The Cup Collector
ROB5

By:
Jes Jepsen
Keerthikan Ratnarajah
Mathias Neerup
Nicolai Lynnerup

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

Dato: December 11, 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 two 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 two meters. Also he has implemented the stable grasping of cups that are within one 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 tree parts. All tree 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, which on the complete map is equivalent to
$$r = 4\,px$$
- The robot can drive up to 5 km/hour, which on the complete map is equivalent to
$$v = 50.000\,px/hour$$
- The elevators are denoted with pixelvalues 128,129,130,131 and 132.

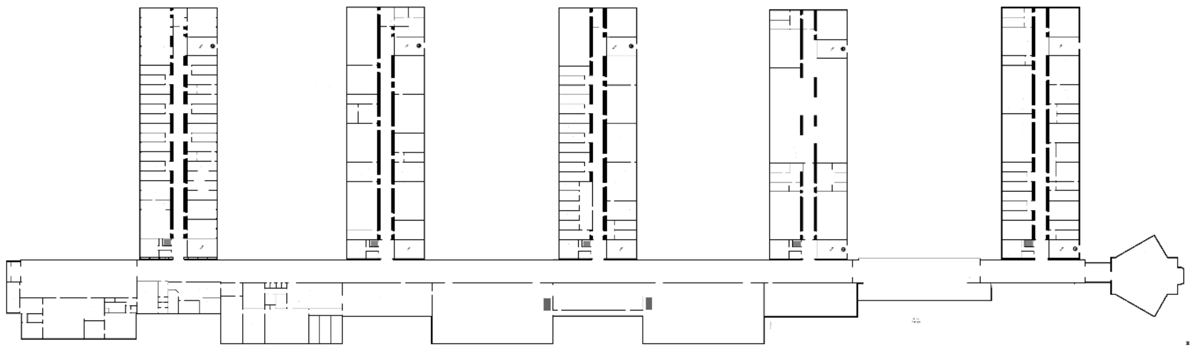

Figure 1: The complete map of SDU.

1.2 Planning

All rooms in the map must be checked in order to find cups. The robot must be within two meters of a cup in order to actually detect the cup and within one meter in order to collect it. Cups are marked in the map using one pixel with grayscale value 150. Cups can be unloaded at one of 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.

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

- **Wall expansion**, so the free–space, Q_{free}, is four pixels away from the walls. In Q_{free} the robot will not collide with the walls or other obstacles.
- **Offline wavefront** from the two unloading–stations.
- **Detection** of the rooms and blocks.
- **Priority queues** for each block and the a priority queue for the big hall way on the map.

1.2.1 Solution

The first thing in the planning part is, that all obstacle (not cups) pixels are expanded with four pixel, as illustrated on figure 2, so the complete map would look like figure 3.

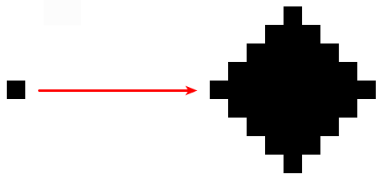

Figure 2: Expansion of one pixel

The map on figure 3 is only used with the offline wavefront, to ensure that the robot not collides with the walls or other obstacles while unloading the cups. The online wavefront in section 1.3 also uses this map, for the same reasons, when it moves to the next room.

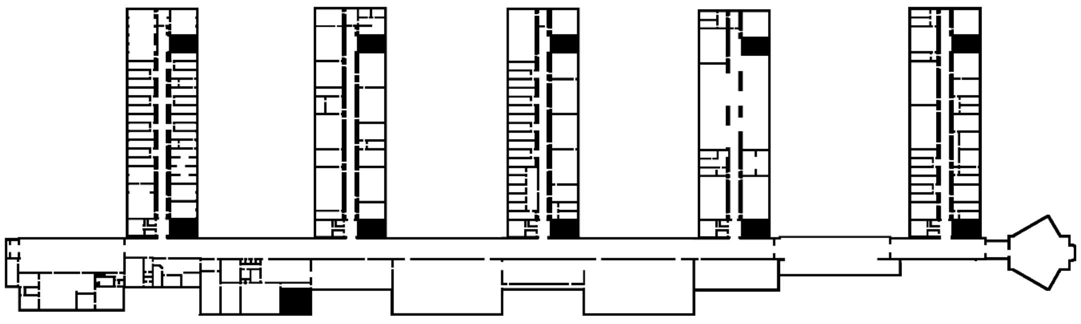

Figure 3: Expansion of one pixel

The offline 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

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

and figure 4 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. 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.

