The Cup Collector
ROB5

By:
Jes Jepsen
Keerthikan Ratnarajah
Mathias Neerup
Nicolai Lynnerup

University teacher:
Jimmy Alison Jørgensen
jimali@mmmi.sdu.dk

Dato: December 3, 2014

Contents

1 The Cup Collector

The cantina at SDU has in a period of time collected empirical data that suggests that researchers and students at SDU are lazy and forgetful when it comes to bringing back used cups to the cantina. To avoid situations where there are no coffee cups and thereby loosing coffee sales (or fresh students), the cantina has decided to assign the task of collecting cups form offices and hallways of SDU, to one person for 2 hours per day.

A master student with some knowledge on vision and point clouds has using a Kinect and a robot arm implemented an algorithm to detect and collect cups within a distance of 2 meters. Also he has implemented the stable grasping of cups that are within 1 meter of the robot. Unfortunately, this master student never attended ROB05 and therefore has no skill in navigating robots.

The problem is therefore presented to this year's class in ROB05.

The task contains 3 parts. All 3 must be targeted in the project.

1.1 Limitations and information

- The map "complete_map_project.pgm" use a scale 10pixel:1m and figure 1 shows the complete map.
- The map use the scale 10px to 1 meter
- The robot can only carry 20 cups at the time
- The robot is NOT a point robot. It has a circular shape with a radius of 0.4m
- The robot can drive up to 5km per hour
- The elevators are denoted with pixelvalues 128,129,130,131 and 132.

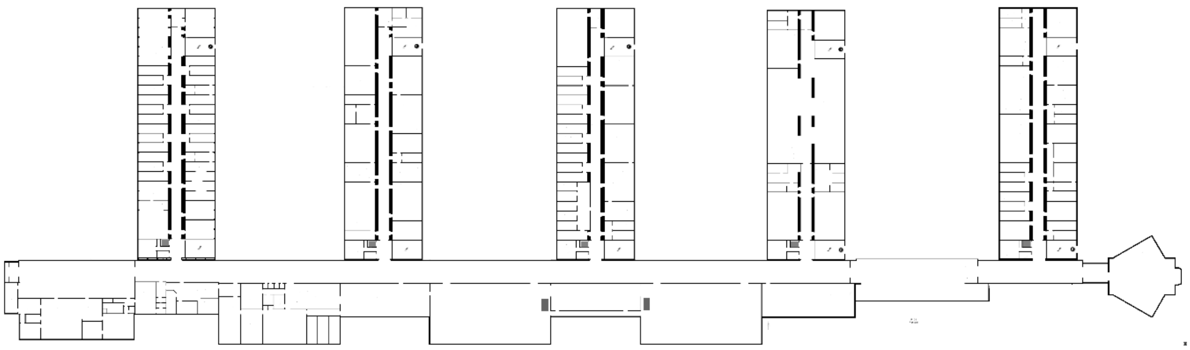

Figure 1: The complete map of SDU.

2 Planning

All rooms in the map must be checked in order to find cups. The robot must be within 2 meters of a cup in order to actually detect the cup and within 1 meter in order to collect it. Cups are marked in the map using one pixel with grayscale value 150. Cups can be unloaded at the two offloading stations in the cafeteria. The offloading stations are represented with pixel values 100. The robot must start and end at an offloading station.

You are free in regard in choice of algorithms. However, please document what algorithm you choose, how many kilometers the robot moves and how long it takes to calculate the path the robot takes.

All planning is done offline, and it involves the functions:

- **Wavefront** from the two unloading–stations
- **Detection** of the rooms and blocks
- **Priority queues** for each block and the

2.1 Solution

The first thing that happens in the planning part, is that a wavefront is generated from the two positions where the unloading–stations are located in the cafeteria. Cups are also seen as obstacles, when the wavefront is generated. The pixel values for the two unloadings stations are

$$\begin{aligned} Unloading - station\,1 &: \qquad (3100{,}1400) \\ Unloading - station\,2 &: \qquad (2150{,}1400) \end{aligned}$$

and figure 2 shows how the wavefront expands from the two unloading–stations in the complete map. The unloading stations are the two darkest areas on the figure.

