The Cup Collector
ROB5

Lukas Christoffer Malte Wiuf Schwartz, Michael Kjær Schmidt,
Mikael Westermann, Mikkel Skaarup Jaedicke & Nikolaj Iversen

Autumn 2014

Contents

1 Introduction

This paper describes a solution for The Cup Collector project, and has been devised for the course Robots in Context. The goal is to implement 3 different parts of a robot system for picking up cups all over SDU-TEK.

In part 1, a method to scan all rooms and pick up cups is implemented. The route is planned offline using a map (image) of the facility.

In part 2, a floor sweeping mode is implemented in the robot. In this mode, the robot sweeps the floor of SDU-TEK.

For part 3, a method for localizing the robot is necessary. The used hardware is described and two methods of localisation are implemented and compared.

1.1 Code:

The final code is available at:

`https://github.com/niive12/CupCollector/tree/master/code/To%20report`

Animations of part 1 and 2 is also available on GitHub.

2 Planning

2.1 Method

The robot is modeled as a point robot by letting the configuration space be the freespace padded with obstacles, with padding length equal to the robot dynamics radius. This padding is obtained using the brushfire algorithm with a Norm-2 potential function (rounded corners), and then simply applying a threshold in the map for the padding.

2.1.1 Offline processing

As all information about the map (excluding the location of the cups) is known to the robot prior to movement, it was chosen to carry out much of the planning offline. A short overview of the approach is given below.

1. Generate map of brushfire values, $M_{B,0}$, from original image.
2. Generate a new image with closed doors, considered obstacles, using $M_{B,0}$.
3. Generate a map of brushfire values, $M_{B,D}$ from the image with closed doors.
4. Generate a set of coordinates, S_D, from $M_{B,D}$, such that the brushfire values of the coordinates in S_D are odd multiples of the robot scanner radius.
5. Generate a set of coordinates, S_L (initialized to S_D), by merging coordinates with local brushfire value maxima in $M_{B,D}$, with S_D, selecting only coordinates that are not within one robot scanner radius of any coordinate already in the set.
6. Generate a set of coordinates, S_M (initialized to S_L), of all the coordinates that need to be scanned to guarantee detection of all cups, by adding from the freespace all remaining coordinates that are not within one robot scanner radius of any coordinate already in S_M.

(a) Brushfire based coverage

(b) Points that were not covered by wall distance

(c) List formed by merging the two list

Figure 2.1: Offline planning

To detect all cups, the entire map must be scanned. In other words, there is a minimum number of coordinates for the robot to cover. Several sets of coordinates satisfy the criterion that the entire map would be scanned if they were each visited. Of course, setting this set to hold all freespace coordinates would not make for very good off-line planning. Therefore, it was chosen to use a map, $M_{B,0}$, generated by the brushfire algorithm, mapping a brushfire value to each coordinate, to generate the set of coordinates.

We denote all coordinates with brushfire values equal to odd multiples of the robot scanner radius S_B. These coordinates is shown in figure 2.1a. S_B consists of lines of points, with all points on a given line having the same distance to the walls. If the robot always moves to the nearest point in S_B, it would follow these lines and cover most of the map. However, large rooms (with large doorways and local lines inside) would be entered multiple times, including the long

and relatively narrow hallways. This is neither very pedagogical or practical, considering the interruptions created from the need to open doors, and the extra length that would be traveled in hallways, compared to if each room was visited only once. Therefore, it was chosen to create a different map, $M_{B,D}$, of brushfire values from the idea that all doors are closed and function as obstacles. Selecting coordinates from $M_{B,D}$ according to the same criterion as the one used for $M_{B,0}$, a different set, S_D, of coordinates to visit is obtained. S_D is similiar to S_B, but with an important difference: There are no lines going through doorways. With this set of coordinates, if the robot always moves to the closest point, it will still stick to the lines while, this time, staying inside the rooms until all the local lines have been visited.

S_D is not the complete set of coordinates the robot needs to visit. Some local brushfire value maxima in the centres of rooms would need to be scanned, but are not in S_D, because they are not odd multiples of the robot scanner radius. Of course, these local maxima are easy to find, using $M_{B,D}$, and the coordinates are merged with S_D, creating a new set, S_L. The coordinates are shown in figure 2.1b. The coordinates are merged into S_L one at a time, and are only chosen if no coordinates of S_L are (already) within one robot scanner radius. The final list is shown in figure 2.1c.

S_L consists of almost all points that need to be scanned in order to detect all cups. Only a few coordinates of freespace remain to be scanned. These are merged with S_L into a new set, S_M. They are added much in the same way as the local brushfire value maxima: The entire freespace is searched for coordinates that are not close to any coordinates in S_M, and these are added to S_M until no more coordinates remain to be scanned. Thus, S_M is the complete set of coordinates that need to be scanned to detect all cups.

The robot will need to visit the offloading stations when its cup capacity is reached, and it should reach these stations quickly. Since the offloading stations are stationary, a map, M_{OL} is created, holding values generated by Dijkstra's Shortest Path with Weighted Costs algorithm, using a Norm-2 potential function for the wave propagation from the offloading stations.

