NETCPS REU
August 2, 2019

Indoor Navigation for the Blind:
Processing Digital Floor Plans Using OpenCV-Python for Beacon Placement

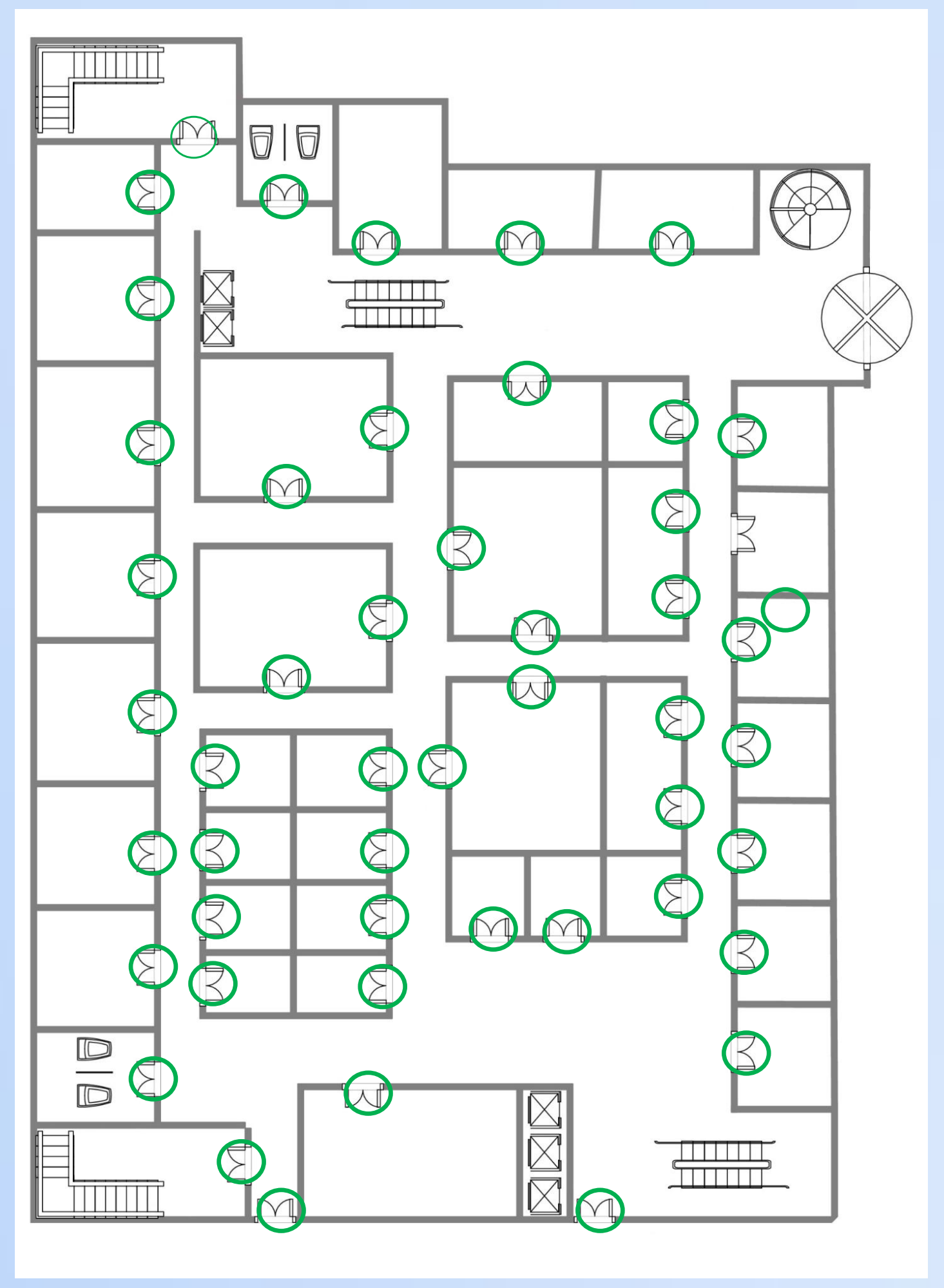
Charlie Broadbent | Faculty Mentor: Dr. Vinod Namboodiri

The Problem

When setting up a building for indoor navigation, one of the first steps is deciding on where to place Bluetooth beacons that will be used to transmit navigational information to users.

