University of Bristol

COMSM0012 Robotics Systems PG

# Coursework 1

*Team name: CCLS-1*

| *Members:* | *Contribution:* |
|---|---|
| Luis Diaz [ld15325] | 25% |
| Carlos Pino [cp15437] | 25% |
| Suppanat Ruangdech [sr15417] | 25% |
| Christos Sevastopoulos[cs15427] | 25% |

April 19, 2016

## Contents

## 1 Introduction

It is common knowledge that navigation and localization constitute a highly important part of robotics. In this coursework, the functionality of the robot is determined by the use of an NXT Mindstorms microcontroller along with an NXT servo-motor and an Ultrasonic sensor. The movement and the control of the robot is accomplished using MATLAB in conjunction with the Mindstorms NXT toolbox. Specifically, the project consists of the simulated and the actual implementation parts respectively.

As far as the localization of the robot is concerned, the Particle filter approach is employed. At the next stage, the path planning procedure is going to be achieved using the A* algorithm. Since the ground is prepared, the real implementation of the robot will demonstrate the robustness of our work.

## 2 Robot Building and Design

The building of our robot (Figure 1) was based on a custom approach which apart from two front wheels, a roller is used as an additional attachment. The rationale behind the building is illustrated in the following sentences. In particular , we placed two servo-motors behind each front wheel in order to simplify the control of the robot and a third servo-motor to control the sensor and thus prevent any possible collisions that might occur. On the other hand, the use of the roller acts as a technique to overcome the undesirable effect of friction. What is more, we had also to take under consideration the fact that the height of the arena's walls is approximately twenty centimeters and as a consequence the sensor should not be placed over that.

Furthermore, another vital advantage of our design is the fact that we have the ability to perform measurements using the sensor that are not dependent on the robot's motion. Particularly, odometry will not be a serious concern since less actuators are involved in our endeavor. As a result, this eventually leads to less energy consumption, less unnecessary error modeling and to a more accurate response from the sensor.

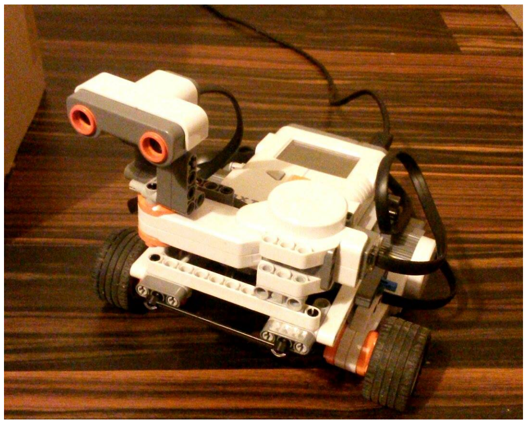
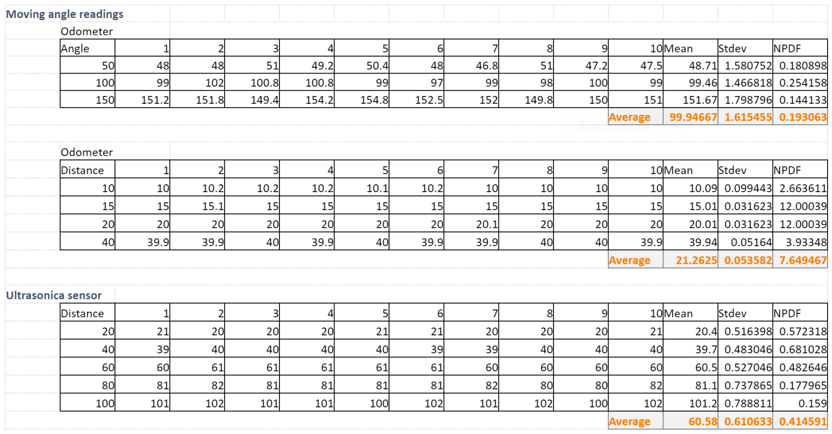


Figure 1: Robot Design.