2.1.2 Online processing

Each time the robot moves one step (from one coordinate to another), it needs to do three things: Scan, pick up cups within reach, decide whether or not to move to the offloading stations. The coordinates that need to be scanned are all in the set S_M. Thus, to ensure the entire map is scanned, the robot only needs to visit all coordinates of S_M. The scan range is shown as the red circle in figure 2.2a. Optimally, the route taken through these coordinates is as short as possible, making this a case of the Traveling Salesman Problem. To keep the number of computations for calculating the route low, it was chosen to find a suboptimal solution to the problem. For each coordinate visited in S_M, the nearest neighbour strategy is used to select the next coordinate in S_M to visit. This movement is simple to code, as illustrated by below pseudo-code:

```
while( S_M is not empty ){
   c = Nearest coordinate in S_M;
   s = Shortest path to c;
   for( each coordinate i in s ){
      Move robot to i;
      Scan with robot scanner centered at i.
      Remove i from S_M;
   }
}
```

The above pseudo-code handles the scanning of the entire map. The picking up of cups within reach can easily be added after the scan, but what about the cups that are found by the scanner, yet are unreachable? To keep the code simple, these cups are handled indirectly: They (or configuration space coordinates close to them) are added to S_M: Figure 2.2a shows a cup that is unreachable. By keeping to the nearest neighbour the robot will move to the left. In figure 2.2b the robot is in a position where the cup is reachable and the coordinate is removed from S_M. Following this logic the map is covered as shown in figure 2.2c.

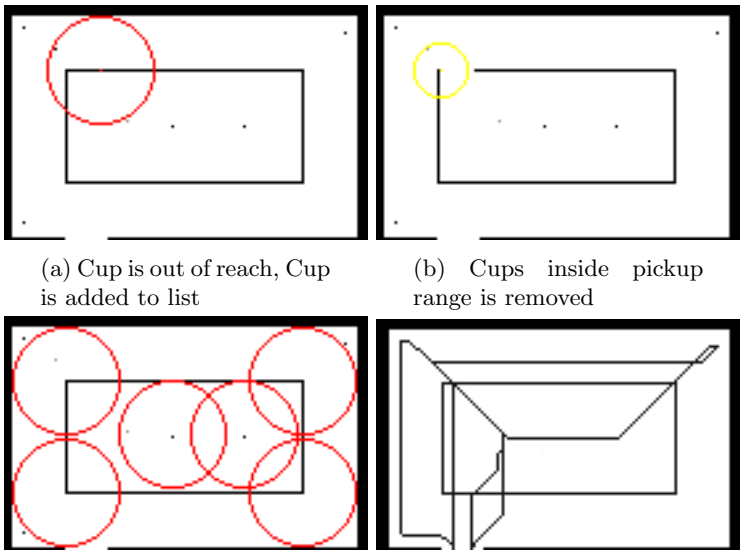
```
while( S_M is not empty ){
   c = Nearest coordinate in S_M;
   s = Shortest path to c;
   for( each coordinate i in s ){
      Move robot to i;
      Scan with robot scanner centered at i;
      for( each cup coordinate k in the circle ){
         if( k is reachable ){
            Pick up the cup at k;
         }
         else{
            Add k's nearest configuration space coordinate to S_M;
         }
      }
      Remove i from S_M;
   }
}
```

Now, all that's left is handling the situation where the robot reaches its cup capacity. It was chosen to make the necessary modification to the above pseudo-code as simple as possible: If the robot reaches its cup capacity, it returns to the nearest offloading station coordinate, taking the shortest path available, scanning for cups on the way. When the offloading station is reached, the robot returns to the nearest-neighbour strategy (it does not return the way it came). The modification, concluding the robot movement logic, is indicated in the pseudo-code below.

```
while( S_M is not empty ){
   c = Nearest coordinate in S_M;
   s = Shortest path to c;
   for( each coordinate i in s ){
      Move robot to i;
      Scan with robot scanner centered at i;
      for( each cup coordinate k in the circle ){
         if( k is reachable ){
            Pick up the cup at k;
            if( cup capacity reached ){
               z = Shortest path to offloading station (using M_OL);
               for( each coordinate j in z ){
                  Move robot to j;
                  Remove j from S_M;
                  Scan with robot scanner centered at j;
                  for( each cup coordinate g in the circle ){
                     Add g's nearest configuration space coordinate to S_M;
```

```
                    }
                }
            }
        }
        else{
            Add k's nearest configuration space coordinate to S_M;
        }
    }
    Remove i from S_M;
  }
}
```

The selection of the nearest neighbour is performed using the wavefront algorithm, propagating a wave around the robot until a coordinate in S_M is found. The robot movement is shown in figure 2.2d.



(a) Cup is out of reach, Cup is added to list

(b) Cups inside pickup range is removed

(c) Coverage in critical areas

(d) The traveled path

Figure 2.2: Online planning

2.1.3 Door detection

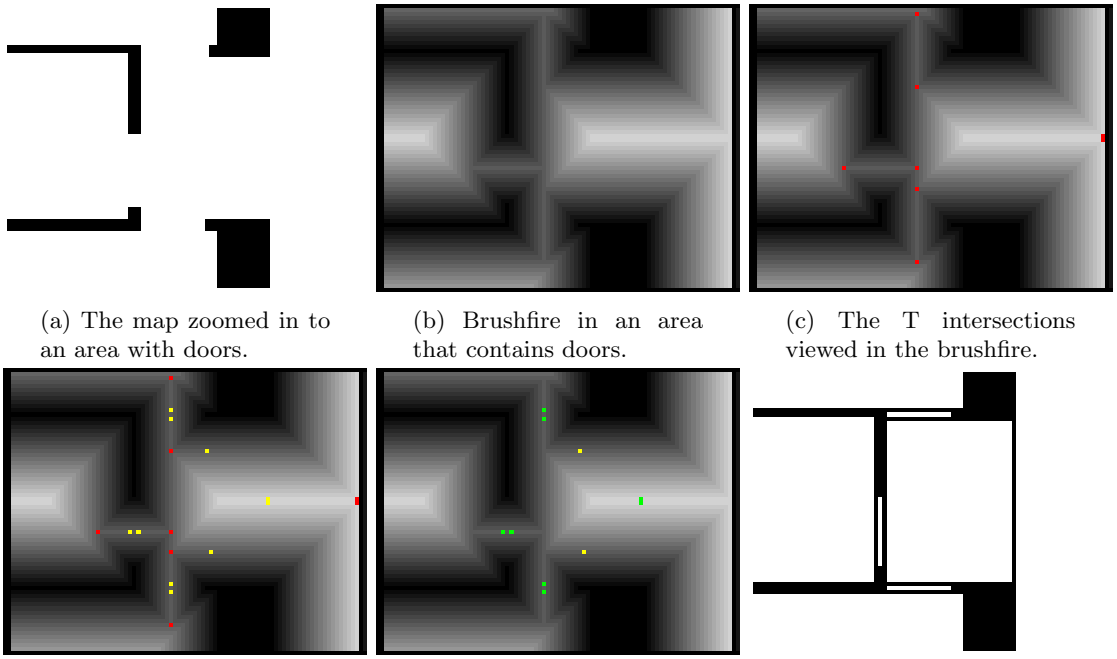
In order to isolate every room each door is detected and drawn onto a pixelshademap (a grayscale image object).

