Monte Carlo Localisation: Localising a Mobile Robot Within a Known Environment

Adam Hubble
P17175774

Abstract – Mobile robot localisation is the problem recognised as the determination of the position and orientation, or pose, of a mobile robot within a given environment, from using sensory data that the robot collects as observations of its subjected environment, overtime. This paper explores the probabilistic localisation technique renowned as Monte Carlo Localisation (MCL), to mitigate the abovesaid localisation problem via pose estimation, for a wheeled mobile robot that exists within a known environment.

Keywords – Mobile robot; Localisation; Position and orientation; Pose; Sensory data; Observations; Probabilistic; Monte Carlo Localisation; Pose estimation; Known environment

I. Introduction

In the proceeding passages, explores the operational composition of the Monte Carlo Localisation (MCL) algorithm [1], as the localisation technique instructed for overcoming the mobile robot localisation problem [2], and for the application of a mobile robot navigating within a known environment. As well, is the operational overview and evaluation of the MCL techniques implementation, that has been adapted to the purpose of the Pioneer P3-DX [3] model of mobile robot, instructed for use.

Figure 1: Depiction of the Pioneer P3-DX mobile robot [3].

Notably, opposing the traditional application of the array of Sound Navigation and Ranging (SONAR) sensors that the Pioneer P3-DX model employs and operates with, and per the support of the Robot Operating System (ROS) software platform [4][5], the robot alternatively utilises an array of Laser Rangefinder (LR) sensors, that are preferred for the degree of precision demanded for estimating the pose of the robot, overtime. The preference for precision correlates with the information obtained by ultrasonic sensors, which is known to be "noticeably uncertain" [6], in consequence of ultrasonic sensor emissions undergoing "multiple reflections or specular reflection away from the sensor, giving false distance readings" [7].

Subsequently, the paper is assembled into five divisions of focus, one of which focuses is this very introduction. Explored by the proceeding sections, are discussions relevant to the operations of the localisation technique appointed, the adapted implementation of the technique said, as well as its performance evaluation in accordance with the relevant test routines conducted and concluding observations that concern the localisation techniques resultant capability, to address the localisation problem announced.

II. Localisation Technique

Recognisably, there exists two fundamental challenges in mobile robotics, relative to the accomplishment of accurate and efficient sensor-based localisation, "global position estimation and local position tracking" [8]; for which, a mobile robot "seeks to estimate its position in a global coordinate frame" [1] of a given environment or space. Simply, a mobile robot should be able to estimate and represent its "pose (location, orientation) relative to its environment" [2]. Global position estimation can be understood as the "ability to determine the robot's position in a priori or previously learned map" [8], based upon the robots' sensory observations of the subjected environment, and the displacements measured in the robots odometry data, as it pursues navigation. Upon initially localising the robot in the map, local tracking can then be acknowledged as the "problem of keeping track of that position over time", which is vastly problematic and considered the "most-studied problem" [1], as the global position estimation then "has to accommodate small errors in its odometry as it moves" throughout the environment. Given the nature of the work presented, the localisation problem anticipates that the robot does not initially know its position nor orientation in the environment provided, for which presents a "much more difficult localisation problem, that of estimating its position from scratch"; this is recognised as the global localisation problem, which is problematic as the robots pose estimate "cannot be assumed to be small" [2], initially. Moreover, even more problematic is the kidnapped robot problem [9], which is defined as a condition, for when a "robot is instantly moved to another position" in the corresponding environment, without being detected and acknowledged by the procedure governing the global position estimation of the robot. This problem is known to be typically used to exercise a robot's "ability to recover from catastrophic localisation failures" [2], where unlike the global localisation problem, it is possible that the robot "might firmly believe itself to be somewhere else at the time of the kidnapping". From knowing its position and orientation in the global coordinate frame of a given environment or space, can the robot "make use of existing maps, which allows it to plan and navigate reliably in complex environments" [8], whilst also being "efficient", when accompanied by accurate local tracking.

In attempt to rectify the two underlying challenges posed by mobile robot localisation, the work proposes the probabilistic, Monte Carlo Localization (MCL) algorithm, that adopts the techniques of Monte Carlo methods that were first "introduced in the seventies, and recently rediscovered independently in the target-tracking" [1] domain of computing. As recognised, MCL has the potential to "solve the global localisation and kidnapped robot problem in a robust and efficient" [8] manner and accommodate "arbitrary noise distributions"; although, the regular MCL algorithm is known to be prone to the kidnapped robot problem, given that there "might be no surviving samples nearby the robot's new pose after it has been kidnapped" to converge to. Fundamentally, MCL purposes to represent the pose or belief of the robot by "a set of samples" or particles, drawn from the "posterior distribution" of robot poses (a cloud of particles); MCL represents the posteriors that approximate the desired distribution (spread) via a "random collection of weighted particles", which simply estimate the state of the robot, recursively, as a particle filter [10].

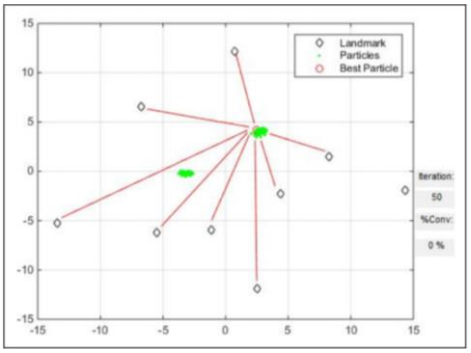

Figure 2: Visualisation of a kidnapped robot problem instance, where the red-linear lines denote the sensory readings of the robot, to each of the landmarks in the robot's environment from the origin, represented by the best particle [9].