Perhaps the simplest way to do this is to manually have someone look over a floor plan; however, this can be time consuming, and prone to human-error, especially with very large floor plans.

What if there was a tool to allow us to automate this process? Luckily, there is: **IBeaconMap.** The problem is it that it is written in MATLAB, which means it cannot be distributed without a fee.



How can we automate this?

OpenCV, a library of programming functions, is an alternative for implementing computer vision.

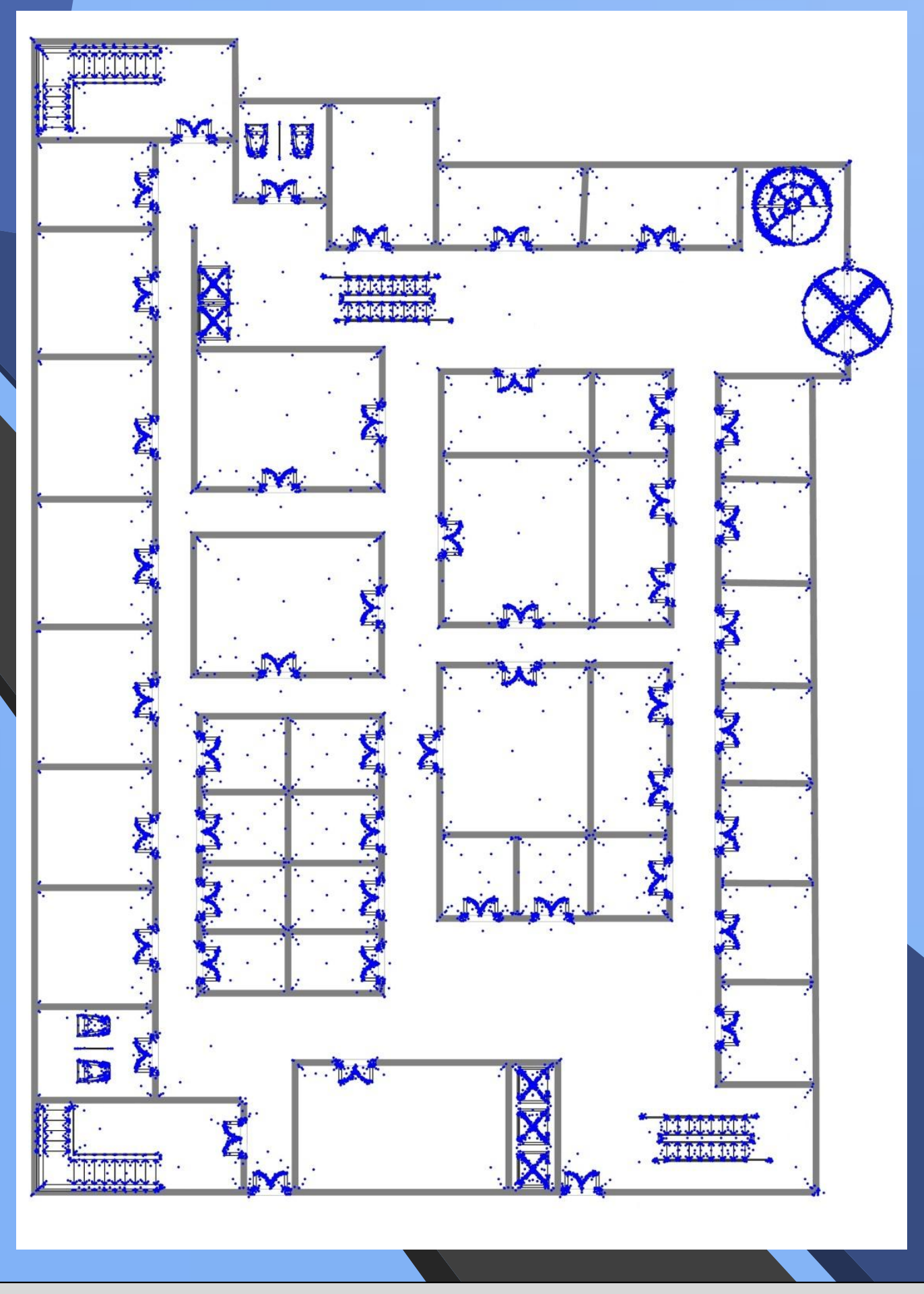
Using OpenCV (specifically, the **Python** libraries), can we create a software similar to **IBeaconMap** that is just as accurate?

