Assignment 2 – Robot Localization in a Known Environment

Introduction:

This report details the design and implementation of a mobile robot whose task was to accurately estimate its position within a 2D map where the environment was known. The selected "Pioneer" mobile robot and its surrounding map were given using the MobileSim simulator software. The behavior of the "Pioneer" robot was implemented using the Aria library, and an occupancy grid map was supplied and displayed using the SFML C++ library in the Visual Studio IDE. The objective of the application was to design a C++ solution to allow the robot to navigate the created scene and generate an accurate representation of its position within the 2D map using the Monte Carlo Localization method. The solution was realized by taking elements of the work from previously completed labs including a previous implementation of the Monte Carlo Localization algorithm (Particle Filter) within a 1D map. To test the quality of the solution, a number of tests were run within the MobileSim environment.

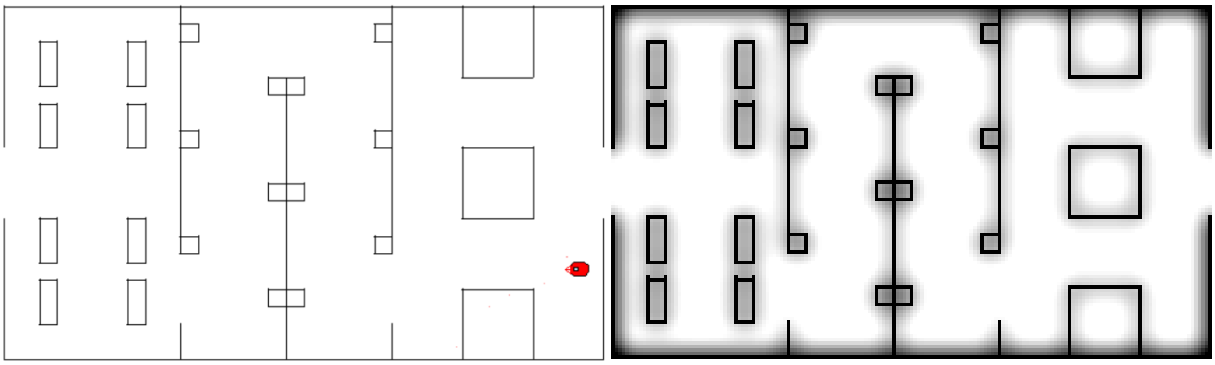

Figure 1 – *MobileSim Simulation Environment & associated Occupancy Grid Map*

Localization Technique:

Monte Carlo Localization:

The Monte Carlo Localization technique, which originates from Particle Filter theory, is a method of localizing a robot within a known environment. The objective of this technique is to iteratively estimate the position and orientation of the robot within its environment on the assumption that the coordinates of the environment are known. This is achieved by utilizing the change in the robots motion $\boldsymbol{U_t}$ and the measured sensor values $\boldsymbol{Z_t}$ of the robot as it travels. The starting point of the algorithm is a set of particles at the previous time step $\boldsymbol{S_{t-1}}$. This contains a set of $\boldsymbol{X_{t-1}}$ which is a vector including the robots position and orientation $\boldsymbol{X_{t-1}}$ = [X,Y,θ] and an associated weight $\boldsymbol{w_{t-1}}$. The algorithm is based on iterative random sampling, which utilizes the Bayesian Particle Filter model. Therefore $\boldsymbol{S_{t-1}}$ is the previous belief of the localization model. At the first time step, $\boldsymbol{t = 0}$, the set particles are given random values within the bounds and each particle is assigned the same weight. With each continuous time step, the motion model is updated, adding the change of motion with some noise to the robots position and orientation [X,Y,θ]. The corresponding weight is also updated using the sensor model as shown in Figure 2 below.