Within the context of localisation, a particle filter as MCL classifies to be, aims to "track a variable of interest as it evolves over time" [10]. Particle filters employ a range of characteristics to achieve said behaviour and as sample-based density approximation methods, overlook previous works submitted to the field of mobile robot localisation, like that of Kalman's filter-based techniques and Markov's topological and grid-based approaches to localisation [8][1], that alternatively approximate posteriors in parametric form. Some of the many characteristics that render particle filters superior to the previous works listed, are the following [8]:

- Particle filters can accommodate (almost) arbitrary sensor characteristics, motion dynamics and noise distributions.
- Particle filters are recognised universal density approximators, that feature less-restrictive formations of posterior density when compared to parametric approaches.
- Particle filters allocate computational resources in areas of the posterior density that are deemed most accurate,

through sampling in proportion to the posterior likelihood (belief).

- ❖ Particle filters are capable in adapting to the quantity of computational resources available for their execution by managing the number of samples online.
- ❖ Particle filters are regarded as easy to implement, which defines them as an attractive paradigm for addressing mobile robot localisation.

However, particle filters also entertain many deficiencies for the application of mobile robot localisation, which derive from the stochastic nature of density approximation offered; some of said deficiencies in focus of MCL, are exactly:

- ❖ If the sample set size is small, a well-localised robot may lose track of its position due to MCL failing to generate a sample in the corresponding location of the robot's environment.
- ❖ If there are no samples nearby the robot's new pose after being kidnapped, the regular MCL algorithm is unable to combat the kidnapped robot problem.

Beyond the scope of the work presented, exists an "adaptive sampling scheme" [1], renowned as Adaptive Monte Carlo Localisation (AMCL), that determines the number of samples drawn from the distribution "on-the-fly" to represent the robots pose; this scheme purposes to "trade-off" the computational expense of the algorithm's execution, for a depreciating accuracy of representation of the robots estimated pose. Resultingly, MCL operates with a constant sample count that is computational unnecessary overtime and consequentially costly, whereas AMCL uses more "samples during global localisation when they are most needed" and fewer when tracking the robot, as the position of the robot becomes "approximately known". Hence adaptive.

Particle Filter (Dieter Fox) Algorithm Pseudocode

1. **Inputs**: $S_{t-1} \leftarrow \{\langle x_{t-1}^i,\ p_{t-1}^i\rangle \mid i = 1, 2, \dots, n\}$
2. $S_t \leftarrow \emptyset$
3. $\alpha \leftarrow \emptyset$
4. **for** $i = 1, \dots, n$ **do**
5. Sample index j from discrete distribution given by weights in S_{t-1}
6. Sample x_t^i from $p(x_t \mid x_{t-1}, u_{t-1})$ conditioned on x_{t-1} and u_{t-1}
7. $p_t^i \leftarrow p(z_t \mid x_t^i)$
8. $\alpha \leftarrow \alpha + p_t^i$
9. $S_t \leftarrow S_t \cup \{\langle x_t^i,\ p_t^i\rangle\}$
10. **end for**
11. **for** $i = 1, \dots, n$ **do**
12. $p_t^i \leftarrow \frac{p_t^i}{\alpha}$
13. **end for**
14. **Output** S_t sample set (posterior distribution)

A. Monte Carlo Localisation

MCL is in essence a recursive Bayes filter, which can be utilised to estimate the "posterior distribution of robot poses conditioned on sensor data" [2]; sensory data is surveyed by the robot throughout its exploration routine in the environment that it is exposed to. Summarily, Bayes' filters address the problem of "estimating the state x of a dynamical system from sensor measurements". Providing the nature of the work presented, the dynamical system can be acknowledged as the relation between the mobile robot and its environment, where the state x, characterises the "robot's pose therein (often specified by a position in a two-dimensional Cartesian space and the robot's heading direction)". Fundamentally, Bayes' filtering aims to estimate the state x, or a probability density, "over the state space conditioned on the data" concerning all previous observations and measurements of the robots odometry and its environment, which is the posterior renowned as belief (posterior belief), that can be given by a probability density function (PDF), characterised as such:

$$Bel(x_t) = p(x_t \mid z_t, u_t)$$

Where x denotes the state or pose of the robot, x_t is the state of the robot at time t, such that the states representation can be characterised as $x_t = \langle x_t, y_t, \theta_t\rangle$, given all past sensory perceptions about the robot's system (control measurements) $u_{1:t} = \{u_1, u_2, \dots, u_t\}$ and all sensory perceptions about the robot's environment (observations) $z_{1:t} = \{z_1, z_2, \dots, z_t\}$; both of which measurements contain uncertainties. At time t, the posterior PDF can also be characterised as:

$$Bel(x_t) = p(x_t \mid z_t, u_{t-1}, z_{t-1}, u_{t-2}, \dots, u_0, z_0)$$

Simply, the posterior distribution (belief) is represented by n weighted, random samples or particles $S = \{S_i \mid i = 1, 2, \dots, n\}$, which as mentioned and denoted above, are each comprised of a position $\langle x, y\rangle$ and orientation (heading) θ, with a discrete likelihood or probability (belief) of $p \geq 0$; at time t, this is collectively denoted as $S_t = \{\langle x_t^i,\ p_t^i\rangle \mid i = 1, 2, \dots, n\}$, where x_t once more represents the state or pose of the robot at the given interval. Derived from the Markov world assumption, Bayes' filters assume that the "environment is Markov, that is, past and future data are (conditionally) independent if one knows the current state" of the robot x_t. When accounting for both Bayes rule and the assumption said, posterior belief can then be characterised as:

$$Bel(x_t) = \alpha p(z_t \mid x_t)\ \overline{Bel}(x_t)$$

Where the term:

$$\overline{Bel}(x_t) \leftarrow \int p(x_t \mid x_{t-1}, u_{t-1})\ Bel(x_t)\ dx_{t-1}$$

represents the probabilistic model of system dynamics, or prediction, or motion model alternatively, given that it "reflects the state transition due to robot motion" [9]; this model considers the detected change in the sensory perceptions of the robot's system u_t, captured by its odometry sensors (prediction phase). Whereas the term $p(z_t \mid x_t)$ denotes the probabilistic model of perceptions, or correction, or sensor model instead, given that it "incorporates sensor readings to update the robots state" x, which encompasses the calculation of the likelihood p of the robot making observation z_t (correction phase). Meanwhile, α denotes the normalisation constant used to ensure that the integral of the posterior belief is normalised to the value of one; it is necessary for the importance factors to be normalised and sum to the value of one, so that they "define a discrete probability distribution" [2]. In acknowledgement of the models mentioned, the recursive state of Bayes' filters can be componentised into the following, identified phases:

Prediction – at each timestep t, a set of samples S_t are drawn from the "previously computed sample set" [1] S_{t-1}, relative to the likelihood of each states "p-value", in response to the robot traversing and its pose requiring to be reapproximated after the "motion command" is executed. Here the probabilistic model of system dynamics is applied to each state or particle comprising the posterior distribution $Bel(x_t)$ by sampling from the density $p(x_t \mid x_{t-1}, u_{t-1})$ [8]; this is conditioned by the robots control measurements u_t.

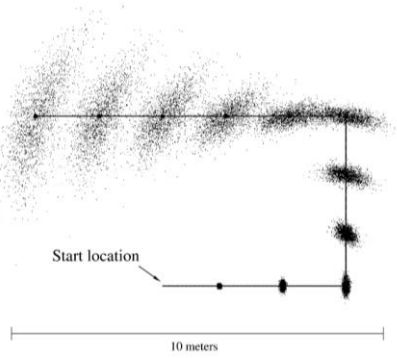

Figure 3: Visualisation of sample-based density approximation, for compiling the posterior belief of a non-sensing robot, using motion model (odometry sensor) measurements only [2].

Correction – at each timestep t, the robot's sensory observations of the environment z_t are then factored, for re-weighting each of the samples in the set S_t, relative to the probabilistic model of perceptions $p(z_t \mid x_t)$; this determines each particles likelihood of accurately representing the current pose of the robot x_t, given the evidence z_t collected. Upon sample set S_t being re-weighted, the posterior distribution can then be corrected and converge, via resampling from the set using a selection mechanism, that selects "higher probability samples that have a high likelihood associated with them" [8]; this formulates an updated set of samples S_t, that better (more densely) approximates the pose of the robot at time t, relative to its environment.

Proceeding from the posterior distribution being updated, both prediction and correction phases of MCL are repeated, recursively, to address the local tracking problem of mobile robot localisation. Preliminarily, the "initial belief characterises the initial knowledge about the system state" [2] x, however, in the absence of said knowledge, the

posterior distribution (particle filter) is typically initialised by "a uniform distribution over the state space", given that the probability of the robot existing in the environment with all possible poses is initially equivalent.

Monte Carlo Localisation (MCL) Algorithm Pseudocode

1. $\bar{x}_t \leftarrow \emptyset$
2. $x_t \leftarrow \emptyset$
3. **for** $i = 1, \dots, n$ **do**
4. $\quad x_t^i \leftarrow motionModelUpdate(u_t, x_{t-1}^i)$
5. $\quad p_t^i \leftarrow sensorModelUpdate(z_t, x_t^i)$
6. $\quad \bar{x}_t \leftarrow \bar{x}_t \cup \{\langle x_t^i, p_t^i \rangle\}$
7. **end for**
8. **for** $i = 1, \dots, n$ **do**
9. $\quad$ draw x_t^i from $\bar{x}_t$ with probability $\propto p_t^i$
10. $\quad x_t \leftarrow x_t + x_t^i$
11. **end for**
12. **Output** x_t robot state (posterior distribution)

III. Software Implementation

Aligned with the functional operations of the MCL algorithm explored, mobile robot localisation via MCL is achieved using three distinct procedures; one of which methods purposes to instantiate and initialise the particle filter, while another resamples and updates the particle filter, recursively, and the remaining method estimates the pose of the robot, via the posterior distribution populated, in cooperation with data clustering techniques. In the proceeding passages, identifies the implementation of the MCL algorithm proposed, in relation to the targeted development platform, RoS [4].

A. RoS Terminal Instruction

For executing the software submitted, below, features the ordered series of command-line instructions required to evaluate and deploy the Pioneer P3-DX mobile robot, with localisation behaviours native to the MCL algorithm implemented.

RoS Command-line Instruction Sequence

Command shell one:
Installing project dependencies (RoS Melodic)
$sudo apt install ros-$ROS_DISTRO-pr2-teleop ros-$ROS_DISTRO-joy ros-$ROS_DISTRO-slam-gmapping ros-$ROS_DISTRO-map-server
Build the package
$cd catkin_ws/src/
$git clone https://github.com/justagist/socspioneer.git
Build the catkin workspace
$catkin_make

Command shell two:
Run a simulated environment and open a previously saved map
$cd catkin_ws/
$source devel/setup.bash
$roscore

Command shell three:
Run the RoS visualisation tool (RVIZ)
$rosrun rviz rviz
Open the graphical tools suite to load RVIZ and the map server

Command shell four:
Start the simulation with a robot, obstacle and provided world
$cd catkin_ws/src/socspioneer/data
$rosrun stage_ros stageros lgfloor.world
Key R can be used to toggle 2D and 3D perspectives of the world
Key D can be used to toggle the laser field-of-view visualisation

Command shell five:
Navigate the robot in the simulated environment
$roslaunch socspioneer keyboard_teleop.launch
Keys WASD translates the robot multi-directionally
Keys QE orientates the robot bidirectionally

Command shell six:
Load the map of the provided world from a map server
$cd catkin_ws/src/socspioneer/data
$rosrun map_server map_server lgfloor.yaml

Resultingly, a map series appears, originating from the map server [11] and RVIZ sessions [12] instructed for execution.