What are the goals of this tool?

Since the base functionality of the program needs to locate the doors on a floor plan, this is what I set out to achieve. Therefore, we want a tool that:

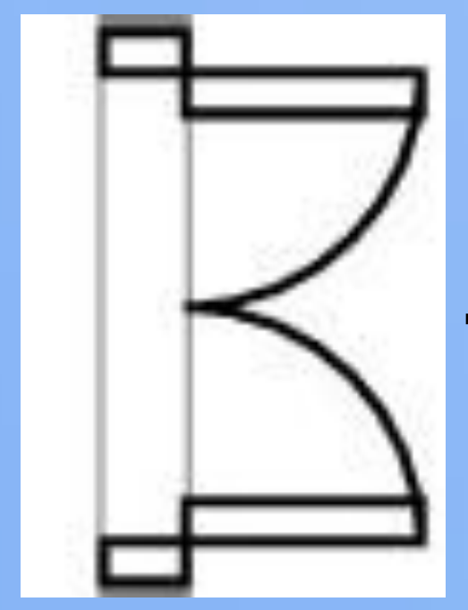
1. Marks as many doors on a floor plan as possible without missing any.
   i. At the same time, it should make as many *least* redundant marks as possible (i.e. marks that are not doors).
2. Is simple to use for a person of any level of computer experience.

How does it work?

