Intelligent Mobile Robots
Assignment 2 - Localisation in a Known Environment

Arpit Sharma
P Number: P2613237,
Intelligent Systems, and Robotics MSc,
De Montfort University, Leicester, England.

Figure 1: A Pioneer P3DX robot and a Hokuyo URG 04LX UG01 Laser Scanner

1 Introduction

Mobile localization is one of the main problems of mobile robots. Generally, it is referred as “the most fundamental problem to providing a mobile robot with autonomous capabilities” [1]. The ”global localization problem”[5][2] is much more severe as the ”error”[3] in the sensor readings sum out to be huge, and hence, cannot be ignored.

An algorithm, Monte Carlo localization, in other words, particle filter localization, is used in the experiment for the Pioneer robot to find itself in the environment using the particle filter given to it the internal map of the environment, beforehand. The position and heading of the robot are predicted and corrected simultaneously by the robot as it moves in the environment.

The combination of the Pioneer P3DX robot with the Hokuyo URG Laser Scanner (shown in figure 1) mounted on its top can be used for this experiment.

2 Localization technique

The particles are initialized randomly all over the environment, uniformly, as the robot cannot judge its location in the beginning, and hence, the probabilities of the robot being anywhere in the configuration space are equal. New states are predicted as the robot moves in the environment and hence, the particles move relative to the robot.

In the algorithm, the belief, $Bel(x)$ is represented by by a set of n distributed weighted samples as follows:

$Bel(x) \approx \{x_t^i, w_t^i\}_{i=1,.....,n}$ [7]

When the laser sensor detects a point in the environment corresponding to a cell in the occupancy grid, let us say, having probability, p at a distance, r, and heading theta. All the particles are treated as if what would the robot detect if the robot were in their place, at the same distance, r and heading theta. The lesser the difference in the probability found by the sensor and the probability given by the particle (retrieved from the map), the more certain we are of the particle representing the correct position of the robot, and hence, more ‘weight’ is assigned to the particle and vice-versa. Otherwise, if the difference is very large, the particle must be re-sampled near the more promising region (in the vicinity of the particles with higher weights) in the map. This is done for every particle in the algorithm. It is expected that the particle will eventually converge to the real position of the robot according to the probability density function (PDF), which is a discrete function composed of particles. Some noise is applied considering that the odometer and laser sensor readings are not fully accurate.

It is based on the Bayes Filter, which depends on the current observations, a set of previous observations, and previous control measurements. The PDF at time t is given by the equation,

$$Bel(x_t) = p(x_t|z_t, u_{t-1}, z_{t-1}, u_{t-2}........, u_o, z_o) \text{ [7]}$$

It is recursively updated in two stages:

1. Prediction - To predict the latest values of the $Bel(x_t)$ according to the control measurements, according to the following formula.

$$Bel^{-}(x_t) = \int p(x_t|x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

2. Correction- To correct the new values of $Bel(x_t)$, according to the observations, according to the following formula.

$$Bel(x_t) = \alpha p(z_t|x_t) Bel^{-}(x_t)$$

2.1 Monte Carlo Localization Algorithm Flowchart

The flowchart given in figure 2, was built to implement the algorithm. The steps are explained in the sections below.