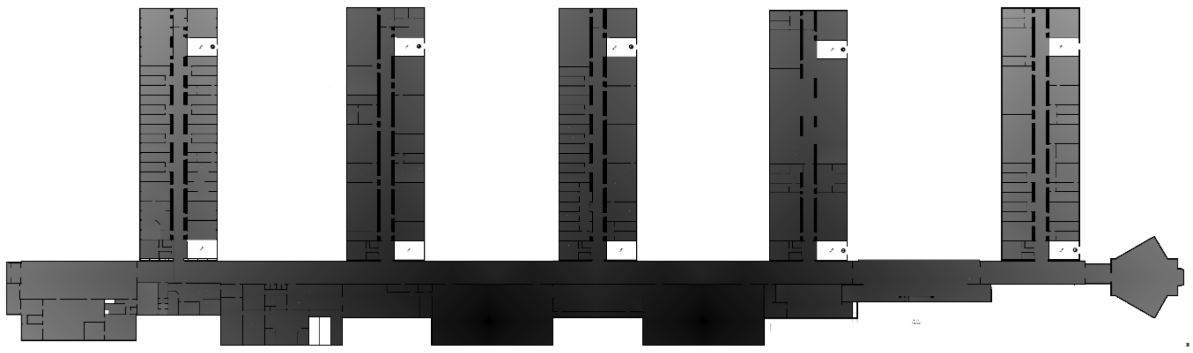

Figure 4: How the wavefront expands from the two unloading–stations in the complete map.

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 5 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.

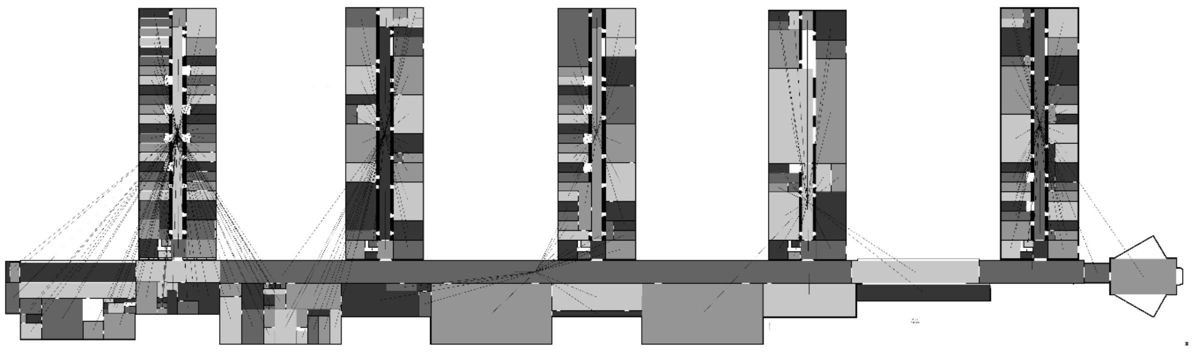

Figure 5: An overview of how the different rooms are detected and connected to each hallway on the complete map of SDU.

1.2.2 Alternatives

As alternatives to planning could the following methods also be used:

- Determine the appropriate edges for the vertical extensions, i.e., the two edges that the sweep line lies between. To make it feasible with this map, the sweep line should be sweeping the map with a angle different from 0 or 90 degrees.

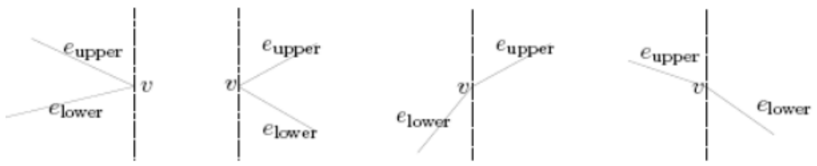

Figure 6: The kinds of events

1.3 Coverage

The Dean feels that it is not economically justified to by a robot system for 595.176 DKK just to collect cups. Hence, it is interesting to have the robot do a second task, namely washing of the floors.

All coverage is done online, and it involves the functions

- **Online wavefront** from the robots current position to the center of an other room.
- **Coverage of a room** is similar for every room.
- **Collecting the cups** within a radius of 4px.
- **Offline wavefront**, as described in section 1.2, when the robot has collected 20 cups and needs unload them at one of the unloading–stations, by using the 2D–array from the offline planning.