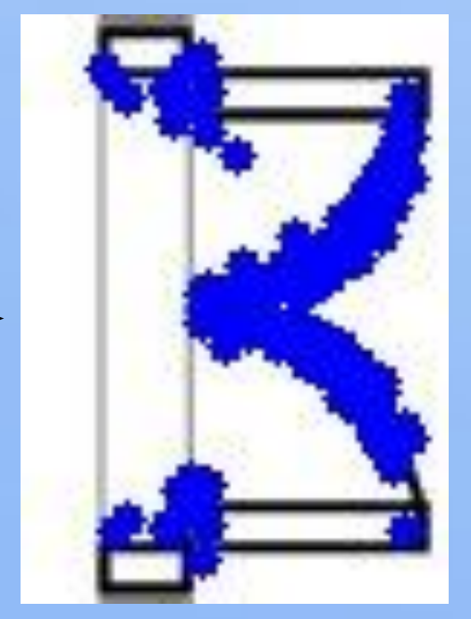


Floor plan with all features marked

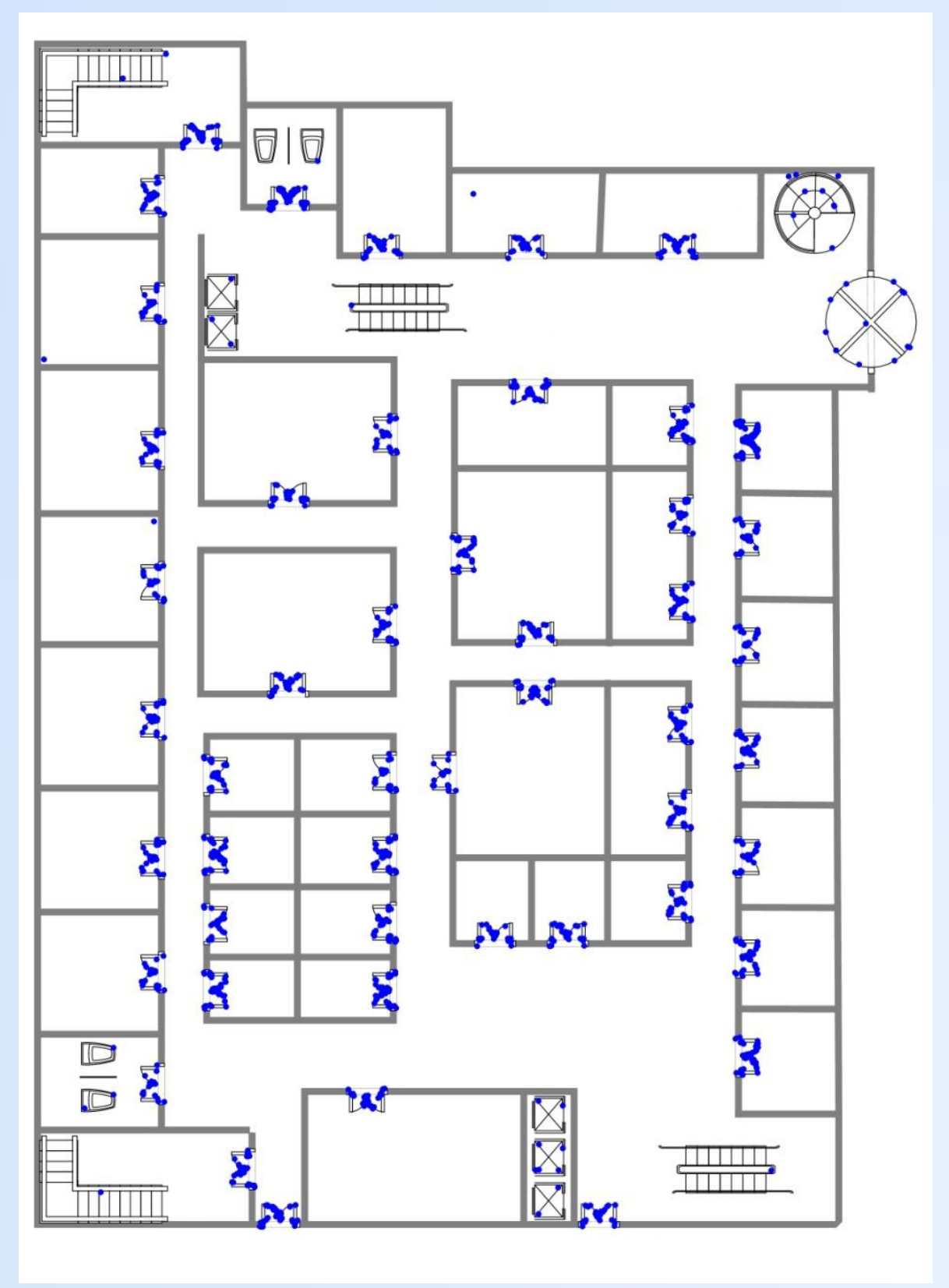
First, we run a feature detection algorithm (I chose **SIFT**) on an image of the floor plan and a cropped image of a door from the same floor plan.