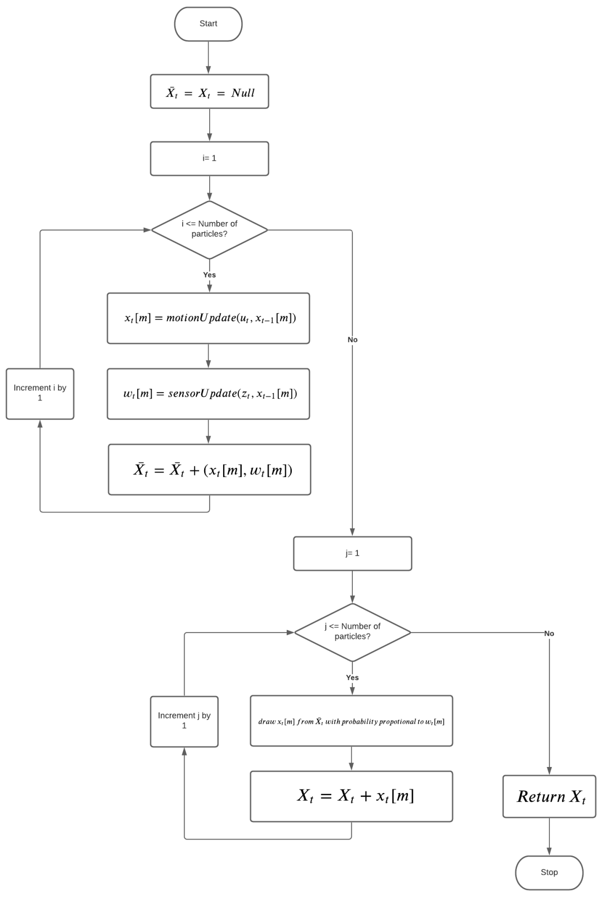

Start
$\bar{X}_t = X_t = Null$
i= 1
i <= Number of particles?
Yes
No
$x_t[m] = motionUpdate(u_t, x_{t-1}[m])$
$w_t[m] = sensorUpdate(z_t, x_{t-1}[m])$
$\bar{X}_t = \bar{X}_t + (x_t[m], w_t[m])$
Increment i by 1
j= 1
j <= Number of particles?
Yes
No
draw $x_t[m]$ from $\bar{X}_t$ with probability propotional to $w_t[m]$
$X_t = X_t + x_t[m]$
Increment j by 1
Return X_t
Stop

Figure 2: Monte Carlo Localization Algorithm Flowchart

2.2 State representation

Each particle represents the possible state of the robot based on the evidence collected from the laser sensor and odometry of the robot. In other words, a particle is just a guess made by the robot of its possible positions in the environment. Hence, they contain the full description of the position of the robot. As in the experiment, the robot does not fly and just travels on a plane surface, a particle can be represented by the possible x-coordinate, y-coordinate, and the angle for the heading of the robot. Hence the state of the robot can be represented by a tuple, (x, y, θ).

As the current state's PDF does not depend on

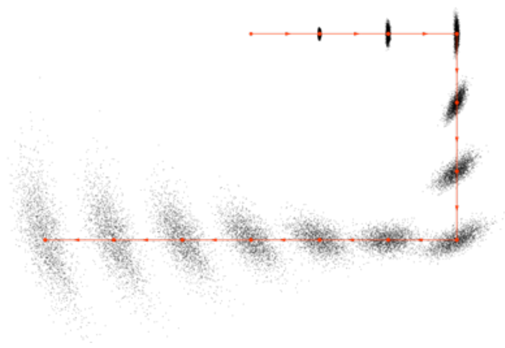


Figure 3: Motion without enough sensing [4]

any previous state, except the one just before it as x_t only on x_{t-1}, it shows that the algorithm uses the Markov [8] property. It can only be used if the environment does not change and is fixed.

2.3 Motion update

As the robot moves, all the particles move along with it, not just in the x-axis and y-axis, but also according to the changes in the heading of the robot. For example, if the robot moves 1 meter to the left, all particles move 1 meter to the left, irrespective of their position. Similarly, if the robot moves 90 degrees clockwise, all the particles also rotate 90 degrees clockwise. The motion update is performed according to the odometry of the robot. The odometers in the real world are not error-free. Sometimes the reading may overshoot or undershoot the wanted motion. The errors can be caused because of the minute difference in the radius of the two wheels of the robot, and this leads to noise in the odometry readings. If the robot just keeps moving without regularly updating the particle filter, these errors can sum out to be huge and could end up in the robot being less sure of its position resulting in the diverging of the particles (as shown in the figure 3), the more it moves in the environment without taking sufficient laser sensor readings.

2.4 Sensor update

As discussed in the above section, to prevent the particles from diverging, the laser sensor updates should take place at a very high rate. Each particle is assigned a weight(probability) according to how accurately it predicts the position of the robot given the laser sensor reading. Then, the M number of particles is selected randomly from the previous belief that has a similar weight as the particle. The consistent particles with the laser sensor readings have a higher chance of getting selected and vice-versa. Hence, eventually, the particles converge towards the real position of the robot.

2.5 Drawback

2.5.1 Particle deprivation

If after the particles have converged (the robot is localized), the robot faces the "kidnapped robot problem"[6], that is, it is picked and placed at some other position far away in the map, the converged particles will not get selected anymore as they are far away, and far away particles rarely get selected because more importance is given to the close ones, and, will eventually get eliminated. Hence, the algorithm will stop working. This problem is caused mostly when the number of particles is very less, and the map is big. There is always some possibility of the algorithm to discard all particles during the re-sampling, which were near the correct state.