The wavefront–values is stored in a 2D–array, so when it is time for emptying the container, it is possible run the wavefront–function quickly in the online part. The generated array takes up some memory, but it saves time keeping the wavefront–map to the two unloading stations. When the cup container is filled, a path to the nearest unloading–station is found by using the offline wavefront function, `makeWaveFrontoffline(currentX,currentY)`. For each pixel the robot moves through the wavefront path to a station, each coordinate is pushed into a vector–pair, so it can take the same path back to where it were interrupted by popping the values from the vector–pair. Just like Hansel and Gretel with the breadcrumbs, except that the robot will not get lost in the woods, find a gingerbread house, be fattened and so on.

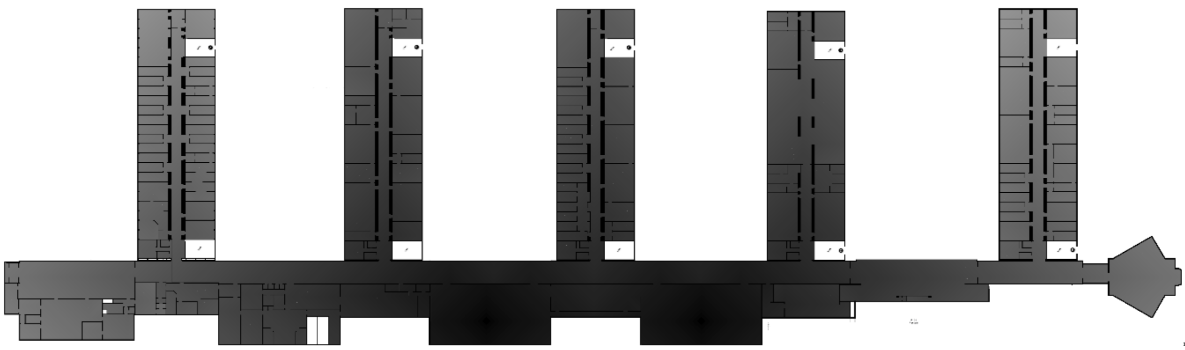

Figure 2: How the wavefront expands from the two unloading–stations in the complete map.

Detection of a room has four steps

1. First the upper left corners (C_{ul}) of the squares are detected. Each pixel in the map is checked with a 3x3–mask, shown in table 1, which looks at if the pixel value neighbors are obstacles, which corresponds to a left corner. If the mask matches, then the pixel is marked as a C_{ul} and the position for the pixel is stored in a data type, called `square`.

0	0	0
0	P	
0		

Table 1: 3x3 mask for detecting upper left corners

2. Next the upper right corners (C_{ur}) is found by taking the the position for each C_{ul}, and then keep moving to the right, until it hits an obstacle. When it hits an obstacle, the pixel just before is marked as a C_{ur} and stored in `square`. On table 2 is this approach shown.
3. Next the lower left corners C_{ll} is found, just like the C_{ur}, by taking the the position for each C_{ul}, but instead it keeps moving down, until it hits an obstacle. When it hits an obstacle, the pixel just before is marked as a C_{ll} and stored in `square`. On table 2 is this approach shown.
4. The lower right corner (C_{lr}) is not detected like the other three corners, it is calculated by taking the height distance (in pixels) between C_{ul} and C_{ll}, and assuming that the C_{lr} has the same height distance from the C_{ur}. Like the other corners, the C_{lr} is marked and stored in `square`.

