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.
- 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.

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:

- Offline Wavefront
- Graph and sub–graphs

2.1 Method

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. The pixel values for the two unloadings stations are

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

and figure 1 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 it is possible to access the generated array quickly when the cup container is filled and there shall be found a path to the nearest unloading–station for emptying the container.

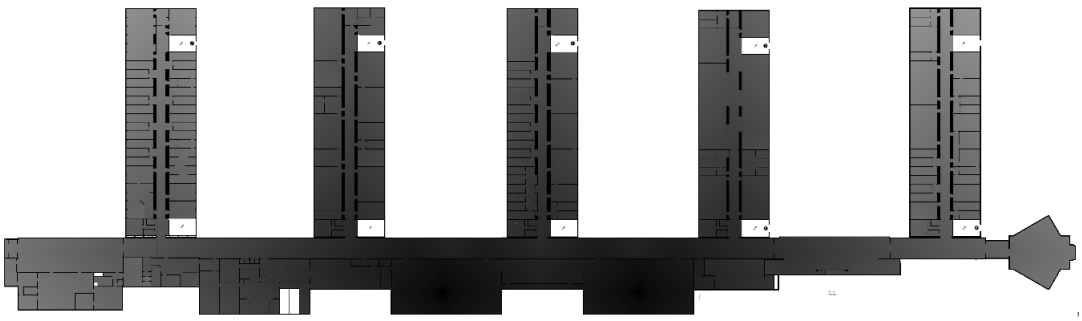

Figure 1: How the wavefront expands from the two unloading–stations in the complete map.

To control the order of the rooms to be cleared, has been made a graph consisting of sub–graphs, one for each block. Each room in a blok is connected to the center of the block, and each center of the blocks are connected to the closest unloading–station. To detect a room, the program 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 the pixel 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 list.

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 a data type, called `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}	⋯	⋯	⋯	⋯	⋯	⋯	C_{ur}	0
0	⋮							⋮	
	⋮							⋮	
	⋮							⋮	
	⋮							⋮	
	⋮							⋮	
	⋮							⋮	
	C_{ll}	⋯	⋯	⋯	⋯	⋯	⋯	C_{lr}	
	0								

Table 2: Detection of the room corners

2.2 Results

How long does it take?

2.3 Conclusion

What works and what does not? Why?

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.

3.1 Method

How is the problem solved?

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?