One way of solving this problem is by adding a few particles randomly in every iteration. This prevents the algorithm from getting trapped by making sure that at least a few particles are everywhere on the map in case the robot is picked and dropped to some other place on the map.

2.5.2 Computationally expensive

For a big map or for high accuracy, large number of particles are needed, but this results in high computational expense as more particles have to be updated after each cycle. As in the MCL algorithm, the number of particles remain fixed, the computational power consumption remains constantly high. That is one of the reasons, AMCL algorithm is proffered over it.

2.5.3 Can only be used in a fixed environment

The algorithm can only work in a fixed environment because of the reasons discussed earlier.

3 Software Implementation

3.1 ROS Terminal

It is the backbone of the ROS development environment. All the commands have been types in the command in six different terminals in order as follows:

3.1.1 Terminal 1

```
sudo apt install ros-$ROS_DISTRO-pr2-teleop
ros-$ROS_DISTRO-joy ros-$ROS_DISTRO-slam
-gmapping ros-$ROS_DISTRO-map-server
cd ~
cd catkin_ws
source devel/setup.bash
roscore
```

3.1.2 Terminal 2

```
cd ~
cd catkin_ws/src/socspioneer/data
rosrun stage_ros stageros lgfloor.world
```

3.1.3 Terminal 3

```
roslaunch socspioneer keyboard_teleop.launch
```

3.1.4 Terminal 4

```
cd ~
cd catkin_ws/src/socspioneer/data
rosrun map_server map_server lgfloor.yaml
```

3.1.5 Terminal 5

```
rosrun rviz rviz
```

3.1.6 Terminal 6

```
python -m pip install XlsxWriter
```

3.1.7 Terminal 7

```
cd ~
cd catkin_ws/src
cd pf_localisation0/src/pf_localisation
chmod +x pose_printer.py
rosrun pf_localisation0 pose_printer.py
```

3.1.8 Terminal 8

```
cd ~
cd catkin_ws/src/pf_localisation0/src
chmod +x node.py
rosrun pf_localisation0 node.py
```

3.2 The sections of the problem implemented

3.2.1 Initialize all the attributes of the algorithm

This is the very first stage when the program runs. All the parameters including the maximum number of particles, total number of readings of the laser sensor for weight calculations, noise scaling factors, the variables for the clustering technique, and the publisher for printing the estimated robot position to the console and an excel file, are initialized.

3.2.2 Initialise Particle Cloud

The particles are initialized when a new initial pose is set. For all particles, noise is added to the pose components using pseudo-random numbers around the initial pose. The x and y co-ordinates of the noise, are generated using random decimal number with Gaussian distribution with a pre-decided fixed sigma value using the following formula:

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

The heading or rotation is calculated using a decimal number with von Mises distribution[9] or circular normal distribution with its corresponding sigma value. The components are scaled using the respective scaling factors. The x and y components of the noise are simply added, but the rotation noise is added using the 'rotateQuaternion' method.

The new particles after adding noise to them around the initial pose are returned as an pose array at this step.

3.2.3 Update Particle Cloud

The particle cloud has to be regularly updated according to the latest Laser sensor readings. It is where the particle filtering actually takes place. The simultaneous movement of the particles according to the movement of the robot is taken care by the 'PFLocaliserBase' class. Since the odometry readings are not very promising and are prone to noise in the real world, the laser sensor readings are used to verify the estimated location.

While re-sampling the particles, to have some bias towards high weighting particles, as they are more promising of the robot's position, roulette-wheel selection has been used. Random noise is added to each new particle, from Gaussian distribution to x,y components, and from von Mises distribution the heading of the particles with sigma values different from that in the 'Initialise Particle Cloud' stage.

Additionally, the noise scaling factors are reduced gradually till a point so that there is maximum noise in the beginning, but as the algorithm advances, the noise reduces because of reduced scaling factors as less noise is needed, once the particles have converged to the 'pose' of the robot.

The new particles after adding noise to them around the initial pose are returned as an pose array at this step.

3.2.4 Estimate Pose of the robot

This is the stage that gives the final output of the cycle, that is the estimated pose of the robot. The following four techniques were used to predict the pose from the set of particles.