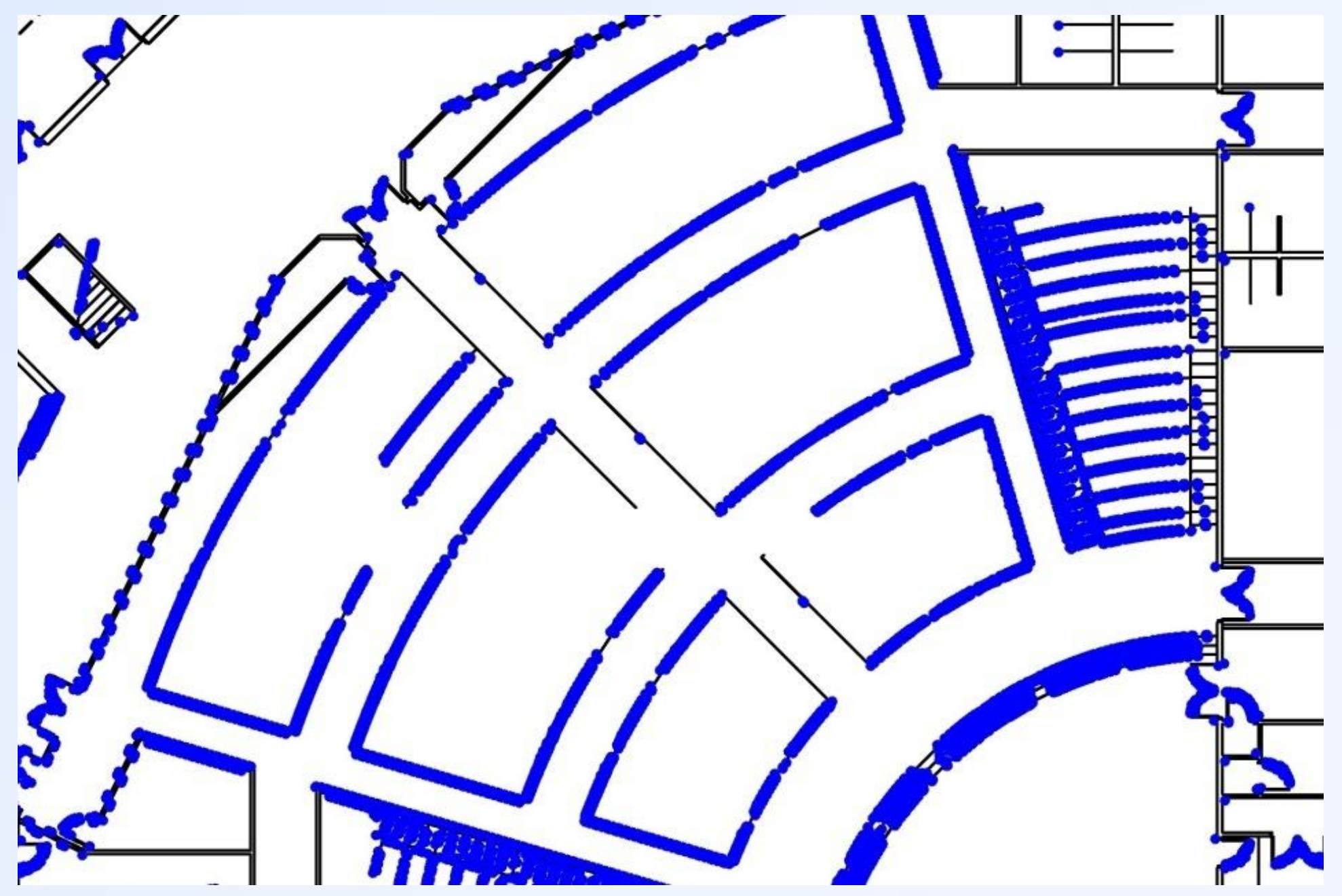
→ SIFT →



This gives us a set of **keypoints** and **descriptors**. We then match the features from the floor plan to the features of the door. This leaves us with the features in the floor plan that are *most likely* part of a door.



Matched features



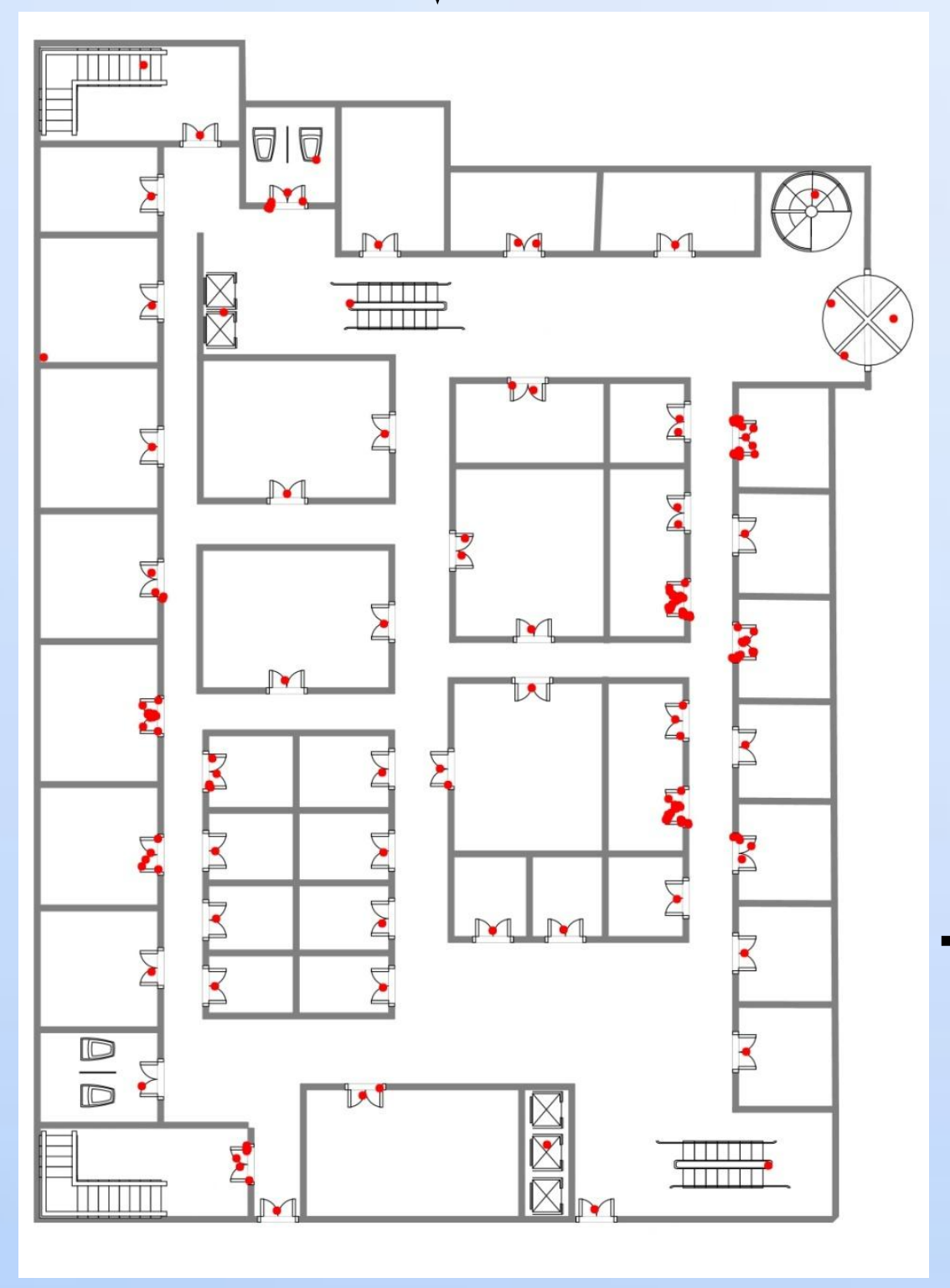
If the floor plan has a lot of curves, they can be detected to be doors.