1.3.1 Solution

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

Figure 7 illustrates covering a room. The dark gray pixels are where the robot is moving and the light gray area is what the robot has covered for cups. The coverage is done in the following steps

1. Starts at upper left corner of the rooms freespace, Q_{room}, and keep moving down to the lower left corner of the Q_{room}.
2. Turn left 90 degrees and move 8 pixels
3. Turn left 90 degrees and keep moving until the limit of the Q_{room} is detected.
4. Turn right 90 degrees and move 8 pixels
5. Turn right 90 degrees and keep moving until the limit of the Q_{room} is detected.
6. Keep repeating the step 2 to 5 until the same y–coordinate as the upper right corner of the rooms Q_{room} is detected. At this point, either the robot has the same position as the upper right corner or the lower right corner of the Q_{room}.
    - If it is position of the lower right corner of the Q_{room}, then turn 90 degrees to the left and keep moving until the robot has the same position as the upper right corner of the Q_{room} (Case A on figure 7). The room is fully covered and the robot can move to the next room.
    - If it is position of the upper right corner of the Q_{room}, then turn right 90 degrees and keep moving until the robot has the same position as the lower right corner of the Q_{room} (Case B on figure 7). The room is fully covered and the robot can move to the next room.

This coverage algorithm is very simple and it is only possible to use it, because all the rooms are squared. 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.

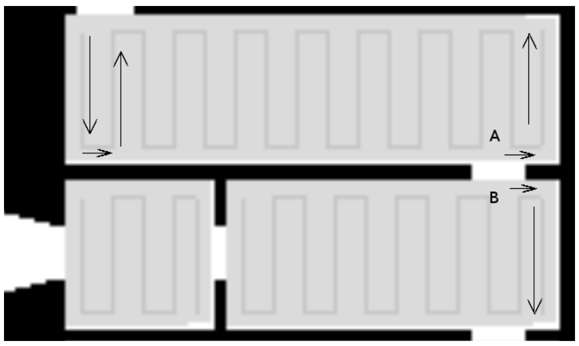

Figure 7: Coverage of a room in the complete map.

Figure 8 shows the complete map covered with this projects program.

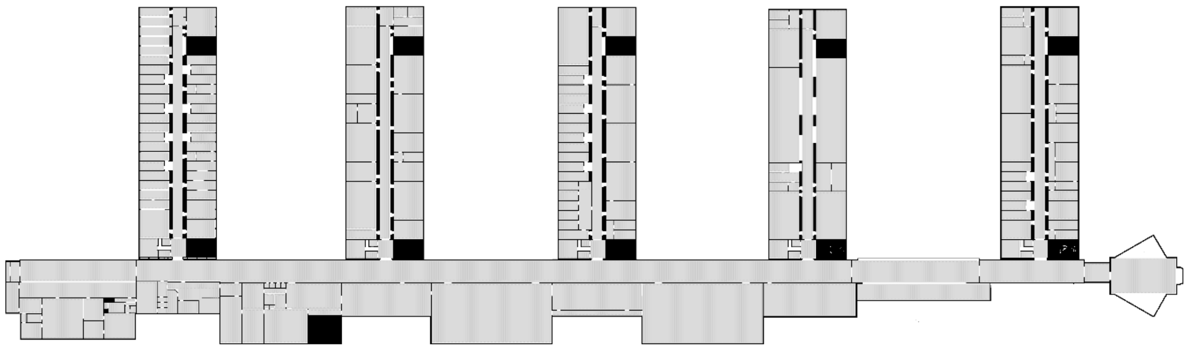

Figure 8: Coverage of the complete map.

1.3.2 Results of the planning and coverage

The calculation of the path the robot takes is *time sec.*

While the robot moves around in the map, a counter keeps track of how many pixels the robot had moved during that time.

If only focusing on washing the floor, the robot moves 513.008 px. The time it takes it to clean the schools floor is then

$$t = \frac{513.008\,px}{50.000\,px/hour} = 10{,}26\,hours$$

If the robot cleans the floor at night, e.g. from 8 PM to 6.15 AM, it would be realistic for it to do the job, without being interrupted or disturbed by the students. The program does not take into account for obstacles that suddenly appears, e.g. a university teacher, standing in one of the classrooms, writing something on a blackboard. Also, it would not be pleasant for the teacher to be hit by the robot, with a speed of 5 km/hour.

Assuming that

- One man hour cost 208 DKK
- The cleaning lady works within the normal working hours
- She uses a washing machine with the same speed and and radius, as the robot
- She uses a normal workday (8 hours) on cleaning the same area.
- The robot does not need maintenance.

then, it still would be feasible for the dean to invest in this robot. In the long run, a cleaning of the floor, done by the cleaning lady, would cost the dean 1664 DKK. If the floor needs to be cleaned every 24 hours, then at one point, within the robots durability, it will make a profit. Saying a cleaning robot costs 595.176 DKK, then approximately after a year it will by paid of, by the saved man hours used on cleaning the floor.

1.4 Localization

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.