1 The Global Mean Technique - This technique just takes the average position and heading of all the particles using the following formula (where $\{X_{particles}\}$ is the set of all particles). This technique is incapable of rejecting the outliers by selecting a more promising cluster. Therefore, not much good output was achieved with this approach.

$$Estimated\,state\ = mean\{X_{particles}\}$$

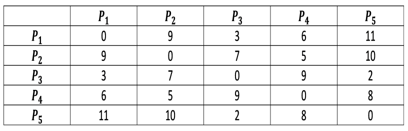
2 The Global Median Technique - This technique was implemented by taking the median of all the components of all the particles using the following formula. This technique was used because the median is not affected by outliers, and hence, the results were much more promising than the "Global Mean" approach results.

$$Estimated\,state\ = median\{X_{particles}\}$$

3 The Best Particle Technique - This technique just selects the particle with the best weight (highest probability of correctly estimating the robot's pose i.e. highest weight) by using the following formula (where $x_{highest\,weight}$ is the particle with the highest weight). Again this technique can lead to an erroneous output just because the probability calculated on the erroneous laser sensor readings happened to be the same as returned by that particle .

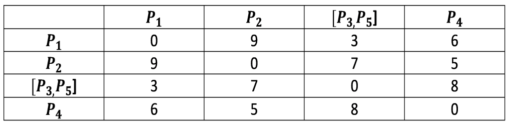
$$Estimated\,state\ = x_{highest\,weight}$$

4 The Hierarchical Agglomerative Clustering Technique - Finally, Hierarchical Agglomerative Clustering (HAC)[10] was used for selecting the



	P_1	P_2	P_3	P_4	P_5
P_1	0	9	3	6	11
P_2	9	0	7	5	10
P_3	3	7	0	9	2
P_4	6	5	9	0	8
P_5	11	10	2	8	0

Figure 4: First distance matrix



	P_1	P_2	$[P_3 P_5]$	P_4
P_1	0	9	3	6
P_2	9	0	7	5
$[P_3 P_5]$	3	7	0	8
P_4	6	5	8	0

Figure 5: Next distance matrix

densest and promising cluster and get rid of outliers. It is the most sophisticated clustering technique used in the experiment.

In working, the technique is very straight forward.It uses 'agglomerative' technique that uses bottom-up approach. Let P_1, P_2, P_3, P_4,and P_5 be five particles(just for illustration).

As shown in figure 4, a distance matrix is made by finding the distance between particles/clusters .Then, the minimum distance in the matrix is searched. In the above example, as P_3 and P_5 have the minimum distance of 2 (the diagonals don't count), hence, they form a cluster. Hence in the next distance matrix, they form a single unit as shown below.

Again, the minimum distance is calculated between particles, between particle and cluster, or between clusters. Here to find the distance between clusters, linkage method implemented comes into picture. The 'single linkage' method was used in the experiment that uses the 'minimum' of the distances between the each combination of particles between clusters. For example, for P_2 particle and $[P_3, P_5]$ cluster,

$Distance\, between\, them =$

$minimum(distance(P_2, P_3), distance(P_2, P_5))$

$\Rightarrow minimum(7, 10)$

$\Rightarrow 7$

Hence according to it, next distance matrix (as shown in figure 5) is computed and then, its next hierarchy of cluster is computed. Finally a 'Dendrogram' or a tree of clusters is generated. Finally, the cluster with the highest weight(probability) is selected. The mean of all the pose components of all the particles of the best cluster are calculated, and it is returned as the estimated pose of the robot. The maximum inter-cluster distance allowed was kept as low as 0.4 to have higher number of clusters for higher accuracy.

4 Tuning and Testing

4.1 Number of laser scan readings

The total number of readings of the laser sensor used for weight calculations per cycle was made equal to 60. That means each time the weights update, the last 60 laser scan readings are considered. The default value was 50, but better results were obtained at 60 scan readings.

4.2 Number of particles

The total and the fixed(as the algorithm is not 'adaptive' MCL) number of particles used in the experiment were 1000, according to how big the map is. The results with 200 particles were not very promising and often lead to particle deprivation in some regions of the map, even at the starting of the program. Generally, the number of particles are decided according to the size of the map,but more the number of particles, the better the algorithm performs, but it becomes much more computationally expensive. Therefore, 1000 particles were sufficient as the results were much more promising and accurate. Increasing particles beyond this point resulted in a lag in the program because of increased computation.

4.3 Noise

Two sets of noise have been used in the experiment.

The first type was the initial noise for spreading the particles uniformly all over the map as the algorithm starts. Translational, drift, and rotational noise have been used for the x-coordinate, y-coordinate, and the heading of the particles, of value 0.1, 0.1, and 0.07 respectively. Gaussian Standard Deviation for the first two components was set to 150 and the VonMises Standard Deviation for the rotational component was made 100. The values are higher than those used for the pose updating phase because the particles have to be spread all over the map. Hence, the value was decided upon visualizing whether the particles were spreading all over the map in the starting on the map or not.