Feature matching does not work perfectly, however. It leaves is with many more points than we need, and we need to somehow reduce these points down to a single point that is on top of a door

↓

k-means clustering

↓



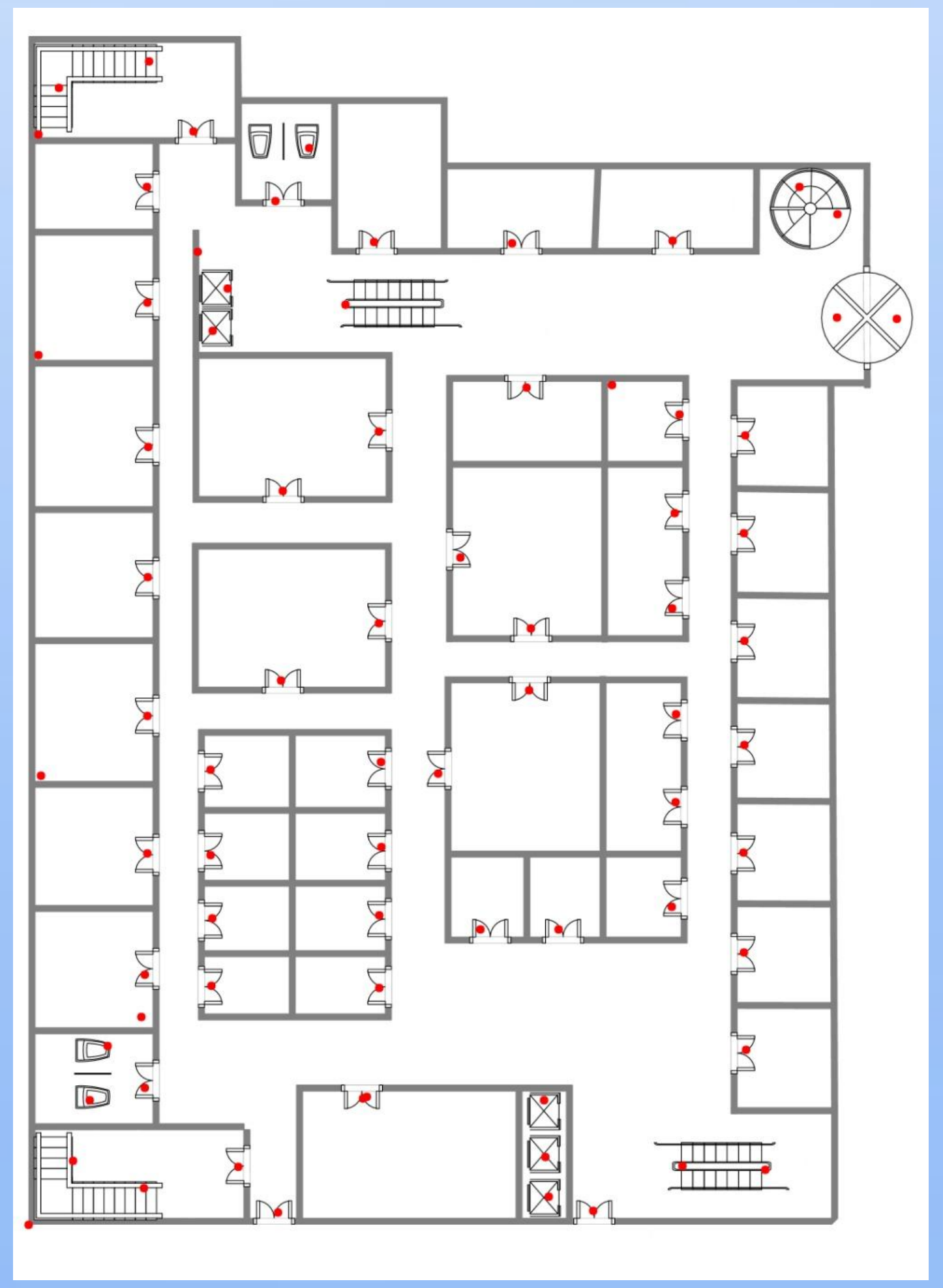
We can make a significant reduction using **k-means clustering**, which will find the centers of some clusters of keypoints, but not all, meaning we must reduce further.