In the real world, no actuator is perfect: they may overshoot or undershoot the desired amount of motion. So when a robot tries to drive in a straight line, it will inevitably curve to one side or the other due to minute differences in wheel radius. Figure 9 shows how the position of a robot can get more and more uncertain, as it moves in straight lines. A robot becomes less sure of its position if it moves blindly without sensing the environment, so it is necessary to sensors to localize the robot in the environment, and not only count on the odometry alone. To solve this, on the robot is a 2D laser range scanner mounted to sense the environment around it.

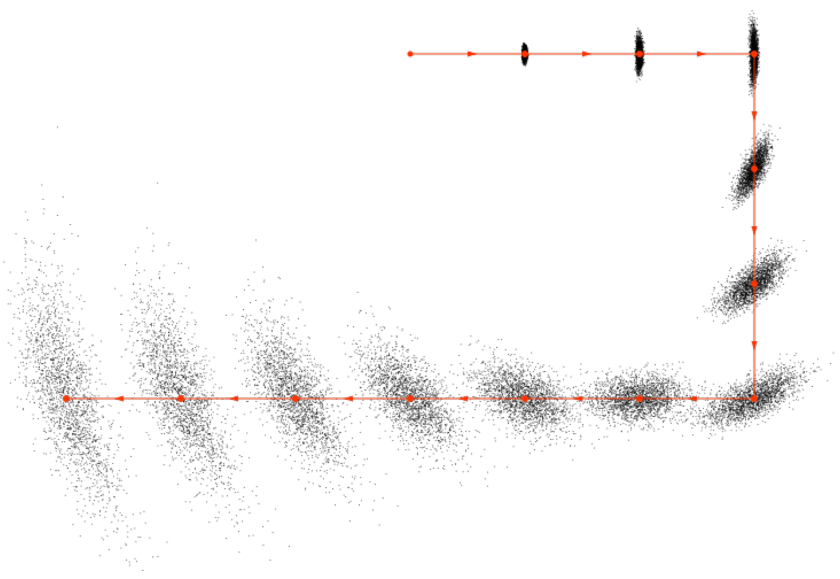

Figure 9: Belief after moving several steps for a 2D robot using a typical motion model without sensing.

All localization is done online, and it involves the functions:

- **Monte Carlo localization (MCL)** for determining the localization of the robot as it moves. This algorithm is normally used for robots to localize using a particle filter.
- **Set linear speed** for the robot, to determine how fast it should move
- **Set rotations speed** on each wheel of the robot, so it possible for it to turn 90 degrees to either left or right.

1.4.1 Solution

The MCL is given a map of the environment, and then the algorithm estimates the position and orientation of a robot as it moves and senses the environment. This is used to determine the position of the robot, by following[1]

1. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, i.e. a hypothesis of where the robot is. First N numbers of normal distributed hypothesis of the robots position is made of the entire map. At this point, the robot has no information about where it is and can only assumes it is equally likely to be at any point in configuration space.
2. All the hypothesis is compared to what the 2D laser range scanner sees, and that hypothesis with the highest probability, i.e. how well the actual sensed data correlate with the predicted position, an is assumed being the robots current position.
3. Whenever the robot moves, it shifts the particles to predict its new state after the movement, so, N normal distributed hypotheses surrounding our current position–hypothesis is made within a smaller area around the predicted current position. Maintaining the entire map as sample size would be computationally wasteful and make the the calculations less precise.
4. Step 2 to 4 is repeated with the previous belief as input, because ultimately, the particles should converge towards the actual position of the robot. This is repeated until the robot is either turned off or done collecting the cups.

Figure 10 shows the implemented Monte Carlo method on the robot. The yellow dots are the hypothesis that is normal distributed around the robot and the red dot is the robots current position according to the hypothesis and what the 2D laser range scanner sees. All white pixels are free space and the black pixels are obstacles. The robot is seen as a point robot in this example.

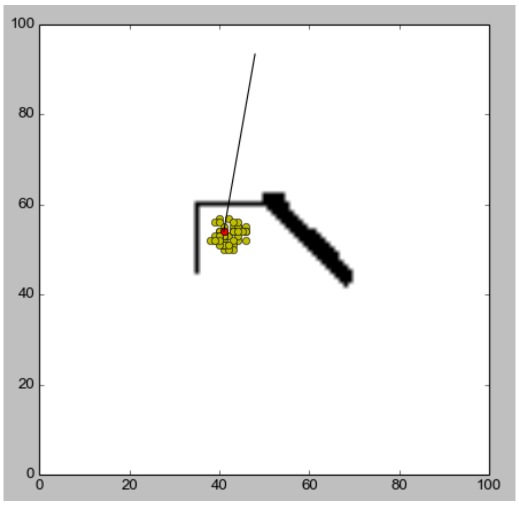

FIGURE 10: MONTE CARLO HYPOTHESIS (YELLOW DOTS) AND THE CURRENT POSITION (RED DOT)