The second type of noise was of the same type as the first one but was used during the pose updating phase. Although initially, the values were even larger than the first one, they are adaptive, as in they reduce overtime to much lower values of 0.05, 0.05, and 0.02 respectively, as the algorithm progresses. Also, lower standard deviations of 30 and 75 were used respectively, because these noises are just for adding a little noise to the particles in the clusters. A very low noise reduction factor was used so that the noise decreases very gradually.

4.4 HAC cut-off distance

Pose Estimation Constant, Cut-off distance (the maximum inter-cluster distance allowed) of the hierarchical clustering tree for HAC algorithm was set to a lower value of 0.4 to form more clusters to achieve better accuracy.

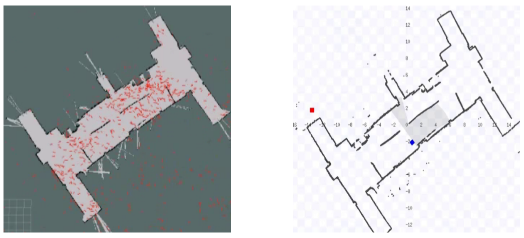


Figure 6: The particles are randomly distributed in the starting of the algorithm

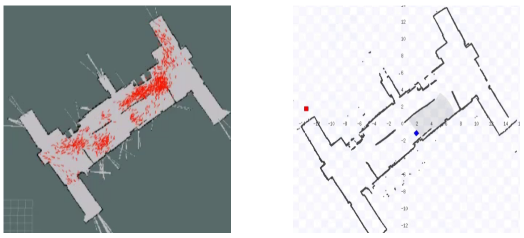


Figure 7: The particles began to dense around the promising regions

5 Results

5.1 Stages in the localization of the robot using the MCL algorithm

5.1.1 Stage 1

In the starting, the particles were evenly distributed all over the place as the chances of the robot being anywhere in the environment were equal. No cluster formations were noticed at this stage as shown in figure 6.

5.1.2 Stage 2

The particles began to concentrate near the possible locations of the robot because of the updated odometry and laser scan readings, as shown in figure 7.

5.1.3 Stage 3

The clusters began to form at this stage , as shown in figure 8. This is very important to notice that, due to the symmetry in the map, the four clusters in the middle of the map are formed as the algorithm cannot yet decide on which side the robot truly is. The fifth one on the corner was most

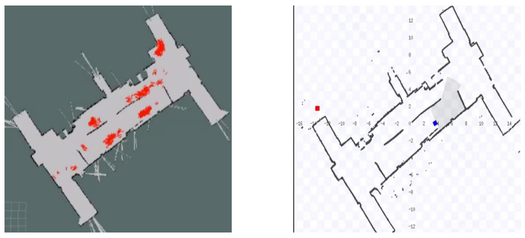


Figure 8: 'Symmetric' cluster formation starts

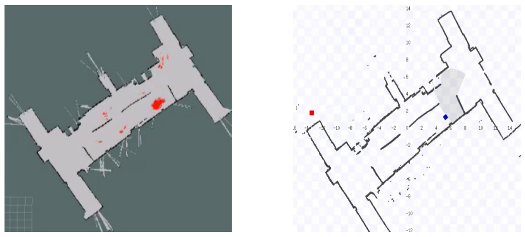


Figure 9: less promising clusters vanish and the best cluster becomes denser

probably formed because of the rotation and had started to vanish at this stage as the probability of the robot being there, calculated by the algorithm was extremely low.

5.1.4 Stage 4

As can be easily seen in the figure 9, due to the notch on the right side, and an open area on the left side of the robot, the symmetry in the map ends, and hence, the other clusters start vanishing. The cluster around the robot's true position begins to become denser.

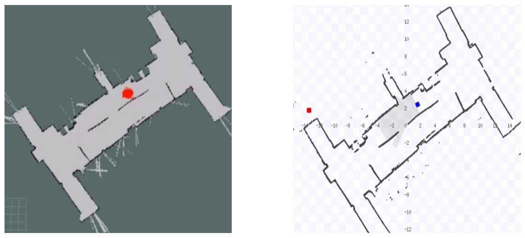


Figure 10: The particles converge around the true robot 'pose'

5.1.5 Stage 5

Finally, the particles converge around the true position of the robot as the algorithm is now sure about the position of the robot, as shown in figure 10.