→ Brute-force checker

↓

We reduce the rest of the points by checking if other points exist within a certain radius. If there exists such a point, we add a new point at the midpoint. This process continues until all points are sufficiently far away from each other.

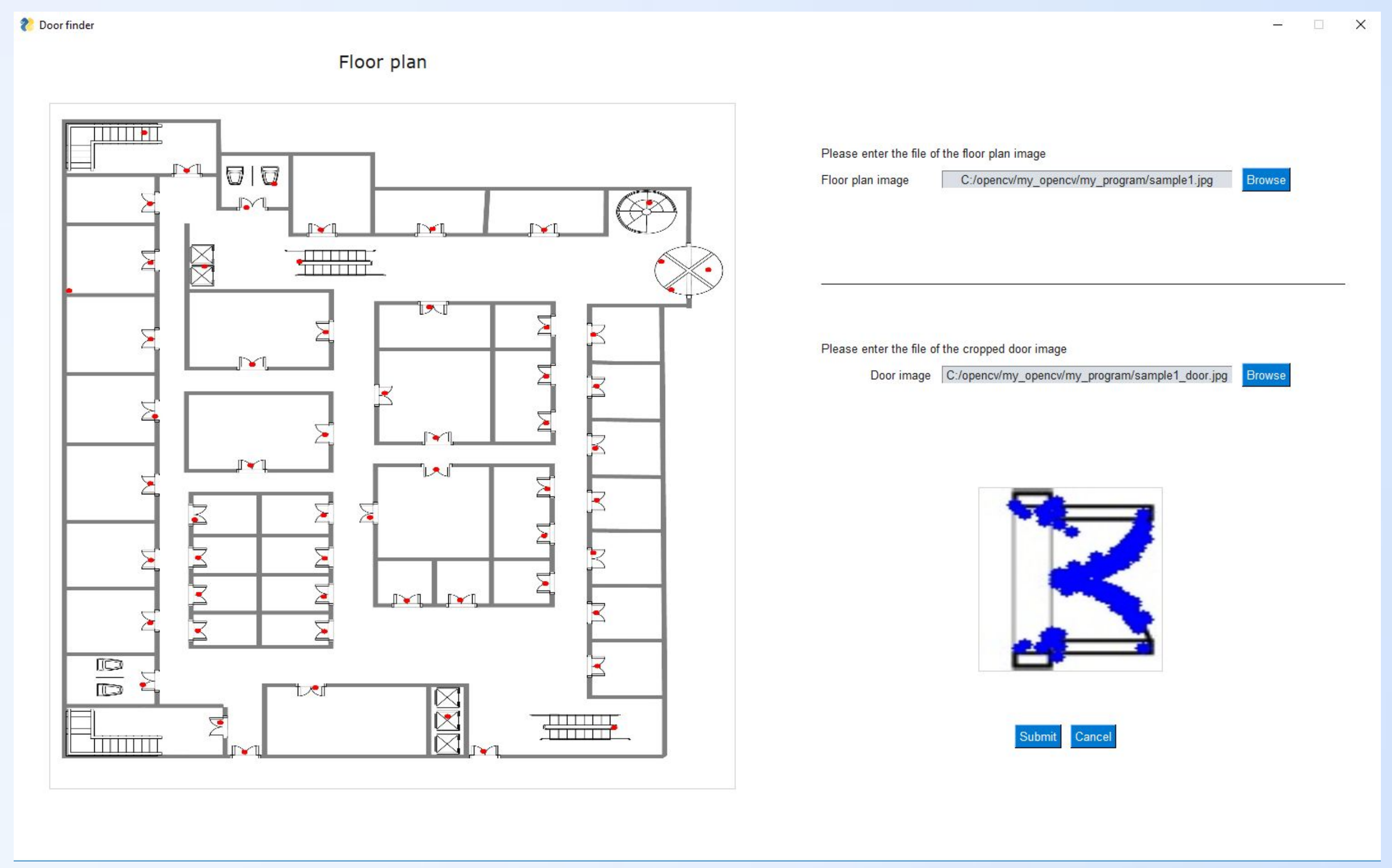
The leftover points are not perfectly placed, and there will exist redundant points—so how accurate is this process?