0	0	0							
0	C_{ul}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	C_{ur}	0
0	$\vdots$							$\vdots$	
	$\vdots$							$\vdots$	
	C_{ll}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	C_{lr}	
	0								

Table 2: Detection of the room corners

When all the rooms are detected, the center of each room is calculated, by taking the half of the weight and height of the room, and stores it as a vector–pair. Each room in the list is then check wherever a room is actually a room or a hallway by looking at the relationship between the wall of the room. Some thresholds are set, so if the relationship excites these threshold, then the room is detected as a hallway instead. On figure 3 can it be seen how the different rooms are connected to each hallway. The hallways has a priority queue, which prioritizes the room after the distance between the center of the hallway to the center of the room. Each room is put in the priority queue of that hallway center that is closest to the center of the room. This priority for each hallway is the order of which room that are cleaned and emptied for cups first.

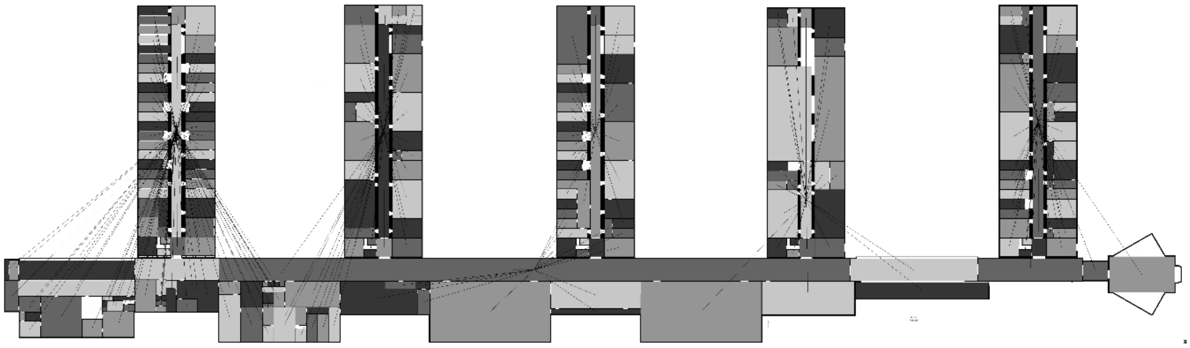

Figure 3: An overview of how the different rooms are detected and connected to each hallway on the complete map of SDU.

2.2 Results

3 Coverage

The Dean feels that it is not economically justified to by a robot system for 80.000 euro just to collect cups. Hence, it is interesting to have the robot do a second task, namely washing of the floors. Therefore calculate a coverage path that covers most of the floor in the map. Again the robot must start and end at the offloading stations.

Again, you are free to choose algorithm and you should document the choice, the distance in kilometres that the robot moves and how long it takes to compute the coverage path.
All coverage is done online, and it involves the functions

- **Wavefront** from the robots current position to the center of an other room

3.1 Method

The coverage plan for the complete map is

1. Block A
2. The Big Hall
3. Block B
4. Block C
5. Block D
6. Block E

When the room is fully covered, then the center of the next room is popped from the priority queue of the block, that the robot is currently in. A wavefront is then generated from that center-position, until the wavefront hits the position of the robot, and then the path to that center is found. The wavefront is stopped as soon as it hits the robots position, because the robot will never gonna use the rest of wavefront, and it saves a lot of time and memory by not making a full wavefront map. This makes it also possible to run the wavefront online, because most often the robot does not have to drive so far to get to the next room center.

3.2 Results

How long does it take?

3.3 Conclusion

What works and what does not? Why?

4 Localisation

Finally, a method to compute the state (configuration) of the robot is required. The method should be applied to a real system such as the Nexus platform from the course and due to the size of the university it is important that the robot is able to use features to precisely measure its whereabouts. These features could be based on the Hokoyo 2D laser scanner mounted on the robot.
Again document what algorithm, and test how well it performs. You should at least write what model you choose for the robot and show that the localisation works better than odometry alone.

4.1 Method

How is the problem solved?

4.2 Results

How long does it take?

4.3 Conclusion

What works and what does not? Why?

5 Future work

What changes could make the solution more optimal/feasible?