Monte Carlo Localisation Algorithm

```
1  S_t-1 = {X_t-1(i), w_t-1(i)};                    … Particle Set
2  S_t = 0, a = 0;                                  … Initialise
3  for i = 1,… n do
4     particle(j) = particle(i);                    … Sample Particle j
5     particle(j).x = particle(i).x + dX + noiseX;  … Predict Next State
6     particle(j).y = particle(i).y + dY + noiseY;
7     particle(j).θ = particle(i).θ + dθ + noiseθ;
8     particle(j).w = 1-(Perception-Sensor Reading); … Update Weight
9     a = a + particle(j).w;                        … Update Factor
10    S_t = S_t U {X_t(i), X_t(i)};                 … Update Set
11 for i = 1,… n do
12    particle(j).w = particle(j).w / a;            … Normalise Weight
13 return S_t
```

Figure 2 – *Monte Carlo Localization Algorithm*

Occupancy Grid Map:

The starting point of this project was to analyze the Occupancy Grid map. The supplied 2D map image of the "Mine" map is 171 pixels wide and 101 pixels long. The Occupancy Grid map also contains the actual perception of the 2D obstacles associated with the "Mine" map. This is demonstrated using the grayscale value of each of the pixels within the Occupancy Grid map, ranging from 0 (black) to 255 (white). Each pixel with a grayscale value of 0 demonstrates an obstacle within the map, where each pixel with a value of 255 demonstrates an area free of obstacles within the map. The "Mine" Occupancy Grid map is shown in Figure 3 below along with the interpreted pixel coordinates and probability scale relationship. The Occupancy Grid Map is used during the Weight Update phase of the algorithm.

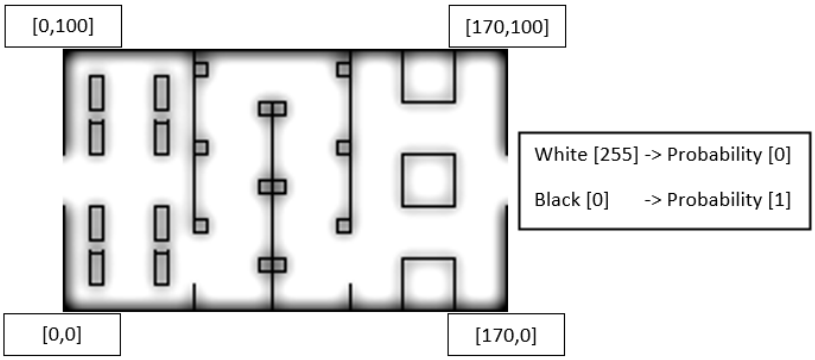

Figure 3 – *"Mine" Occupancy Grid Map*

Environment Coordinate Map:

The 2D "Mine" map detailing the Map Coordinates was also supplied as shown in Figure 4 below. The map provides the global coordinates of the known environment within which the robot travels. This includes the coordinates of any obstacles within the environment. The X and Y bounds of this Coordinate Map provide the range within which the initial random particle positions are selected.

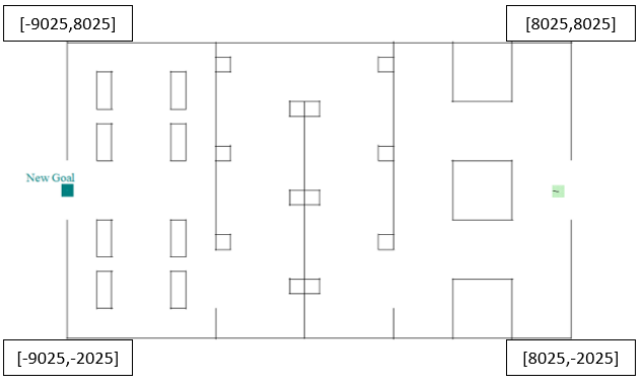

Figure 4 – *"Mine" Coordinate Map*

Motion Model Update:

The objective of the Motion Model Update phase of the Monte Carlo Localization algorithm is to predict the next position and orientation [X,Y,θ] based on the belief of the previous particles and the system dynamics of the robot [dX,dY,dθ]. For this phase, a particle was sampled and the difference in X,Y and θ was calculated. Due to uncertainty present in the movement of a mobile robot in a real world scenario, an additional gaussian noise is sampled from the system dynamics and added to compensate for each particle component [X,Y,θ]. The Motion Model Update calculation is shown in Figure 5 below.

```
particle(j).x = particle(i).x + dX + noiseX;
particle(j).y = particle(i).y + dY + noiseY;
particle(j).θ = particle(i).θ + dθ + noiseθ;
```

Figure 5 – *Motion Model Update calculation*

Sensor Model Update:

The objective of the Sensor Model Update phase of the Monte Carlo Localization algorithm is to update the weights [w] of each particle as a measure for how accurate the particles position and orientation is. The principle is based on the probability that if the particle was in the predicted position, that the sensor reading would reflect this predicted position. The weight of the particle is updated according to the following calculation:

```
particle(j).w = 1-(Perception-Sensor Reading);
```

Where the ***particle(j).w*** is the particle weight, ***Perception*** is the current particle perception in the Occupancy Grid Map and ***Sensor Reading*** is the actual particle perception in the Occupancy Grid Map where the obstacle was sensed. For this implementation using the Aria C++ library, the Left and Right sensor of the Pioneer robot were used to sweep the environment and gather sensor data. This Sensor Model Method is demonstrated in Figure 6 below.

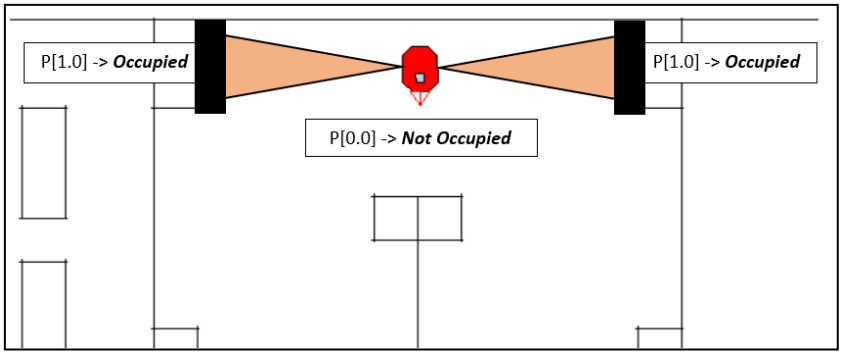

Figure 6 – *Sensor Model Update strategy*

Software Implementation:

As previously stated, the Occupancy Grid Map and Coordinate Map of the environment were provided. Also provided was a main.cpp file initializing the Occupancy Grid Map, the Aria Robot Actions and the SFML Visualization window. The software implementation of the Monte Carlo Localization algorithm was broken into three main methods. The structure of each method is described below in Figure 7.

MONTE CARLO LOCALISATION IMPLEMENTATION – PSEUDO CODE

```
Localisation::setup()
{
      ** Initialise Variables **
      Previous Robot Position [X,Y,θ]
      Occupancy Grid Dimensions
      for (x = 0 … 170)
      {
            for (y = 0 … 100)
            {
                  Initialise Particle Set [X,Y,θ,Weight]
            }
      }
}

Localisation::update()
{
      Calculate Motion Difference [dX,dY,dθ]
      for (x = 0 … 170)
      {
            for (y = 0 … 100)
            {
                  Calculate Distance [Robot    -> Grid Cell]
                  Calculate Distance [Particle -> Grid Cell]

                  if(Distance 1 AND Distance 2 < Threshold)
                  {
                        Sample Particle [X,Y,θ]
                        Update Particle [dX,dY,dθ]
                        Update Particle Weight from Sensor value
                        Update factor alpha
                  }
            }
      }

      for (x = 0 … 170)
      {
            for (y = 0 … 100)
            {
                  Normalise Particle Weights
            }
      }
}

Localisation::draw()
{
      for (x = 0 … 170)
      {
            for (y = 0 … 100)
            {
                  if(Particle Weight > Threshold)
                  {
                        Draw Particle
                  }
            }
      }
}
```

Figure 7 – *Implemented MCL Algorithm – Pseudo code*

Localization::setup():

This method was used to initialize all required variables. Only one instance of this method exists, and that is at application start up. The most important functionality developed within this method was to initialize the set S_{t-1} with a set of particles X_{t-1} and associated weights w_{t-1}. This was achieved by declaring a particle array of size [170][100] and assigning [X,Y,θ] values within the bounds of the Map within a for loop. Also within the for loop, the particle weights were assigned where initially each particle was assigned the same weight. A summary of the actions within this method are defined below:

- Determine Occupancy Grid Dimensions.
- Initialize particle set. Implement For Loop assigning random [X,Y,θ] values within the bounds of the map and assigning equal particle weights.