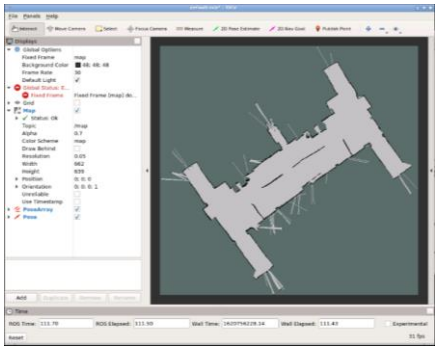

Figure 4: Illustration of the RoS visualisation tool: RVIZ [12], displaying occupancy map topic data that depicts the robot's environment, upon session initialisation.

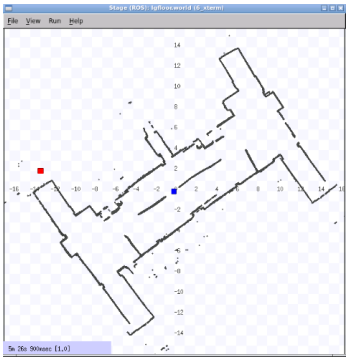

Figure 5: Map server [11] visualisation, displaying the prebuilt 'lgfloor.world' environment, that features a red-indicated obstacle, blue-indicated mobile robot, and black-indicated structural arrangement of the world.

B. Particle Filter Initialisation

Extending the state of the software package provided to facilitate the implementation of the MCL algorithm, and in focus of the subclass: PFLocaliser, that extends its base class: PFLocaliserBase, preliminarily, the parameters associated with an instance of the particle filter required independent variable declarations, for operation. Appropriately, this is addressed within the classes' constructor method, thus enabling all other methods contained within the class to share equivalent access rights, which allows the values of the parameters to be multipurposed and to remain consistent across multiple methods, instead of their existing multiple declarations of the same variable(s). Therefore, within the method bespoken, contains the declarations and initial configurations of the variables constituting to the MCL algorithm. In which features the noise coefficients of the robot's motion model, that as "compensation" [9] factors, aim to model uncertainty in the measure of the robot's motion overtime; accordingly, each of the parameter's values are initialised by "a uniform distribution over the state space" [2], as no prior knowledge of the robot's state would yet be known. Additionally, there also exists variable declarations for particle resampling noise coefficients, that are randomly derived from uniform distributions, to offset the posterior distribution of the filter, in attempt to mitigate the presence of the kidnapped robot problem [9] initially, and to account for translational changes in the robot's position, whilst the resampling procedure executes overtime. Therein, also defines the active clustering technique applied for estimating the robots pose, as well as the constant count of online particles, representing the posterior distribution of the filter, and the number of sensory readings that are considered and compared with the predictions conditioned by the robots control measurements u_t, when computing the particle weights in the algorithm's correction phase.

Figure 6: Code listing, visualising the '__init__()' method declaration, located within the PFLocaliser subclass.

For instantiating an instance of the particle filter, there exists a method, namely 'intiialise_particle_cloud()'; per the title appointed, the method functions to initialise the filters posterior distribution or the poses of particles (samples) comprising a cloud-like formation, upon the starting location of the robot in the corresponding environment being declared or updated, or alternatively when another occupancy grid map instance is instantiated. Operationally, the method conforms to instantiating a series of particles that comprise the particle cloud via a 'for-loop', for the length of the online particle count specified. Therein, each of the particles or pose objects are normally sampled a position, that is affected (multiplied) by the translational (x axis) and nautical (y axis) noise amounts sampled by the initial, uniform distributions, as well as an orientation, too affected by a uniform amount and sampled from a normal distribution but of an angular displacement type alternatively. Where the affecting amounts attempt to model the uncertainties presented by the robot's motion-tracking (odometry) capabilities and changes in the robot's position whilst the posterior distribution is being resampled, that the algorithm employs for achieving a relatively accurate state representation. To model a Gaussian-type distribution from the circular-orientation of angular displacement, rotational noise is alternatively sampled from a von Mises distribution [13], that is also "known as the circular normal distribution", on the interval $[-\pi,\ \pi]$; this was more appropriate, given that it is the circular analogue of normal distribution.

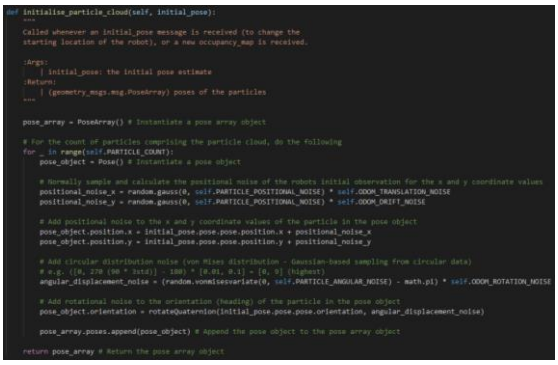

Figure 7: Code listing, visualising the 'initialise_particle_cloud()' method declaration, located within the PFLocaliser subclass.