## 3 Calibration and Sensor Modeling

As far as the calibration process is concerned, many sensor readings were taken in order to model the error. Therefore , we had the ability to compare the readings to the manual measurements taken with a metric tape. Furthermore, after placing the sensor in different orientations (using steps of 45 degrees). Finally, to model the odometry, the Gaussian distribution was used (Figure 2 and Figure 3).



**Moving angle readings**

Odometer

| Angle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Mean | Stdev | NPDF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 48 | 48 | 51 | 49.2 | 50.4 | 48 | 46.8 | 51 | 47.2 | 47.5 | 48.71 | 1.580752 | 0.180898 |
| 100 | 99 | 102 | 100.8 | 100.8 | 99 | 97 | 99 | 98 | 100 | 99 | 99.46 | 1.466818 | 0.254158 |
| 150 | 151.2 | 151.8 | 149.4 | 154.2 | 154.8 | 152.5 | 152 | 149.8 | 150 | 151 | 151.67 | 1.798796 | 0.144133 |
| | | | | | | | | | | Average | 99.94667 | 1.615455 | 0.193063 |

Odometer

| Distance | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Mean | Stdev | NPDF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 10.2 | 10.2 | 10.2 | 10.1 | 10.2 | 10 | 10 | 10 | 10 | 10.09 | 0.099443 | 2.663611 |
| 15 | 15 | 15.1 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15.01 | 0.031623 | 12.00039 |
| 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20.1 | 20 | 20 | 20 | 20.01 | 0.031623 | 12.00039 |
| 40 | 39.9 | 39.9 | 40 | 39.9 | 40 | 39.9 | 39.9 | 40 | 40 | 39.9 | 39.94 | 0.05164 | 3.93348 |
| | | | | | | | | | | Average | 21.2625 | 0.053582 | 7.649467 |

**Ultrasonica sensor**

| Distance | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Mean | Stdev | NPDF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 21 | 20 | 20 | 20 | 21 | 21 | 20 | 20 | 20 | 21 | 20.4 | 0.516398 | 0.572318 |
| 40 | 39 | 40 | 40 | 40 | 40 | 39 | 39 | 40 | 40 | 40 | 39.7 | 0.483046 | 0.681028 |
| 60 | 60 | 61 | 61 | 61 | 61 | 61 | 60 | 60 | 60 | 60 | 60.5 | 0.527046 | 0.482646 |
| 80 | 81 | 82 | 81 | 81 | 81 | 81 | 82 | 80 | 80 | 82 | 81.1 | 0.737865 | 0.177965 |
| 100 | 101 | 102 | 101 | 101 | 100 | 102 | 101 | 102 | 100 | 102 | 101.2 | 0.788811 | 0.159 |
| | | | | | | | | | | Average | 60.58 | 0.610633 | 0.414591 |

Figure 2: Calibration Data

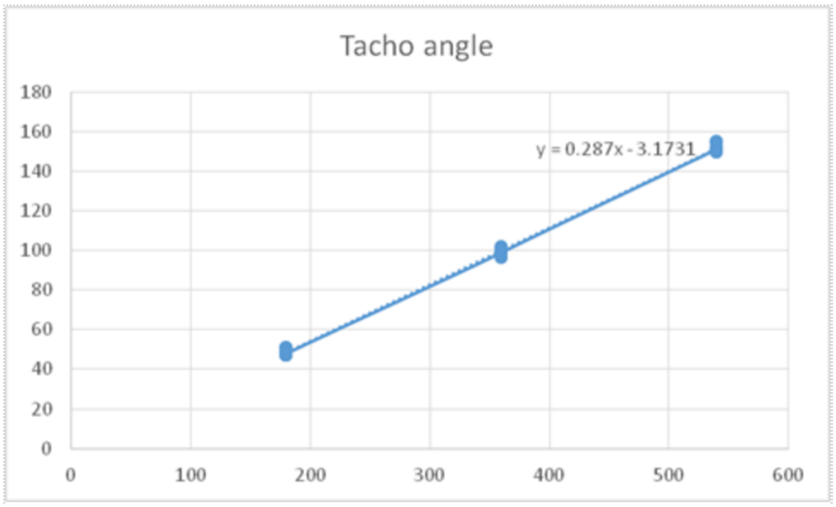


Figure 3: Turning Plot

## 4 Particle Filter

In the past decade, several localization algorithms were developed and along with the increase of the computers' performance , more probabilistic algorithms were robustly produced.