Final result

UI

The UI is barebones, but very simple.



Door finder

Floor plan

Please enter the file of the floor plan image

Floor plan image: C:/opencv/my_opencv/my_program/sample1.jpg [Browse]

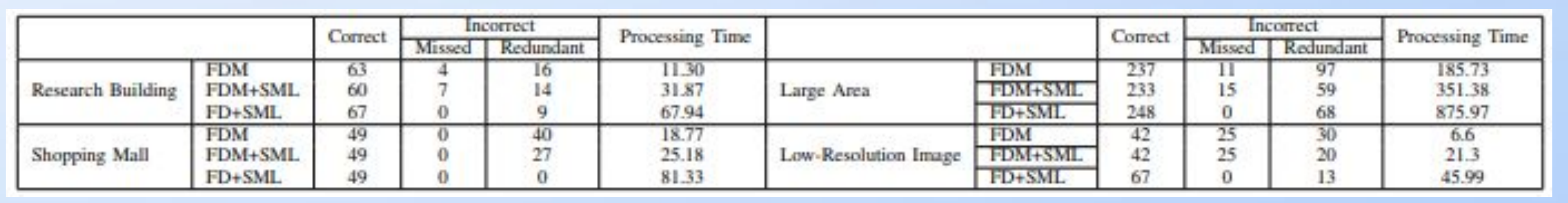
Please enter the file of the cropped door image

Door image: C:/opencv/my_opencv/my_program/sample1_door.jpg [Browse]

[Submit] [Cancel]

Results

	Doors found	Missed doors	Redundant points	Processing time (s)
Shopping Mall	49	0	27	5.38
Sample plan (low res)	16	2	16	1.33
Research Building	50	7	27	3.18
Large Area	208	18	204	2247



		Correct	Incorrect		Processing Time
			Missed	Redundant	
Research Building	FDM	63	4	16	11.30
	FDM+SML	60	7	14	31.87
	FD+SML	67	0	9	67.94
Shopping Mall	FDM	49	0	40	18.77
	FDM+SML	49	0	27	25.18
	FD+SML	49	0	0	81.33

		Correct	Incorrect		Processing Time
			Missed	Redundant	
Large Area	FDM	237	11	97	185.73
	FDM+SML	233	15	59	351.38
	FD+SML	248	0	68	875.97
Low-Resolution Image	FDM	42	25	30	6.6
	FDM+SML	42	25	20	21.3
	FD+SML	67	0	13	45.99

IBeaconMap's results (lower table). The research building, shopping mall, and large area floor plans used were the same images for both programs. FDM indicates the use of only feature detection and no machine learning.

For the most part, it is accurate, though slightly less than IBeaconMap.

Processing time for very large images between the two tools also varies greatly (up to 37 minutes vs. IBeaconMap's 3 minutes).

What improvements need to be made?

Better UI

A more efficient clustering algorithm and/or data structure to store keypoints to improve processing time on large images.

Improved optimization of OpenCV's feature detection, feature matching, and clustering algorithms to improve accuracy and reduce redundancy.