To locate a door a brushfire map (an object containing brushfire values for all coordinates) is created. In figure 2.3 the stages of detection is shown for an area that contains three doors. To better illustrate this detection each brushfire value is multiplied by ten.

In figure 2.3c the door detection is going through the brushfire map and looking for where two waves meet. When the waves meet a T section is formed and this is detectable in the map. In the direction of the T there is potentially a door as shown in figure 2.3d

In figure 2.3e the list of doors is narrowed down by checking if they have an obstacle on both sides in the distance correlating to the brushfire value.

To separate the rooms every door step is painted from the door until it meets an obstacle as shown in figure 2.3f.



(a) The map zoomed in to an area with doors.

(b) Brushfire in an area that contains doors.

(c) The T intersections viewed in the brushfire.

(d) Possible doors in relation to the T sections.

(e) The correct doors.

(f) The door steps.

Figure 2.3: Door detection step by step.

2.2 Results

The robot scans the entire freespace, detecting all 287 cups. All cups are picked up and returned to the offloading stations. The total traveled path length is 37294 m, corresponding to a travel time of 7 hours and 28 minutes.

The offline processing taking place before the robot begins its movement takes 40 seconds to execute, while the robot movement code takes 9 minutes and 57 seconds to execute. Timing was measured on a Intel Core i5-3230M CPU with a clock frequency of 3.2GHz.

2.3 Conclusion

Collecting all the cups takes 7 hours and 28 minutes.

2.4 Future work

It is likely that the generated set of coordinates, S_M, that the robot must scan, is not optimally generated. Future work could look into alternatives to the coverage method used here. For example, it might be logical to make the robot stick to areas such as hallways until all the cups in those areas are collected. This way, the robot wouldn't have to return to the same hallway as many times.

The robot movement solves a Traveling Salesman Problem suboptimally, using the nearest-neighbour strategy. This strategy is used because of it's computational efficiency. However, if more computational power is available, another, more optimal, strategy may be applied to solving the problem better.

3 Coverage

3.1 Method

3.1.1 Offline processing

It was chosen to apply the offline processing strategy of the cup collection to the floor sweeping, as it's essentially a coverage problem. The only change is in the radius used in the generation of the coordinate set; Here, the robot dynamics radius is used instead of the robot scanner radius. The set of coordinates to sweep is denoted S_F.

3.1.2 Online processing

The floor sweeping problem is simpler than the cup collection problem, and the chosen solution is similiar to what is discussed for the cup collection. The pseudo-code below shows the floor sweeping algorithm.

```
while(S_F is not empty ){
   c = Nearest coordinate in S_F;
   s = Shortest path to c;
   for( each coordinate i in s ){
      Move robot to i;
      Remove i from S_F;
   }
}
```

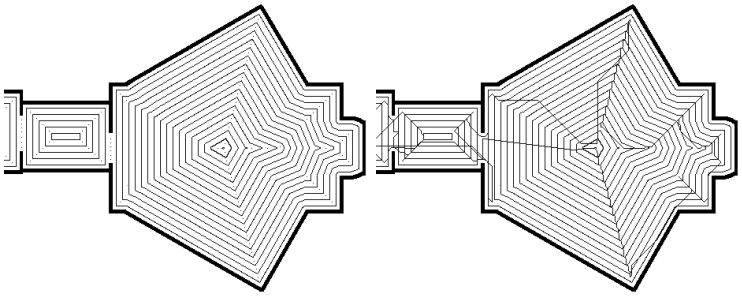
Clearly, this code is highly reliant upon an intelligent offline selection of coordinates for S_F.

3.2 Results

The robot sweeps all reachable floor space. The total traveled path length is 97449.7 m, corresponding to a travel time of 19 hours and 30 minutes.

Figure 3.1 shows part of the offline generated map vs. the traveled path in the same part of the map.

The offline planning part of the code takes 19.201 seconds to execute while the robot movement code takes 2 minutes and 10 seconds to complete the coverage. Timing was measured on a Intel Core i5-3230M CPU with a clock frequency of 3.2GHz.



(a) Part of the offline generated map for floor sweeping. The coordinates in S_F are marked.

(b) Part of the traveled path in floor sweeping. The coordinates visited are marked.

Figure 3.1: Floor sweeping planned vs. traveled.

3.3 Conclusion

The same approach to solving the coverage problem was used as described in chapter 2. This guarantees a total coverage of the school when cleaning it. This approach resulted in the robot taking 19 hours and 30 minutes to sweep the whole floor.