For starters, Particle Filter is a variant of the Bayes filter based on importance sampling. Its key idea concerns representing the posterior belief $bel(x_t)$ by a set of random state samples drawn from this posterior.

Algorithm 1 depicts the pseudo code of the particle filter implemented in MATLAB for both areas (simulation and real). In the next section we are going to describe in detail the approaches that were used in every stage of the algorithm.

### 4.1 MATLAB program

For this part, a template program containing the object oriented class "Botsim" and some examples illustrating the functionality of this were used. This class contains several useful functions such as move, turn, set position, get position, etc. The first step we did with this code was to make a copy of that object in order to split the robot from the particles. Therefore, in that manner, we can prevent "cheating" in the simulation. This new object is

**Algorithm 1** Particle filter

```
1: procedure Localization
2:     Generate random located particles
3:     while $converged = false$ do
4:         $Robot \leftarrow ultraScan.$
5:         $Particles \leftarrow ultraScan.$
6:         $ScoreParticles.$
7:         $Resampling.$
8:         $Convergence.$
9:         if $converged = true$ then
10:            goto $end.$
11:        $Respawn.$
12:        $Robot \leftarrow move.$
13:        $Particles \leftarrow move.$
14:    end.
```

called ParticleSim and two additional structures (setWeight and getWeight) were added to save the weighting of each particle. One of the main concerns of this program was to develop a simulation platform before proceeding to the real robot implementation.

**Particle Initiation** The first step in the particle filter is to initiate the particles around the map. We accomplished this step by obtaining the distances of the map's x and y coordinates respectively. Then the number of particles was generated according to these lengths. These particles are evenly separated from each other and facing at a zero degree angle. We can follow this approach because in our weighting function we are shifting the scan vector to find the best orientation (Figure 4).

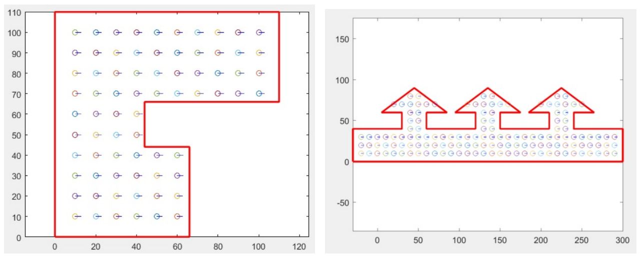


Figure 4: Particle initiation

**Scanning** To be able to locate the robot inside the map a sensor feedback is needed. Hence, an ultrasonic sensor was supplied for the NXT brick. The function "ultraScan" was used to rotate the sensor and obtain readings with equally spaced motor angle rotation. The default setting was 6 but in this approach we chose 8 scans per ultraScan in order to get a better estimation of the robot's position. UltraScan has to be made for the robot and for every particle.

**Scoring** Scoring is to assign some kind of score to each particle. Firstly, for each particle we calculated the difference against the robot in the 8 scans ( taken, but in order to be able to select the best orientation of the particles the circshift function of MATLAB which allows to shift the particles reading vector and save the best position was used. After having the best position of the particles we weighted every particle using the normal distribution formula:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} \tag{1}$$

where $x-\mu$ is the already best difference found of each particle. After weighing all the particles, the results were normalized.Th this way the sum of all the weights is always going to be equal to 1.In this step, after calculating the weight,it is vital to add the damping factor in order to avoid the accumulation of particles into one single point.

**Re-sampling** The approach taken to re-sample the particles was to save every particle weight and sort it from higher to lower. According to the normalized weighting we computed the number of particles that had to be re sampled. The more a particle weighs, the higher the number of particles associated to it ($weighting * numParticles$).