[1]http://en.wikipedia.org/wiki/Monte_Carlo_localization

1.4.2 Results of the localization

To test how accurately the implemented Monte Carlo method detects, where on the map the robot is located, a experimental setup for testing the robots real and virtual position is shown on figure 11. On the figure, a construction similar to the map on figure 9 is made and 10x10 cm is drawn on the floor. This indicates each pixel on the map and makes it possible to compare the current real position with the virtual position.

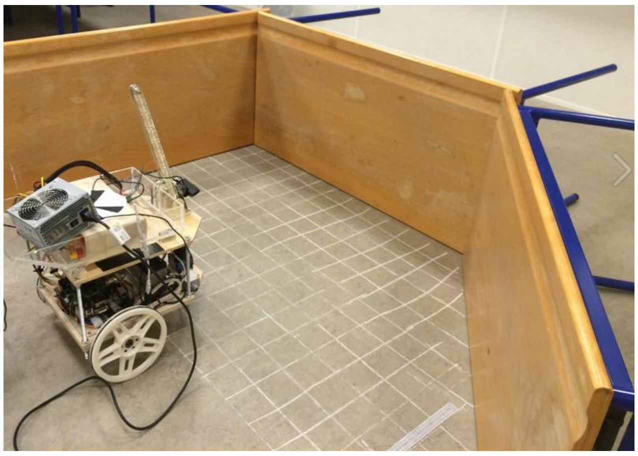

Figure 11: The experimental setup for testing the robots real and virtual position

The steps for testing is done by the following steps

1. The robot is moved to a position
2. The real and virtual position is noted
3. The real and virtual degree is noted

The results is shown in table 3 and by look at the collected data, it can be seen that the virtual position correspond well to the real position.

Mesurement 1	pos x		pos y		Mesurement 4	pos x		posy	
Real position	39	px	53	px	Real position	47	px	53	px
Virtual position	39	px	53	px	Virtual position	39	px	46	px
Real θ	0	degrees			Real θ	90	degrees		
Virtual θ	0-12	degrees			Virtual θ	9-15	degrees		
Mesurement 2	pos x		pos y		Mesurement 5	pos x		pos y	
Real position	42	px	56	px	Real position	48	px	55	px
Virtual position	43	px	56	px	Virtual position	51	px	55	px
Real θ	0-12	degrees			Real θ	335	degrees		
Virtual θ	1-5	degrees			Virtual θ	328-331	degrees		
Mesurement 3	pos x		pos y		Mesurement 6	pos x		pos y	
Real position	40	px	56	px	Real position	44	px	51	px
Virtual position	39	px	56	px	Virtual position	45	px	51	px
Real θ	45	degrees			Real θ	0	degrees		
Virtual θ	45-50	degrees			Virtual θ	5-10	degrees		

Table 3: Results

Testing of the localization and direction was only done on a small map, with few and unique angles, which made it easier for the robot to detect its position. If the map has been larger and with many similar angles, the precision of the robots positions would be expected to be worse, but it would still be better than depending on odometry alone.

1.4.3 Alternatives

As alternatives to localization could the following methods may also be used:

- Kalman filter for smoothing noisy data and providing estimates of parameters of interest, e.g. the position of the .

2 Future work

To improve the coverage time, the planning part should be seen as a the Travelling Salesman Problem (TSP). Currently, it is the shortest distance from the center of the block to each center of room in the block that determines the order the rooms are covered. The TSP asks the following question:

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?[2]

To do this, the following requirements must be met

- The model is a complete map, i.e. all rooms and blocks are connected by an edge. Long edges will only complete the graph, not affect the optimal tour.
- Each weight of the edge is the distance between the two vertexes
- All edges are undirected

The most direct implementation would be using the brute force method, at search for the cheapest route around each vertex. The running time for this approach would be $O(n!)$, but there are 100-130 rooms, so it would take way to long time to use that approach, so if the problem should be solved, the Held–Karp algorithm would be the choice of implementation, because it has a time complexity of

$$O(n^2 2^n)$$

or if another algorithm with a better time complexity.

Another improvement could be to take into account for suddenly changes, while the robot cleans the rooms and collects the cups. Better hardware and improvement of the software is needed to make this possible, but if it is done, then the robot would be able to clean and collect cups around the clock.

[2]`http://en.wikipedia.org/wiki/Travelling_salesman_problem`