3.4 Future work

As was stated in section 2.4, future work could look into alternative coverage methods and better, computationally more complex, solutions to the Traveling Salesman Problem, than the nearest-neighbour strategy applied here.

4 Localisation

Mobile robots navigating in accordance to a map, need localisation. There are various localisation methods and sensors for mobile robots. This section describes how a localisation method was implemented with data from the robot's encoders and its laser range scanner. Due to time constraints, the method was implemented as an offline program on a PC, using data gathered by a Nexus Robot, instead of as an online program running on the Nexus Robot as it moved.

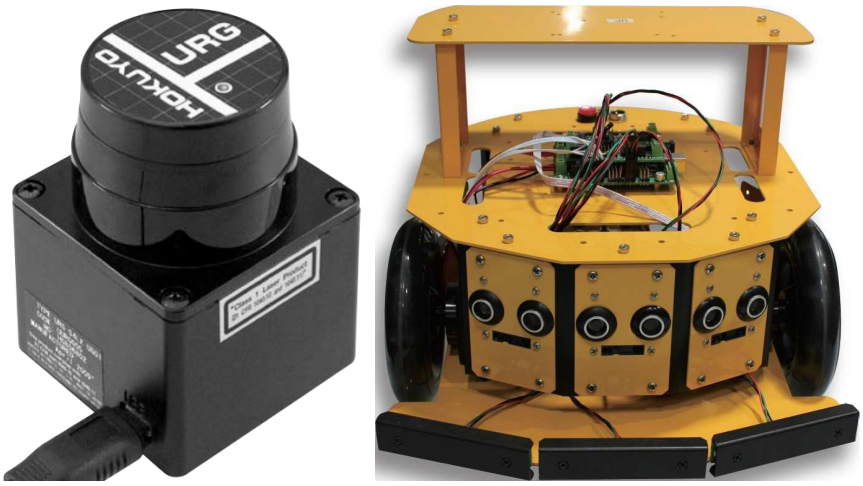
4.1 Method

For the localisation part of the project it was chosen to compare two different approaches:

- Position estimate based on odometry (wheel encoders).
- Position estimate based on line features extracted from the laser range scanner.

Both of these approaches rely on a known starting location. The second also needs knowledge of the environment - it must compare the extracted features with probable features from the map (based on last know location of the robot).

The mobile robot used is a Nexus Robot - 2WD mobile robot kit 10004 (figure 4.1b), fitted with a laser range scanner of type Hokuyo URG-04LX-LG (figure 4.1a).



(a) Hokuyo URG-04LX-LG laser range scanner.

(b) Nexus Robot - 2WD mobile robot kit 10004.

Figure 4.1: Hardware used

4.1.1 Gathering data using UMBmark

The performance of the localisation technique was measured using UMBmark. The position before and after following a preprogrammed path through a 1 x 1 [m] square was compared for the different localisation techniques. While testing, the robot was not given any feedback from the sensors. After each 90° turn, the robot stopped and its position on the test track was measured for reference. Sensor readings from both the encoders and the laser range scanner were gathered and saved for further offline processing. This means that 3 different measurements of the position were gathered:

- Reference position (measured with tape).
- Position based on encoder feedback.
- Position based on feature extraction using the laser range scanner.

4.1.2 Position based on odometry

The position/state Q of the robot is represented by the vector:

$$Q_r = \begin{bmatrix} x_r \\ y_r \\ \alpha_r \end{bmatrix} \tag{4.1}$$

The robots position is estimated from the previous position, plus the movement in the last time period Δt. The change in position ΔQ_r from each timestep is added to the position. This can be written as

$$Q_r\text{'} = Q_r + \Delta Q_r \tag{4.2}$$

The feedback from the robot is given as the movement of each wheel in millimetres. ΔQ_r can be found from the following equations:

$$\Delta x_r = \Delta s \cos\left(\alpha_r + \frac{\Delta \alpha_r}{2}\right) \tag{4.3}$$

$$\Delta y_r = \Delta s \sin\left(\alpha_r + \frac{\Delta \alpha_r}{2}\right) \tag{4.4}$$

$$\Delta \alpha_r = \frac{\Delta s_r + \Delta s_l}{b} \tag{4.5}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{4.6}$$

$$\tag{4.7}$$

Where $\Delta s_r and \Delta s_l$ are the distances traveled by the right and the left wheels and b is the distance between the robots two wheels. Adding this together, the equation for the updated position is:

$$Q_r\text{'} = \begin{bmatrix} x_r \\ y_r \\ \alpha_r \end{bmatrix} + \begin{bmatrix} \Delta s \cos\left(\alpha_r + \frac{\Delta \alpha_r}{2}\right) \\ \Delta s \sin\left(\alpha_r + \frac{\Delta \alpha_r}{2}\right) \\ \frac{\Delta s_r + \Delta s_l}{b} \end{bmatrix} \tag{4.8}$$

4.1.3 Getting data from the laser range scanner

The Hokuyo laser scanner measures distance to the nearest obstacles in a 240° range as shown in figure 4.2. The angular resolution is 0.352° which gives 681 data points for each scan. The scanning rate is 10 Hz.

The laser range scanner communicates with the computer using Serial over USB. The serial communication uses the SCIP 2.0 serial protocol. A basic application for gathering data from the laser range was written in C++[1]. The program gathers and decodes the data received, and saves it as a CSV file. The program stops the laser range scanner when the UMBmark test is terminated.