The estimated pose of the robot predicted by the 'estimate_pose' method is returned and gets published to the 'estimatepose' topic. A dedicated listener, located in the 'pose_printer.py' file, was implemented to listen to this topic, and print the pose after some formatting to the console and an excel file with the time stamp, every second. The two figures, figure 11 and figure 12 in the appendix, present a sample of an output produced by the listener.

6 Conclusion

As the number of particles for the algorithm increase, the accuracy also increases, but at the cost of the computational power of the computer of the robot. They must be selected according to the size of the map as a bigger map would need more particles.

Therefore, it is more sensible to decrease the number of particles as the particles begin to converge to lessen the "computational burden"[11].

MCL can be improved by using the Kullback–Leibler divergence (KLD) [12] in which, the particles are sampled in an adaptive way by estimating the error.

It had to be ensured that a few particles are there, placed randomly on the map to prevent the algorithm from failing if the robot is kidnapped and dropped somewhere else on the map.

References

[1] Ingemar J Cox. "Blanche-an experiment in guidance and navigation of an autonomous robot vehicle". In: *IEEE Transactions on robotics and automation* 7.2 (1991), pp. 193–204.

[2] Wolfram Burgard et al. “Integrating global position estimation and position tracking for mobile robots: the Dynamic Markov Localization approach”. In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190).* Vol. 2. IEEE. 1998, pp. 730–735.

[3] Frank Dellaert et al. “Monte carlo localization for mobile robots”. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C).* Vol. 2. IEEE. 1999, pp. 1322–1328.

[4] Dieter Fox et al. “Monte carlo localization: Efficient position estimation for mobile robots”. In: *AAAI/IAAI* 1999.343-349 (1999), pp. 2–2.

[5] Stergios I Roumeliotis and George A Bekey. “Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065).* Vol. 3. IEEE. 2000, pp. 2985–2992.

[6] Sebastian Thrun et al. “Probabilistic algorithms and the interactive museum tour-guide robot minerva”. In: *The International Journal of Robotics Research* 19.11 (2000), pp. 972–999.

[7] Sebastian Thrun et al. “Robust Monte Carlo localization for mobile robots”. In: *Artificial intelligence* 128.1-2 (2001), pp. 99–141.

[8] Ryuichi Ueda et al. “Uniform Monte Carlo localization-fast and robust self-localization method for mobile robots”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292).* Vol. 2. IEEE. 2002, pp. 1353–1358.