Noise parameters sample from larger uniform distributions initially, to address the probabilistically equivalent potential of the robot existing within the structural arrangement of the environment, with all possible poses; thus, particles are vastly, uniformly spread and do not form cluster-like densities. This presents MLC's ability to represent "multimodal probability distributions" [8], which as a "precondition for localising a mobile robot from scratch", better combats the likelihood of the kidnapped robot problem, as the posterior distribution converges overtime to achieve localisation; respectively, the noise amounts downscale with convergence.

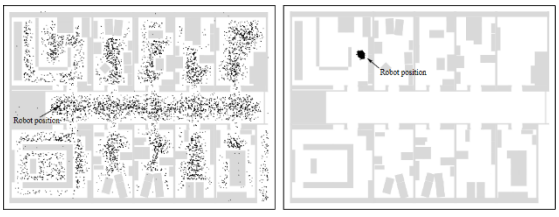

Figure 8: Visualisation of a particle cloud or global localisation initialisation (left), transitioning to the convergence state of achieving localisation of the mobile robot (right), overtime [1].

In correspondence with the values targeted for each parameter's configuration, elected as the initial upper and lower bounds of each motion model noise parameter, the range $[0.01, 0.3]$ was provided, in which any value "within the given interval is equally like to be drawn" [14] from. An insignificant range of values is justified, as motion model noise is considered a scaling factor for the predicted positions and orientations of the robot, which are assumed to be somewhat feasible prior; this is supported by global position estimation only requiring to accommodate for "small errors in its odometry as it moves" [1] throughout the environment, given the maximum velocity permitted by the robot's mechanical composition [3]. Meanwhile, as the initial upper and lower bounds of the particle resampling, positional noise parameter, the range $[75, 100]$ was configured, to disperse the particles considerably far from the origin of the initial pose estimate, relative to the size of the environment provided; as a "precondition for localising a mobile robot from scratch" [8], a sizeable spread is believed to enhance the global localisation of the robot, when the initial pose estimate is arbitrarily selected in the environment. Whereas for the initial upper and lower bounds of the particle resampling, rotational noise parameter, the range $[1, 120]$ was targeted, to consider all possible starting orientations of the robot's state, according to the empirical rule [15], which states that "for a normal distribution, almost all observed data will fall within three standard deviations of the mean or average". Furthermore, as the sample set size, representing the count of particles comprising the cloud or posterior distribution, a constant value of two-hundred is provided, which sensibly addresses the desired, initial, uniform dispersal of said particles respective of the environments size, whilst being more computationally efficient than other counts proposed [2][8][9]. As well, when computing the particle weights in the algorithm's correction phase, ninety sensory readings are accounted for as the robots' observations z_t, to enable the comparison between the state predictions conditioned by the robots control measurements u_t, to be better evaluated and therefore yield more accountability than what a smaller amount could achieve; whilst remaining computationally inexpensive to calculate.

C. Particle Filter Revision

Encapsulating the correction operations of the MCL algorithm, exists an independent method, namely 'update_particle_cloud()'; implied by its title, the method functions to filter the particles comprising the posterior distribution recursively, overtime, through a selective resampling procedure that refines the distribution of the posterior belief gradually, to succumb to a convergence state that more accurately represents the localised pose of the robot. Given its recursive nature, accomplishes local position tracking that is otherwise sought to be a problematic behaviour to attain for mobile robot localisation [8]; as a functionally emulating call-back method, the operations contained are only invocated upon the robot recording new observations of its environment, that are published to the laser scan topic, as scan data. Operationally, the method is comprised of two subsidiary functional blocks, each of which are sensibly abstracted into independent methods that can be invocated via function call statement; this componentised-driven design "helps encapsulate" [16] the behaviours of the algorithm, in favour of code base expansions and maintaining system robustness. Summarily, the initial functional block of the two performs the operations of a fitness (probability) proportionate selection algorithm, namely roulette-wheel selection [17], which purposes to resample the posterior distribution by formulating particles with new positions and orientations, derived from the probabilities of selected particles comprising its prior state. Meanwhile, the remaining functional block applies resampling noise to each of the particles constituting to the current-state cloud, to maintain a degree of dispersal that enables the posterior distribution to update and track changes in the robot's position, mostly, whilst it is being resampled. Preceding each's invocation, it is salient to compute and accumulate each particles likelihood (weight) of representing the state of the robot, accurately, relative to its perceptions of the surrounding environment, given the unreliable and noisy nature of odometer measurements [18]; fundamentally, the particle filter concerns the robot's observations relative to the perceptual model, to confirm the pose estimated by the robot's motion model. This subroutine is orchestrated by a 'for-loop', used to iterate through each particle representing the cloud, and targets the 'get_weight()' method defined in the SensorModel class, to compute each passed particles importance weight, relative to the latest laser scan data that is published to the aforementioned topic.

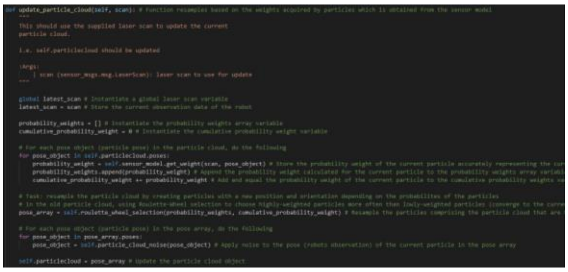

Figure 9: Code listing, visualising the 'update_particle_cloud()' method declaration, located within the PFLocaliser subclass.