[1]The program is called "hokuyo_laserrange" in the included "code" directory

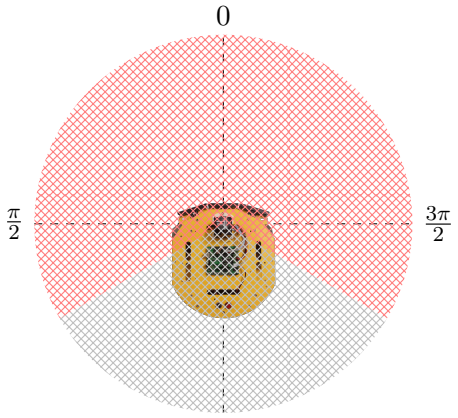


Figure 4.2: The area covered by the laser range scanner.

4.1.4 Feature Detection

To detect the features of the robots whereabouts it was decided to implement the RanSaC algorithm. During tests of this implementation, it was found that RanSaC had tendencies to detect two similar lines next to each other for the same features, when this feature was represented by many points in the sample. Therefore, it was decided to implement the RanSaC with a merge function. The angle and distance to the two lines representing the same feature were then averaged to give one line, represented by a polar vector to the point on the line that is closest to the origin.

It was decided to represent all lines as polar lines, (R, θ), where R is the shortest distance to the line from the origin and θ is the angle to the x-axis. For simplicity, all lines are standardized to conform with $0 \leq \theta < 2\pi$ and $R \geq 0$. The modified version of RanSaC was implemented as follows:

1. Copy the data from the 2D laser scanner into a list, Λ_{points} , of polar points (R,θ), excluding all points with $R = 0$ which correspond to points not visible to the scanner.
2. Copy two unique random points into Λ_{sample} and find the line, λ_{init}, passing through them.
3. Copy all points within $\Delta_{maxdeviation}$ distance to λ_{init} into Λ_{sample}. If the size of Λ_{sample} passes a minimum threshold, go to item 4 else go to 2.
4. Delete all points from Λ_{points}) within $\Delta_{maxdeviation}$ from λ_{init}.
5. Find the best fit line, λ_{final}, passing through the dataset Λ_{sample}. Place λ_{final} in the final line set.
6. Until a specific number of RanSaC-loops are done without finding a line or if the dataset is too small to contain a valid line, go to item 2, else continue.
7. Standardize the lines found to fit the specified format and merge lines considered to represent the same feature.

The data points form the laser range scanner which were found to have $R = 0$ were removed because they were, defined by the scanner, points for which the object is out of reach. The parameters for the RanSaC were experimentally, through trial-and-error, found to fit the robots collected data points as good as possible.

4.1.5 Position based on features from laser range scanner

The state of the robot is in polar coordinates represented as

$$Q_r = \begin{pmatrix} R_r \\ \theta_r \\ \alpha_r \end{pmatrix}$$

and in cartesian coordinates as

$$Q_r = \begin{pmatrix} x_r \\ y_r \\ \alpha_r \end{pmatrix}$$

When the localisation program starts, it has a map of the room the robot is in. The map consists four walls, each represented as

$$wall_n = \begin{pmatrix} R_n \\ \theta_n \end{pmatrix}$$

The initial position of the robot is also known on startup.

The localisation algorithm is illustrated as a flowchart in figure 4.3. An illustration of the global coordinate system and the robot's coordinate system can be seen in figure 4.4.

Mapping The found lines are all represented in the robot's coordinate system, relative to the robot position and orientation. These lines are mapped into the global coordinate system, using the last known robot state. Using the last known state of the robot to map the lines will give some mapping-error. The found lines are then compared to the map. If a found line matches a line in the map, within a certain threshold, it is concluded, that they are in reality the same line and that the difference is due to the mapping error.

Updating robot state There are several scenarios of what the program will do after the matching of lines, depending on the lines that have been matched. The philosophy of the algorithm is that the lines should be used as much as possible, to determine the robot state.

If no lines have been matched, the robot state remains the same, and is not updated.

If only one line has been matched, the robot orientation α_r will be updated so that the found line aligns with the known line in the global map. The same strategy is used if there are several matched lines, and the new α_r is found by averaging the individual orientations.

If two intersecting lines have been matched (and a corner found), the robots position is constrained to an intersection between two lines. Again, finding the intersection is a matter of moving the global lines towards the origin by the distance from the robot to the walls, and then finding the intersection between the two lines. If more than two intersecting lines are matched, the same strategy is used and the new position is calculated as the average of all the found positions.

Thus, as long as at least one line is matched, the robot state is updated. The update correctness should improve with the number of lines matched, if the lines are not parallel.

The weakness of the algorithm lies in missed data points, or data points with no lines detected by RanSaC. The further the robot actually moves from the last updated position, the harder