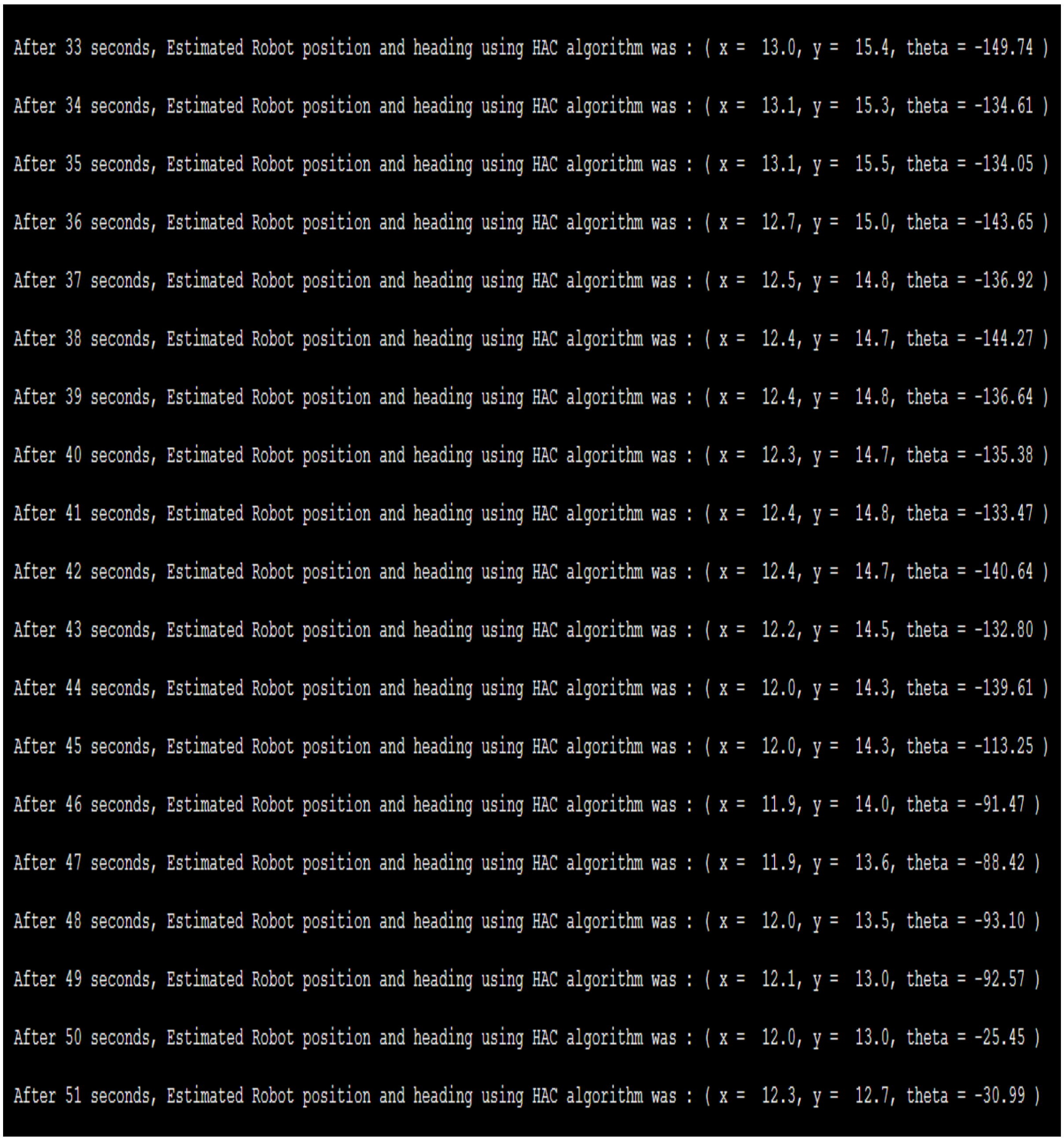
[9] Riccardo Gatto and Sreenivasa Rao Jammalamadaka. “The generalized von Mises distribution”. In: *Statistical Methodology* 4.3 (2007), pp. 341–353.

[10] Daniel Müllner. “Modern hierarchical, agglomerative clustering algorithms”. In: *arXiv preprint arXiv:1109.2378* (2011).

[11] Lei Zhang, Rene Zapata, and Pascal Lepinay. “Self-adaptive Monte Carlo localization for mobile robots using range finders”. In: *Robotica* 30.2 (2012), pp. 229–244.