Localization::update():

This method was used to update the initial set of particles using the Monte Carlo Localization algorithm. This method is executed once a second, following the initial ***localization::setup*** method. The main function of this method is to update the appropriate particles within the set as per the Monte Carlo algorithm. A summary of the actions within this method are defined below:

- Determine Motion Difference of the robot from the last time step [dX,dY,dθ].
- Implement For Loop from x [0…170] and y [0…100] to cover the Occupancy Grid Map.
- Calculate Coordinate Map cell [Grid Cell] according to x and y from For Loop [x = -9025…8025] [y = -2025…8025].
- Calculate the Distance from Robot -> Grid Cell and Particle -> Grid Cell.
- Update the particles if both Distances are less than 1250mm [Bounding Box].
- Determine suitable values for sigma U, sigma L, i, j and use to sample [X,Y,θ] values.
- Update the particle position and orientation values [X,Y,θ].
- If the left or right sensor values are less than 1000mm, update the particle weight.
- Insert the updated particle into the set.
- Update the normalization factor.
- Implement another For Loop to cover the Occupancy Grid Map, in order to normalize the particle weights.

Localization::draw():

This method was used to update the SFML visualization window with the appropriate particles. This method is also executed once a second, following the initial ***localization::setup*** method. The main function of this method is to check that the particle weight is above a certain threshold, before plotting it on the SFML visualization window. A summary of the actions within this method are defined below:

- Check if Boolean value is true to plot particles.
- Initialize SFML rectangular shape object to represent the particles.
- Implement For Loop from x [0…170] and y [0…100] to cover the Occupancy Grid Map.
- Check if the particle weight is greater than a specified threshold.
- If the particle weight is greater than the specified threshold, set the position and orientation of the SFML rectangular object to that of the particle.
- Draw each rectangular object on the SFML visualization window.

Experimental Results:

Testing Objectives:

To test the functionality of the implemented software, the supplied “Mine” Coordinate Map along with the associated Occupancy Grid Map was tested. The Pioneer robot was placed in multiple locations about each of the environments. The first objective was to see the particles converge towards the position of the Pioneer robot over time. The second objective was to display an accurate representation of the Pioneer robots current global position in the console of the application. The localization results of the MCL algorithm for the Pioneer robot in both test environments is shown below. Accompanying the following results of images, is a video which demonstrates the localization algorithm performing.

Map 1 – Mine Map:

Result 1:

- Particles successfully converge about the robot.
- Global position of particles printed in the console provide good estimate of position.

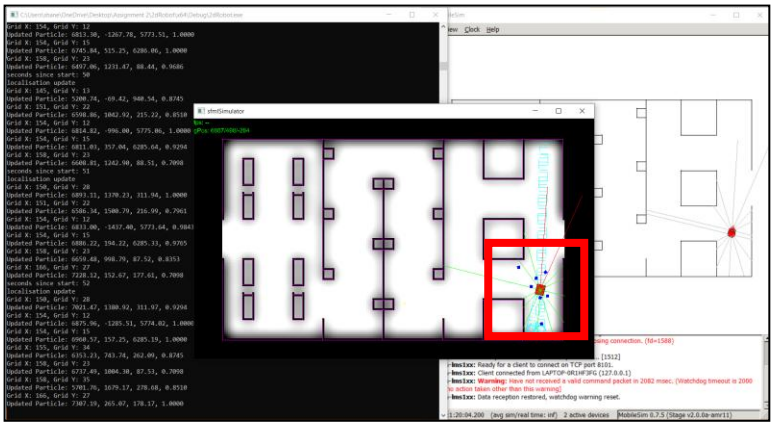

Figure 8 – *MCL Algorithm Test – Result 1*

Result 2:

- Particles successfully converge about the robot.
- Global position of particles printed in the console provide good estimate of position.

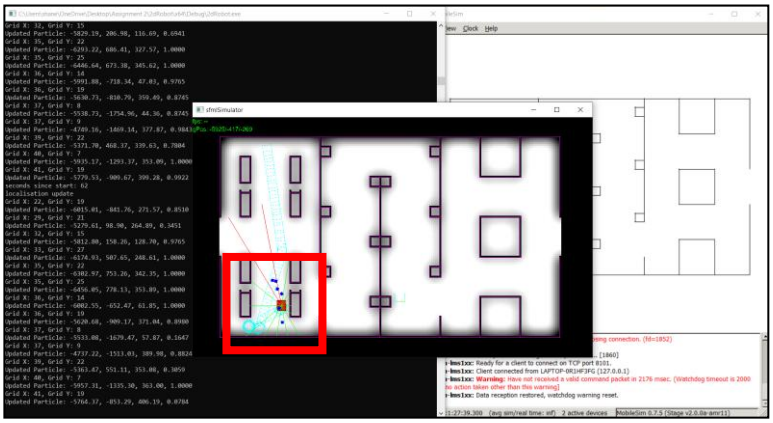

Figure 9 – *MCL Algorithm Test – Result 2*

Result 3:

- Particles successfully converge about the robot.
- Global position of particles printed in the console provide good estimate of position.

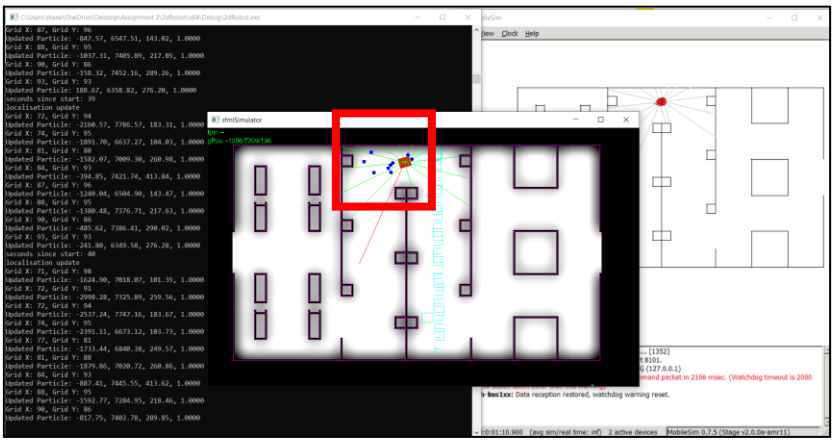

Figure 10 – *MCL Algorithm Test – Result 3*

Result 4:

- Particles successfully converge about the robot.
- Global position of particles printed in the console provide good estimate of position.

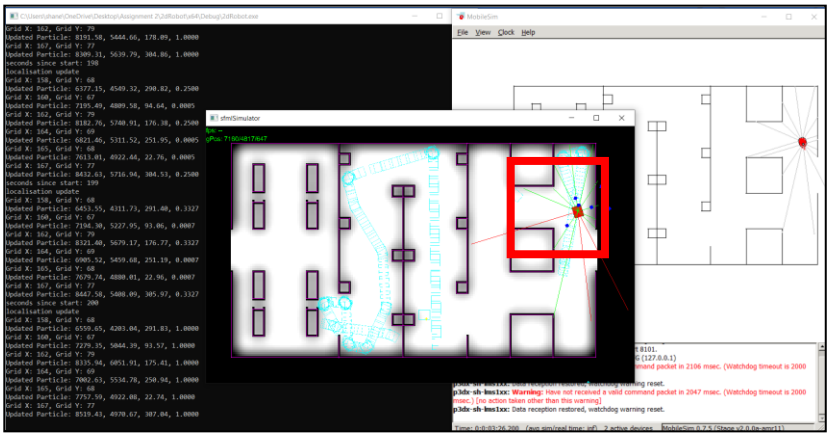

Figure 11 – *MCL Algorithm Test – Result 4*

Result 5:

- Particles successfully converge about the robot.
- Global position of particles printed in the console provide good estimate of position.

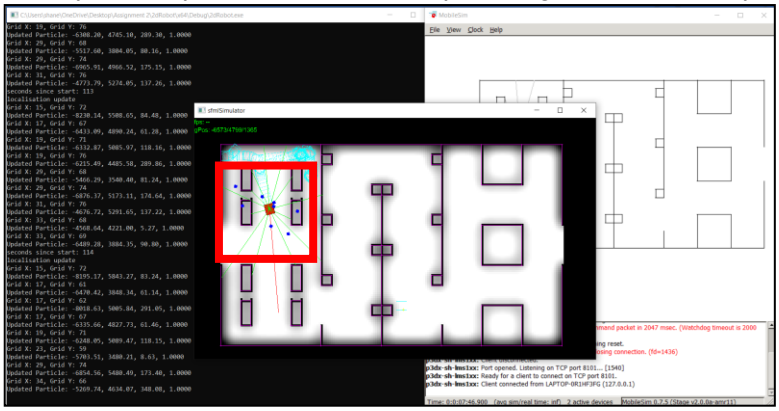

Figure 12 – *MCL Algorithm Test – Result 5*

Conclusion/Learnings:

For the implementation of Robot Localization in a known 2D environment, the Monte Carlo Localization algorithm was used. The objective was to develop a Localization method in C++ to deploy the algorithm and use the SFML visualization library to display the particles as the robots position updated (once a second). As previously discussed, there were three main methods deployed for completing this task, localization::setup(), localization::update() and localization::draw(). The first method was used to initialize the particle set with [X,Y] positions within the bounds of the coordinate map. Also initialized were the orientation [θ] of the particles and each was initialized with the same weight value [w]. The update method was then used to implement the Monte Carlo Localization algorithm to update the particle positions, orientations and weights once a second. The updated particles were printed to the console of the application to give an accurate estimate of the position, orientation and weight of the updated particles. Finally, the third method was used to draw the updated particles above a weight threshold onto the SFML visualization window.

Once the above methods were successfully deployed, the next phase to verify the functionality was to perform testing. From the testing and results phase of this implementation, it was clear that the created Monte Carlo Localization algorithm gave an accurate representation of the particle positions with a high weight within the known 2D environment. The results show the particles with high weights converging towards the global position of the robot which was the initial objective. The algorithm seems to function very well, while demonstrating the importance of using a probabilistic approach to determining both the motion update and the sensor update. It is clear that when deploying a mobile robot in a known or unknown environment, factors such as the Gaussian noise (Motion Model Update) and also the probabilistic approach to obtained sensor readings (Sensor Model Update) are essential requirements. Future works will allow for additional maps to be created with different Occupancy Grid Map sizes and more challenging boundaries/obstacles to perform robot localization with.

Bibliography:

[1] F. Dellaert, D. Fox, W. Burgard, S. Thrun, *Monte Carlo Localization for Mobile Robots*, IEEE International Conference on Robotics and Automation, 2:1322 – 1328, vol.2 · February 1999 [PAPER].

[2] F. Dellaert, D. Fox, W. Burgard, S. Thrun, *Robust Monte Carlo Localization for Mobile Robots*, School of Computer Science, Carnegie Mellon University, Dept. of Computer Science and Engineering, University of Washington, Computer Science Department, University of Freiburg · April 2000 [PAPER].

[3] C. Stachniss, L. Spinello, *Introduction to Monte Carlo Localization*, University of Freiburg – Lecture Notes, School of Computer Science, · 2012 [ONLINE].

[4] A.W. Alhashimi, G. Nikolakopoulos, T. Gustafsson, *Observation Model for Monte Carlo Localization,* Department of Computer Science, Electrical and Space Engineering, Lulea, Sweden, · 2014 [PAPER].

[5] D. Fox*, Adapting the sample size in particle filters through KLD-sampling*, The International Journal of Robotics Research, 22(12), 985–1003, · 2003 [PAPER].

[6] S. Lenser and M. Veloso, *Sensor Resetting Localization for poorly modelled Mobile Robots*, In Robotics and Automation, Proceedings. ICRA'00. IEEE International Conference on, volume 2, 1225–1232, · 2000 [PAPER].

[7] P. Pfaff, C. Stachniss, C. Plagemann and W. Burgard, *Efficiently learning high-dimensional observation models for monte-carlo localization using gaussian mixtures*, In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, 3539– 3544, · 2008 [PAPER].

[8] D. Fox*, Adapting the sample size in particle filters through KLD-sampling*, The International Journal of Robotics Research, 22(12), 985–1003, · 2003 [PAPER].

[9] L. Chen, P. Sun, G. Zhang, J. Niu and X. Zhang*, Fast Monte Carlo Localization for Mobile Robot,* International Conference on Electronic Commerce, Web Application, and Communication ECWAC 2011: Advanced Research on Electronic Commerce, Web Application, and Communication pp 207-211, · 2011 [PAPER].

[10] A. Kaboli, M. Bowling and P. Musilek*, Bayesian Calibration for Monte Carlo Localization,* AAAI Association for the Advancement of Artificial Intelligence, Computer Science, · 2006 [PAPER].