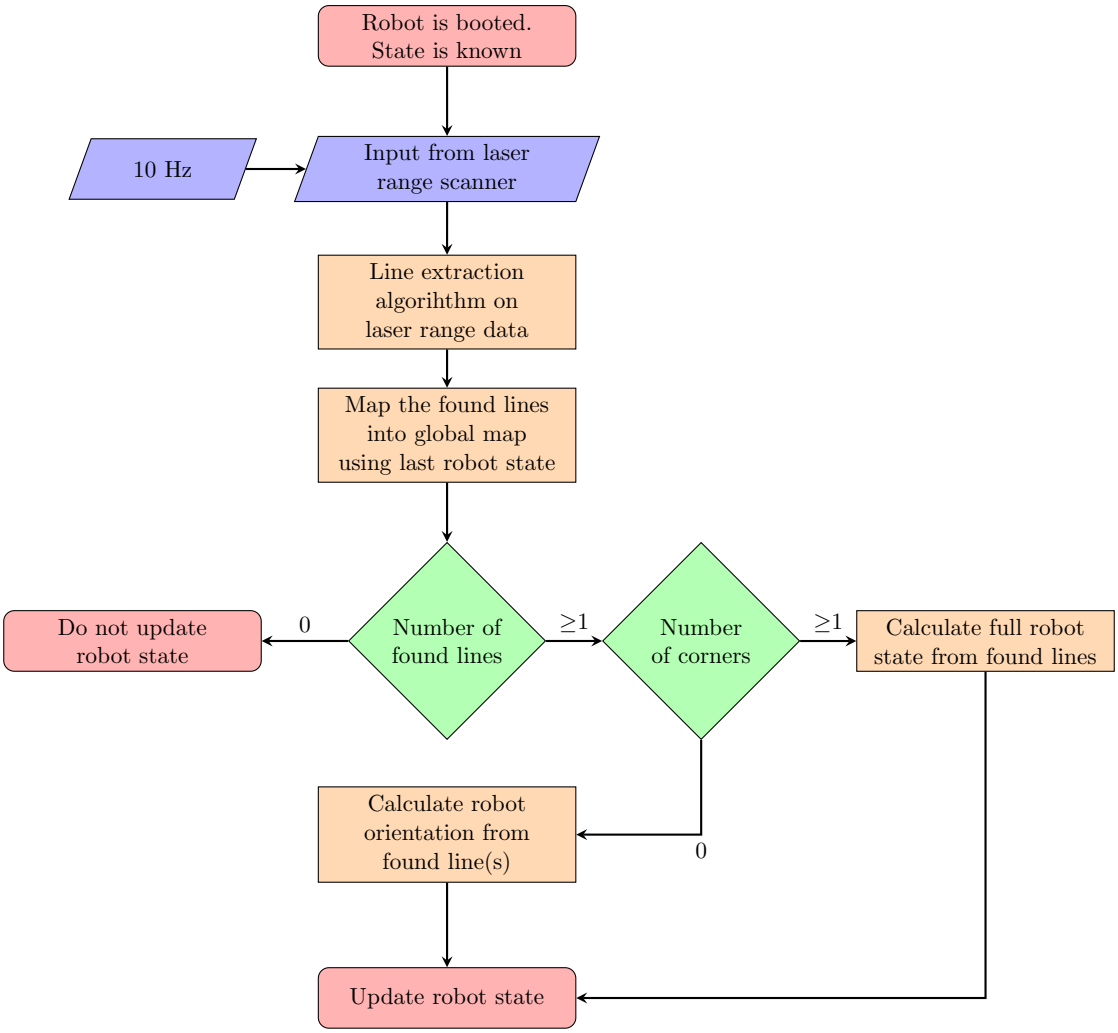


Figure 4.3: The localisation method, purely based on data from laser range scanner.

it is to match the line feature to the global map, and eventually the lines would be matched incorrectly.

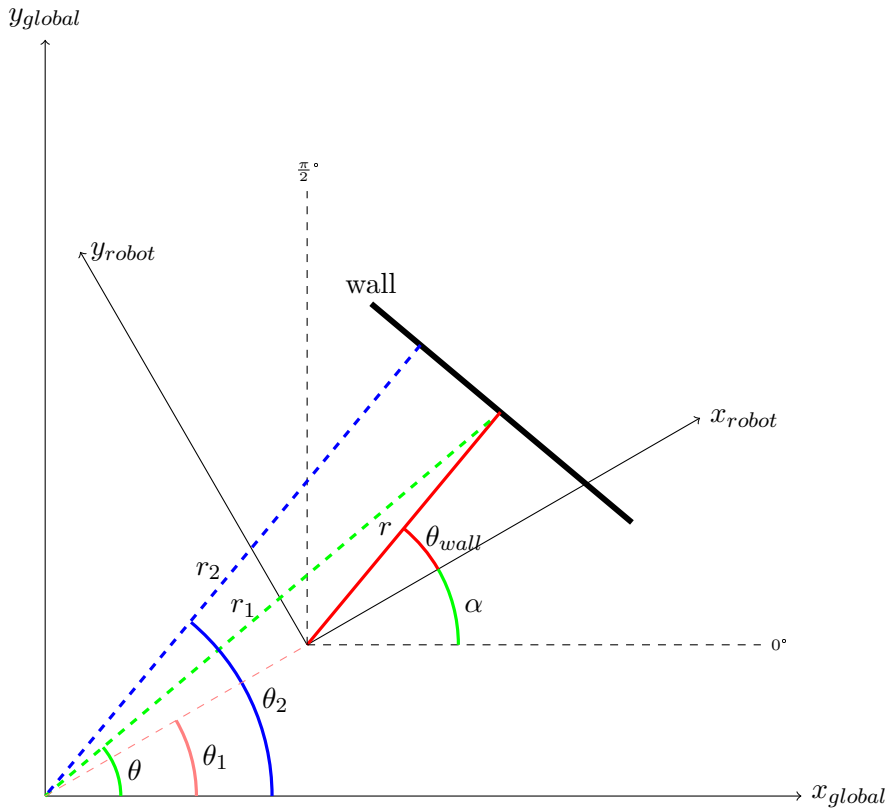


Figure 4.4: Illustration of the global coordinate system and the robot's coordinate system

4.2 Results

The data from the UMBmark test were used to plot three robot paths. The points at which the robot stopped to turn were marked on the floor, and were used as reference points. The laser range scanner and odometry readings were sent to a PC, and were processed using the aforementioned algorithms. A problem occured in the test: some laser range scanner readings were lost, possibly due to partial incompatibility between the PC's OS and the serial communication library used.

However, the line-based localisation algorithm was able to match the features correctly for several rounds, and the results have been plotted in figure 4.5.