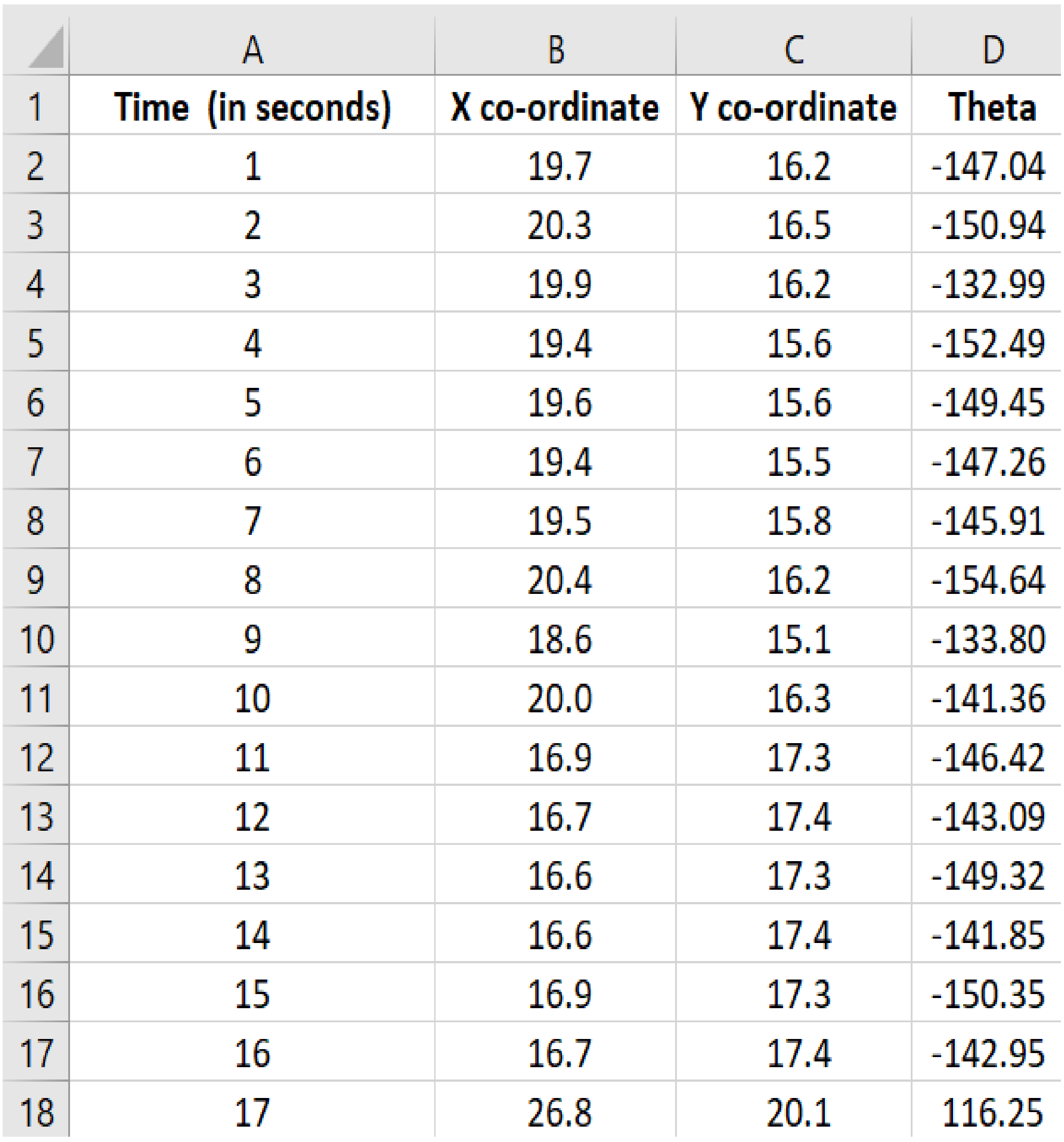
[12] Fernando Martın et al. “Kullback–Leibler divergence-based global localization for mobile robots”. In: *Robotics and Autonomous Systems* 62.2 (2014), pp. 120–130.

7 Appendix



```
After 33 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  13.0, y =  15.4, theta = -149.74 )

After 34 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  13.1, y =  15.3, theta = -134.61 )

After 35 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  13.1, y =  15.5, theta = -134.05 )

After 36 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.7, y =  15.0, theta = -143.65 )

After 37 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.5, y =  14.8, theta = -136.92 )

After 38 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.4, y =  14.7, theta = -144.27 )

After 39 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.4, y =  14.8, theta = -136.64 )

After 40 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.3, y =  14.7, theta = -135.38 )

After 41 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.4, y =  14.8, theta = -133.47 )

After 42 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.4, y =  14.7, theta = -140.64 )

After 43 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.2, y =  14.5, theta = -132.80 )

After 44 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.0, y =  14.3, theta = -139.61 )

After 45 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.0, y =  14.3, theta = -113.25 )

After 46 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  11.9, y =  14.0, theta = -91.47 )

After 47 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  11.9, y =  13.6, theta = -88.42 )

After 48 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.0, y =  13.5, theta = -93.10 )

After 49 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.1, y =  13.0, theta = -92.57 )

After 50 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.0, y =  13.0, theta = -25.45 )

After 51 seconds, Estimated Robot position and heading using HAC algorithm was : ( x =  12.3, y =  12.7, theta = -30.99 )
```

Figure 11: A sample of the estimated pose of the robot printed by the listener



	A	B	C	D
1	Time (in seconds)	X co-ordinate	Y co-ordinate	Theta
2	1	19.7	16.2	-147.04
3	2	20.3	16.5	-150.94
4	3	19.9	16.2	-132.99
5	4	19.4	15.6	-152.49
6	5	19.6	15.6	-149.45
7	6	19.4	15.5	-147.26
8	7	19.5	15.8	-145.91
9	8	20.4	16.2	-154.64
10	9	18.6	15.1	-133.80
11	10	20.0	16.3	-141.36
12	11	16.9	17.3	-146.42
13	12	16.7	17.4	-143.09
14	13	16.6	17.3	-149.32
15	14	16.6	17.4	-141.85
16	15	16.9	17.3	-150.35
17	16	16.7	17.4	-142.95
18	17	26.8	20.1	116.25

Figure 12: A sample of the estimated pose of the robot printed to an excel file