**Convergence** We proceed to calculate the standard deviation of the position of all particles, and when they met a certain value, we could say that we converged in the position of the first particle of the sorted weighted vector. To determine the angle we created a new particle in that position and ran the ultraScan with the findangle function to get the best angle. In this step we have already located the robot position if the standard deviation is met (Figure 5).

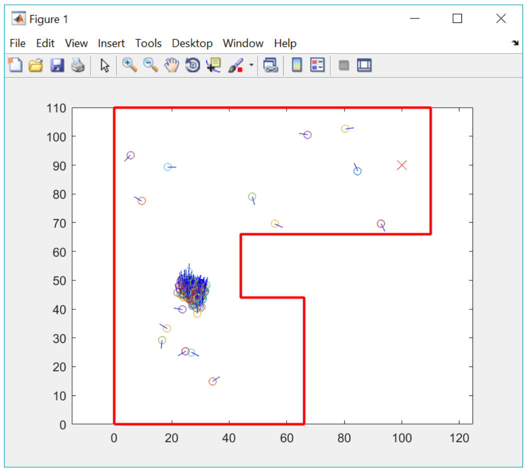


Figure 5: Convergence.

**Respawn** This is a very important step for the robustness of the algorithm, if we don't respawn the particles we can get stuck in one position (local optima) never having the change to find the correct location. In our approach we are respawning 1 every 5 particles in random locations.

**Movement** In order to accomplish the particle filter we need to move the robot in an exploratory way, giving more probability to find the correct position. All the particles must move the same way as the robot. In our program we made the exploratory movement as follows:

- Check from the first scan the distance to the wall if is less than 10 cm, get the reading from the left and right side. The robot it is going to turn to the greater distance between them.
- After checking step number one the robot will move forward 5 cm at a time.

For the particles we make the same movement but we check if with the movement the particle continue inside the map, if not that particle gets respawn in a random location.

## 5 Path Planning

At the beginning a map representing the spatial data in which the robot's motion is going to take place is required. However, additional attention should be placed on the fact that the robot's interaction with the map's edges might affect the completeness of the motion planning. Subsequently, we applied a reduction of the map's dimensions (Figure 6). Once this process was finalized, a representation of the map as a graph was developed know as the visibility graph. Additionally, this visibility graph was constructed using the corners of the map and the target point as nodes.As a last step, in order to make the visibility graph complete, a basic prerequisite was to add the localization point as a node (once the robot was localized).Finally, that visibility graph was used as a input to the A* algorithm, as seen in Figure 7.

Afterwards, the A* algorithm partitions the map to several nodes. Since the map is partitioned , our algorithm generation a motion command that will guide our the robot through the least-cost path.

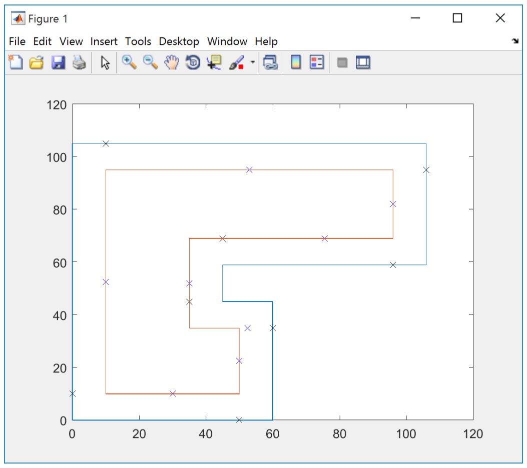


Figure 6: Reduced Map.