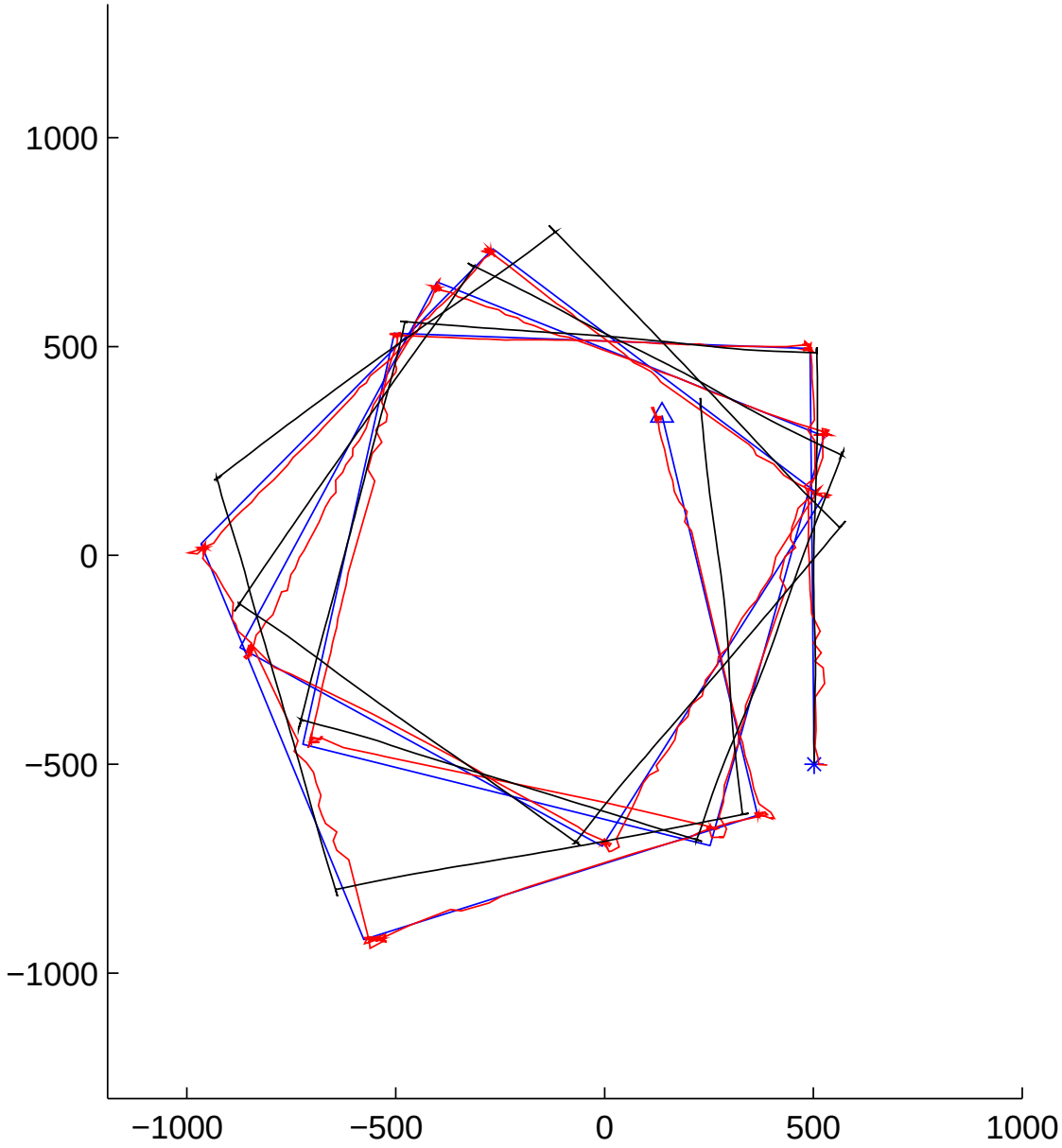


Figure 4.5: Comparison of the two localization methods. The blue line connects the reference points where the robot paused to turn, $*$ is the starting position and Δ is the ending position. The red line represents the 2D laser range scanner localisation and the black line represents the odometry readings.

It is easily seen that the line-based localisation algorithm is better than odometry alone. The line-based localisation seems to move in distorted lines, and this indicates the actual movement better than the reference lines; the reference lines are just straight lines drawn between the reference points.

The results were expected, as the odometry method accumulates error over time, and the line-based method does not.

The deviations of the two models for calculating the position of the robot are seen in figure 4.6. The figure shows the deviation of the robot at the reference points, where the robot paused.

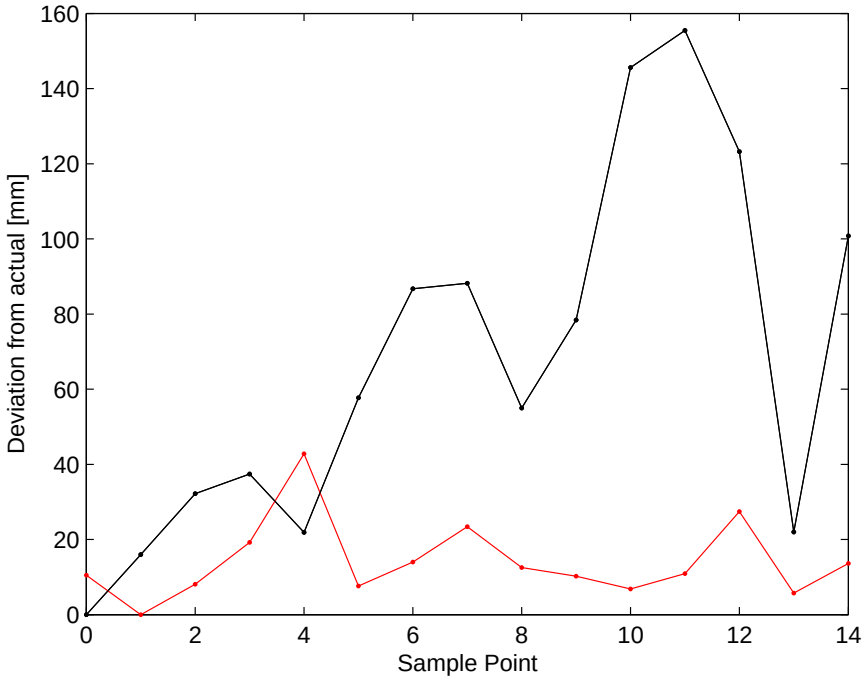


Figure 4.6: Deviation in position of the two localization methods. The red line represents the laser range scanner localisation, and the black line represents the odometry readings.

