremely long time, and also does not perfectly result in points that are at the center of each door. For the most part, it does leave points close enough to a door such that the user, or any one looking that the map, can clearly tell that the point designates a door.

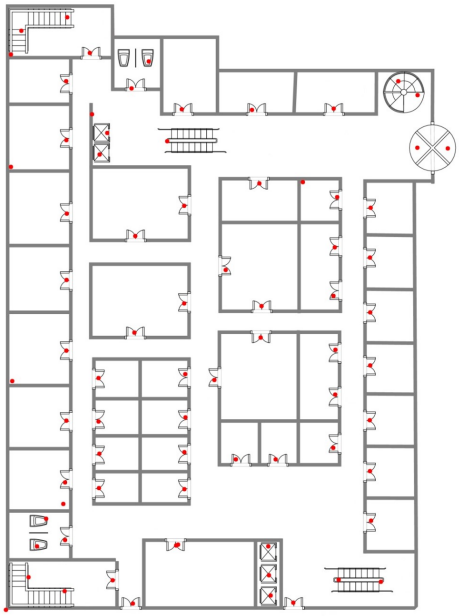


Figure 9. The final result with marked doors and some redundant marks.

This fulfills the basic functionality we were looking for in the software, despite its inaccuracies. At this point in the process, IBeaconMap would analyze the area around each remaining point and use machine learning to remove any redundant points, though this is unrelated to our main goal to analyze the accuracy and limitations of OpenCV. Regardless, it would not be hard to implement the same classifier used in IBeaconMap into this tool.

IV. RESULTS

In general, this tool produced mostly accurate results for the 3 floor plans being tested, all of which are floor plans used in IBeaconMap's testing; however, they are slightly worse than IBeaconMaps (when only using feature detection and matching). They miss a similar number of doors, but this tool leaves more redundant points, especially in the large area image.

		Correct	Incorrect		Processing Time (s)
			Missed	Redundant	
Shopping Mall	OpenCV	49	0	27	5.38
	IBeaconMap	49	0	40	18.77
Research Building	OpenCV	50	7	27	3.18
	IBeaconMap	63	4	16	11.30
Large Area	OpenCV	208	18	204	2247
	IBeaconMap	237	11	97	185.73

Table 1. A comparison between this tool's (OpenCV) and IBeaconMap's results.

The most significant difference between the two tools is the run time. For the 2 lower resolution images, this software runs much faster, but for the large image, it takes up to ten times as long (over 37 minutes). Run-time is not a major concern since the program only needs to be run once for any given floor plan, and for the smaller images, the difference in run-time between the programs is insignificant. The difference between the large images, 3 minutes versus 37 minutes, however, should be improved.

Another important goal in mind was the simplicity of actually using the tool, and although the user interface for this software is very simple, it is easy for anyone to use. All that is required from the user is to select the floor image, the door image, and then click submit. After that, the resulting image with marked doors is generated, saved, then displayed. Obviously, significant improvements can be made, such as displaying a progress bar, but from a functionality standpoint, it is sufficient.

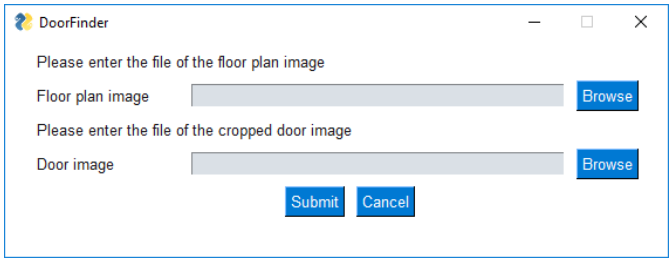


Figure 10. A screenshot of the tool's UI.

V. CONCLUSIONS

Although the software as is could certainly be used in a real-world setting, it is far from ideal. There are several things that can, and should be improved before moving to the next step of implementing a machine learning classifier to remove redundant points:

- The run-time for large resolution images can be drastically improved.

This could be done by tweaking the current algorithm that centers the keypoint clusters, and/or implementing a different data structure that could improve the speed of the current algorithm, as well open up different options altogether.

- Automatic parameter optimization for better accuracy.

Rather than use the same parameters for SIFT and brute force matching on every image, there could be a way to know how to produce the best results by analyzing different details of the input images and their features, and then change the parameters of SIFT, the brue-force matcher, and the clustering algorithms accordingly based on these factors. Since no assumptions can be made about the resolution or formatting of any given floor plan for this tool to work, this is perhaps the most important improvement to be made to ensure the tool works in any situation.

- An improved UI.

If these issues are dealt with and improved, then, with the addition of a machine learning door classifier, this would be an accurate, efficient, usable, and Open-source software tool that to be used for beacon deployment.

REFERENCES

[1] Cheraghi, Seyed Ali Namboodiri, Vinod Walker, Laura. (2017). GuideBeacon: Beacon-based indoor wayfinding for the blind, visually impaired, and disoriented. 121-130. 10.1109/PERCOM.2017.7917858.

[2] Cheraghi, Seyed Ali Namboodiri, Vinod Sinha, Kaushik. (2018). IBeaconMap: Automated Indoor Space Representation for Beacon-Based Wayfinding.

[3] Bradski, G. (2000). The OpenCV Library. Dr. Dobb39;s Journal of Software Tools.

[4] Lowe, D.G. International Journal of Computer Vision (2004) 60: 91. https://doi.org/10.1023/B:VISI.0000029664.99615.94