Roulette-wheel selection – a stochastic selection mechanism, where the "probability for selection of an individual is proportional to its fitness" [17] or importance weight; the method is inspired by life-like roulette-wheels but dissimilarly, each of the slots comprising the wheel are weighted, such that the "larger the fitness of an individual is, the more likely is its selection". Therefore, it can be assumed that the important weight of a particle is "proportional to its likelihood of selection"; given n particles comprising the posterior distribution $Bel(x_t)$, the summation of their probability weights $\sum_{i=1}^{n} p_i$ as previously quoted, equates to the value of one, when each is normalised to the interval $[0, 1]$. Thus, a discrete distribution is formed; the algorithm targets higher-weighted particles more often that lower-weighted particles to achieve denser distributions and subsequently, more refined global estimates overtime, per resampling cycle surpassed.

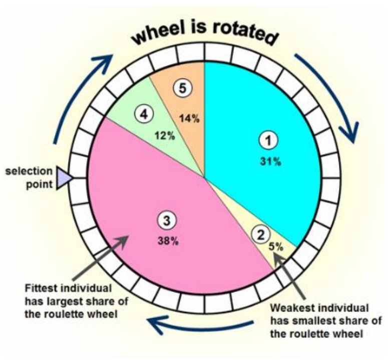

Figure 10: Graphical depiction of the roulette-wheel algorithm [19].

Modelled by a 'for-loop', initially a stop criterion is uniquely defined for every particle that is resampled, as a uniformly randomised percentile of the interval $[0, 1]$ or the summation of all importance weights; this is achieved via a pseudo-random number generator and represents the sum of individual importance weights to be exceeded, before a particle is selected. Upon particle selection, the corresponding particle survives to the "next generation" [19], for which it is appended to the current-state cloud representing the posterior belief of the robot. This iterative procedure recurs until the particle cloud constitutes n samples, which in the case of the online particle count configured, is terminated after two-hundred iterations.

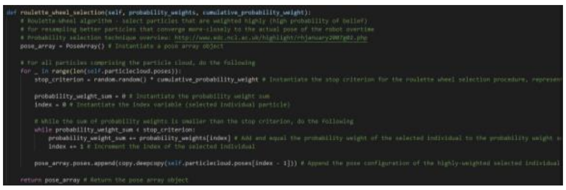

Figure 11: Code listing, visualising the 'roulette_wheel_selection()' method declaration, located within the PFLocaliser subclass.

Uncertainty modelling – to model displacements in the robot's position and orientation, that may have occurred since the resampling procedure of the particle cloud was executed, motion model and resampling noise, as translational, nautical, and rotational types, is applied to each of the particle poses comprising the current-state cloud, to maintain a degree of dispersion that allows the posterior distribution to update and reflect deviations in the robot's state, overtime. Operationally, the method gradually decrements the values that were originally configured for the motion model and particle resampling noise parameters, for emulating a progressive state of convergence and localisation of the robot, whilst jointly catering for potential incidences of the kidnapped robot problem, initially. Upon predefined threshold constants (arbitrarily appointed) being surpassed by the values of each of the parameter's concerned, as addressed by a series of 'if-else' statements, each parameters value is then resampled from its corresponding uniform distribution, to retain insignificant volumes of variance, that appropriate the state of the robot becoming "approximately known" [1] and thus localised overtime. Beyond their resampling, each resampling noise parameter is then reapplied as a deviation factor, like within the initialise particle cloud method, for the Gaussian distributions that are used to resample the positional and rotational noise coefficients, representing the magnitude of potential displacement from the robot's current state. These samples are also affected by the robot's motion model noise parameters, to account for potential odometrical miscalculations, involved in the displacement forecast for the robot's state. Once calculated, position-derived noise is then appended to the position of the corresponding particle in the cloud, whereas rotational-derived noise is separately injected, using the 'rotateQuaternion()' method located within the 'util.py' file, to affect the orientation of the particle. Notably, the interval for each uniform distribution of each noise parameter is arbitrarily settled, to sensibly reflect the potency of noise derived from odometer calculations and displacement in the robot's state, in consideration of the robot's peak traversal capabilities [3].

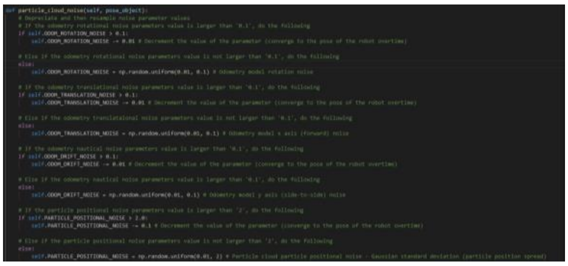

Figure 12: Code listing, visualising the primary section of the 'particle_cloud_noise()' method declaration, located within the PFLocaliser subclass.

Figure 13: Code listing, visualising the secondary section of the 'particle_cloud_noise()' method declaration, located within the PFLocaliser subclass.

D. Robot Pose Estimation

In acquirement of a global estimate of the robot's pose, relative to the current-state particle cloud, exists a method, namely 'estimate_pose()'; per the title specified, the method functions to provide a position and orientation estimate of the robot, relative to the poses of particles featured in the particle cloud or posterior distribution. Operationally, the method achieves global estimation via an array of data clustering techniques, that are used to sample particles conditioned by their fitness and other existential properties, in attempt to represent the robots state most-accurately; implemented as independent functional blocks, each of the data clustering techniques contained within the method only operate in singularity and not in parallel. This is addressed by a series of 'if-else' statements, for which the active clustering technique is conditioned by a string variable, representing the linguistic label associated to one of the pose estimation techniques available.