From figure 4.6 it can be seen that the error accumulates over time when using only odometry to measure the position. The data from the laser range scanner, however, are used to calculate the position considerably better, with a much more steady deviation, than the odometry readings.

When plotting the absolute deviation of the robots angle at the previous used points figure 4.7 is obtained.

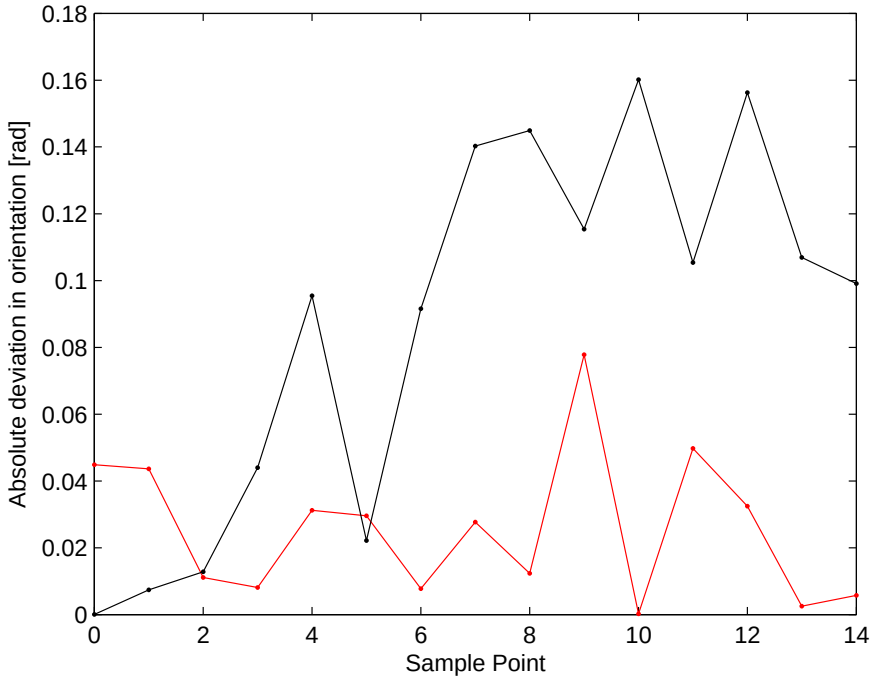


Figure 4.7: Deviation in orientation of the two localization methods. The red line represents the laser range scanner localisation, and the black line represents the odometry readings.

It be seen from figure 4.7 that the deviation in orientation, like the position, accumulates over time. For short distances and few turns, the odometry is better than or equal to that of the average laser scanner readings. This advantage does, however, cease when the robot reaches its fourth turn and onwards it is considerably higher.

It can hence be concluded that the laser scanner is considerably better when driving for longer time and only one method of localisation has to be used. The combination of using both the laser scanner and odometry together is, however, recommendable. Odometry alone is not suitable to stand alone, but they can be used efficiently in combination to achieve accurate localization of a robot on a map. It should still be noted that odometry can be used to accurately decide the robots position and angle when travelling short distances.

4.3 Conclusion

Two offline localisation methods were implemeted on a PC: One based on odometry alone, and one based on laser range scanner line feature extraction alone.

A UMBmark test showed that the line-based method worked better in practice, as it did not accumulate errors over time. The line-based localisation method was able to accurately determine the robot position and orientation.

4.4 Future work

Future work should focus on online implementations of the algorithms. This would allow the robot to actually use its position and orientation in planning its movements.

The localisation algorithm has much room for improvement. The first step would be to include odometry readings in the calculation of the next step. One simple way of expanding the algorithm is to use odometry only if the laser range data are not present or sufficient to update the robot state.

As both odometry and laser range scanner readings were collected, the optimal localisation method should combine them using a Kalman filter. Here, it would be beneficial to include the robots linear and angular velocities/accelerations to predict the robots movement.

5 Conclusion

A method for finding and picking up cups all over SDU-TEK was implemented. The method used off-line planning to determine a route for scanning the whole building. This was done using a door detection algorithm to add doors to the map and brushfire to generate a set of coordinates to be visited by the robot. The collection of cups takes 7 hours and 28 minutes.

A coverage for sweeping the floors was also developed. It was based on the same strategy as the first part, but without the ability to scan for cups. The robot is able to sweep the floors in 19 hours and 30 minutes. The offline planning runs in 30 seconds.

Two offline localisation methods were implemeted: One based on odometry alone, and one based on laser range scanner line feature extraction alone.

The line-based localisation method was able to accurately determine the robot position and orientation, and was superior to odometry-based localisation alone.

Is it feasible to buy the Cup Collector instead of having a man sweeping the floors? Probably not: assuming a man works 2 hours a day 365 days a year, this amounts to 20 440€, and the Cup Collector is 80 000€ excluding service. So if the robot works out of the box (highly unlikely with a sophisticated robot) the payback time is 4 years.