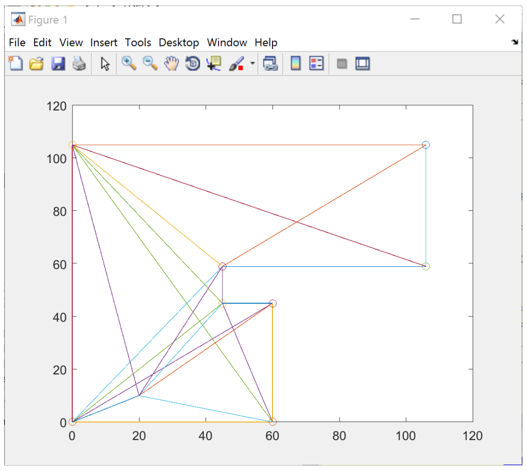


Figure 7: Visibility Graph.

## 6 Real Robot Implementation Challenges

### 6.1 Inaccuracy with Ultrasonic Readings

Initially, the primary challenge lies on the perpetual improvement of the robot's speed and the precision of the sensor readings. As a consequence, many different designs were implemented until a certain level of precision and speed standards was reached.

In particular, a major challenge that needed to be overcome was the misreading of the sensor when the robot's orientation was forming an angle of 45 degrees with wall.Hence, there was a disparity between the sensor's and the particle filter readings respectively. Nevertheless, a way to tackle this was to apply a continuous (instead of a partial) motion of the sensor during which, readings were received every single degree. This was proven advantageous since a direct improvement of the localization was observed (Figure 8).

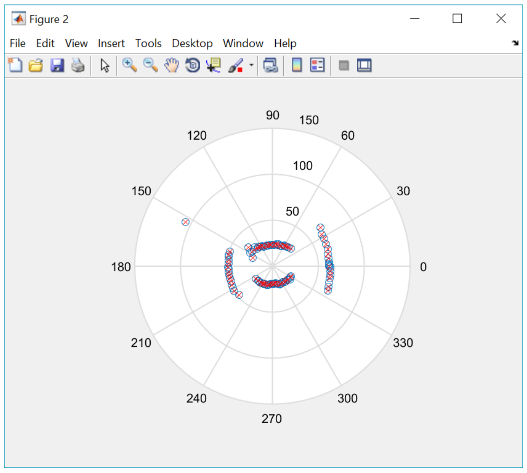


Figure 8: Robot Scan.

## 6.2 Particle Spread Method

Primarily, a specific challenge lies on decreasing the time of convergence of the particle filter. That challenge is strongly related to the initialization of the map and the initial distribution of the particles. Our approach towards dealing with that challenge was to modify the random pattern in which the particles were spread to a more coordinated one such as a rectangular grid. Therefore, we observed a better re sampling of the particles (Figure 9).

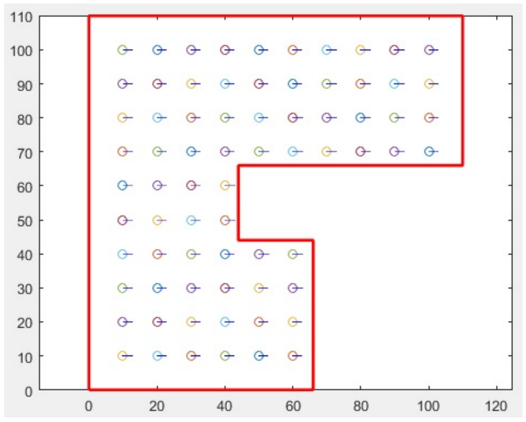


Figure 9: Particles Grid.

## 6.3 Collision Protection during the Path Planning

In general, once the robot is efficiently localized a path planning procedure has to be executed without further measurements.Since the localization is not always exact, a unreliable representation of the map might be deduced which could lead to non welcome collisions. However, in our approach we proposed an additional reading as a preventative way within the path planning stage. Therefore, a clearer navigation is ready to be realized.

## 6.4 Splitting the map into subsections

Originally,since the map includes boundaries and edges that could affect the robot's exploration , the path produced from the localization point to the target point might be dubious.Hence, we split the map to subsections and in each of them we identified an anchor point which acts as a safe place for the robot to move.Instead of determining a path from the localization to the target point , we created a path among the map's zones.As a result , this ensures a safer navigation of the robot.