Global mean – an averaging operation that establishes the mean position and orientation of all particle poses comprising the particle cloud. Iteratively, the technique accumulates the position and orientation values of every particles pose in the current-state cloud, separately by their x, y, z and w axes and with the support of a 'for-loop' declaration, before each accumulation is then riven by the count of online particles configured. Hence global mean $x_t = \bar{x}_t$.

Figure 14: Code listing, visualising the 'estimate_pose()' method, global mean averaging operation, located within the PFLocaliser subclass.

As the technique does not adhere to clustering explicitly, given that their only exists one density of particles that the procedure operates upon, the method is inadequate for rejecting outliers and representing the state of the robot accurately. For which the technique assumes the estimated pose sensitive to ambiguities, derived from the structural symmetries of environments, which initiates particle cloud partitioning that resultingly forms multiple, concentrated densities of particles in distinctive locations in the bespoken environment. Providing the mean operation of the procedure proposed, the robots state would be falsely represented in a space central to all densities formed, where presumably, few or no particles are present; this endorses the techniques negligence in the submitted state of the MCL algorithm.

Figure 15: Visualisation of a particle cloud partitioned into separate, distinguished densities of particles, resulting from the structural symmetry of the environment exhibited [1].

Best particle – an importance weight filtering operation that establishes the particle appearing to represent the pose of the robot most-accurately, relative to all particle's important weights. In which the particle facilitating the highest belief factor is elected to model the estimated pose of the robot, as the 'best' or fittest candidate for state representation. Similarly, the technique iterates through each particle in the current-state cloud, via 'for-loop', to then be able to access each samples importance weight using the 'get_weight()' method previously applicated, which is required for the comparative nature of determining whether the current particles importance is greater than any of the particles iterated prior. Said comparison is sensibly addressed by an 'if-else' statement declaration, therein, the iterated particles x, y, z and w axis values are stored, if the particle is acknowledged better that the currently-known 'best' particle; the probability weight of the iterated particle is also registered, for the proceeding comparisons led by the iterative procedure. Upon the procedure's termination, the estimated pose of the robot inherits the position and orientation composites of the 'best' particle; given the singularity of the estimate, the trackability of the robot's state is difficult to isolate, in correspondence with the resampling frequency of the distribution. Also, the difficulty described is further contributed to in the presence of erroneous laser scan data, that cannot be mitigated, which renders the technique highly sensitive to false representation too. Thus, the technique was also abandoned from the configured state of the MCL algorithm.

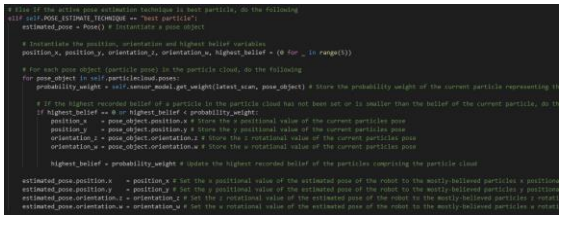

Figure 16: Code listing, visualising the 'estimate_pose()' method, best particle filtering operation, located within the PFLocaliser subclass.

Hierarchical Agglomerative Clustering – an agglomerative type hierarchical clustering algorithm, used to "group objects in clusters based on their similarity" [20], which as a bottom-up approach to data clustering, starts by "treating each object as a singleton cluster", before they are overtime merged "to create bigger clusters" [21] containing all objects, or particles comprising the cloud. The functional purpose of the algorithm when concerning the particle filter, is to identify and cluster similar particles in its distribution, based on their distance relations, such that a summed importance of each data partition can be calculated to identify the cluster facilitating the highest belief factor, that when averaged alike the operation of the global mean technique, will enable the robots state to be more-accurately represented. Operationally, with the support of a 'for-loop', the technique initially prepares the pose data of each particle in the current-state cloud, as a proximity or distance matrix [22], that requires the pose data to be transposed column-wise, to conform to the standard of. This is a formality that is imperative to the operations of the linkage algorithm elected, which is then utilised to "groups pairs of objects into clusters based on their similarity" [20], calculated from the distance matrix that is prepared and passed. Governing the similarity calculations, are a range of linkage methods [22] and associated distance metrics for agglomerative type clustering, which ultimately populates the clusters, where at each step of either algorithm, "the two clusters that are the most similar are combined into a new bigger cluster" [20]. Elected for the purposes of this work, the 'median' method, otherwise known as the Weighted Pair Group Method with Arithmetic Mean (WPGMA) algorithm [23], is targeted for computing the similarities between the particles and deriving the resultant cluster formations; the methods election was motivated by its centroid-averaging focus [22], which was sought accurate, as a representation of the mean of medians. Such that when two clusters are merged to form a new cluster composite, the average of their centroids provides the new clusters centroid, where the distance between the two clusters is correlated with the distance between their centroids [20]. Supporting the methods grouping operation, the median linkage method utilises the Euclidean distance metric, which "calculates the shortest distance between two points" [21]; given two vectors or points p_1 and p_2, the Euclidean distance between them is calculated as:

$$\sqrt{\left(x_{p_1} - x_{p_2}\right)^2 + \left(y_{p_1} - y_{p_2}\right)^2}$$

Proceeding from the grouping operations of the linkage method, the hierarchical clustering formations of the particle cloud are then encoded as a linkage matrix, representing the formations as a "hierarchical clustering tree" [20], which requires partitioning by a cutting operation, to instantiate the desired series of particle clusters. Specific to the method concerned, a threshold value is provided to define the "cut height" of the clustering tree, in which is used to segment the tree into "several groups" representing the clusters; relative to the count of online particles configured and their noise distributions, a cut height of '0.35' was passed to generally produce several clusters, that would assumably source the actual pose of the robot better than many or very few clusters alternatively. Advancing from the population of the clusters, each particle in the current-state cloud is then iterated through via 'for-loop', and its associated importance weight is then accumulated for the appointed cluster that it is identified as a member of; this is given by the flat cluster vector computed, containing the cluster identity that each particle or "observation" [20] belongs to. Upon amassing each cluster's posterior belief, the cluster with the highest belief factor that assumes to be the most-accurate state representation of the robot, can then be identified; using a 'for-loop', the position and orientation values of each particle constituting the cluster identified, can then be averaged via the series of accumulation and division operations explored within the global mean technique. The value computed thus represents the pose estimate of the robot, which in the presence of outlying and erroneous data types, can combat, given the formulation of multiple clusters, that mitigate the effects of ambiguities, through an average-selection scheme rather than strictly averaging the poses of the entire distribution. As well as mitigating sensitivity via centroid linkage [20], given the focus upon the point of intersection of all medians, which in the case of three clusters, performs as though the robots state is being triangulated. Providing these factors, HAC is elected as the active clustering technique in the submitted state of the MCL algorithm.

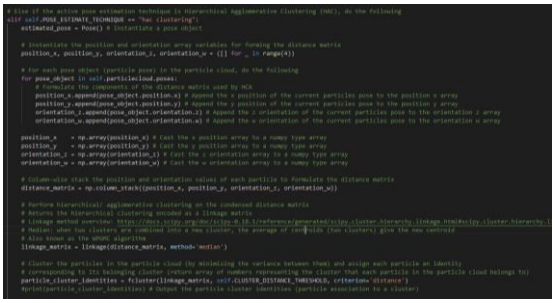

Figure 17: Code listing, visualising the primary section of the 'estimate_pose()' method, Hierarchical Agglomerative Clustering (HAC) algorithm, located within the PFLocaliser subclass.

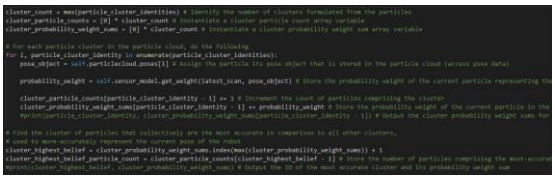

Figure 18: Code listing, visualising the secondary section of the 'estimate_pose()' method, Hierarchical Agglomerative Clustering (HAC) algorithm, located within the PFLocaliser subclass.

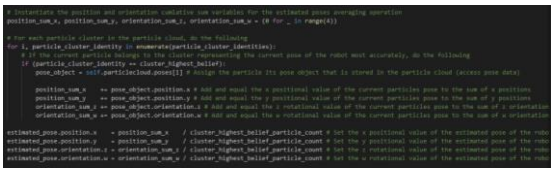

Figure 19: Code listing, visualising the tertiary section of the 'estimate_pose()' method, Hierarchical Agglomerative Clustering (HAC) algorithm, located within the PFLocaliser subclass.

From acquiring the pose estimate of the robot, the position and orientation values of the pose can then be output to a command shell window sponsored by the RoS development platform, per timestep surpassed. Unlike the robots position estimate, its orientation value is cast from quaternion to Euler notation, using the provided method 'euler_from_quaternion()'; this purposes to allow the orientation estimate of the robot to be acknowledged as a yaw coefficient, that is better understood as a measure of degrees, for which it is further casted from radians into, to fulfil.

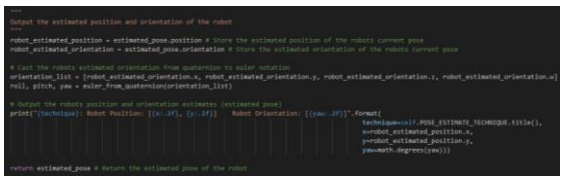

Figure 20: Code listing, visualising the 'estimate_pose()' method, console output configuration, located within the PFLocaliser subclass.

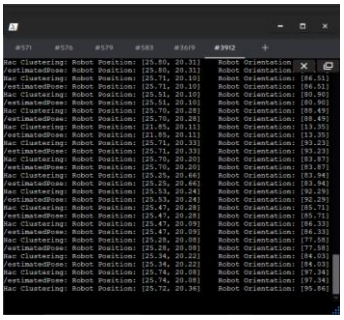

Figure 21: Command shell window visualisation, displaying the output representing the estimated pose of the robot, published to the 'estimatedPose' data topic and directly after HAC operations.

Additional to the textual output of the robots estimated pose, within the RVIZ visualisation tool of the RoS development platform, the estimated pose topic data is also displayed graphically, via a lone pose depicting an arrow; where the foot of the arrow represents the localised, position estimate of the robot, and its head representing the orientation (heading) estimate of the robot.

Figure 22: RVIZ tool visualisation, graphically depicting the estimated pose of the robot, published to the 'estimatedPose' data topic.

IV. Evaluation

A. Parametric Influence

Throughout the development cycle of the MCL algorithm, the number of laser sensor readings that the robot considers when updating its state via the probabilistic model of perceptions, was modified to establish balance between the accuracy of the robot's state representation and the computational expense incurred. As already mentioned within the earlier developments of this paper, for the configuration submitted, the robot utilises ninety prior observations to correct the poses of particles calculated by its prediction model; this has allowed the robot to appropriately evaluate its state predictions whilst remaining computationally inexpensive to calculate. Aligned within the RVIZ visualisation tool, the spread of the posterior distribution is seemingly less abnormal upon the robot traversing through its environment and whilst remaining stationary, compared to the original, twenty readings configured. Moreover, it can be observed that an accurate state of convergence is more-frequently achieved, which assumes that a higher observational capacity reduces the likelihood of false state representation via environmental ambiguities.

In focus of the count of online particles used to construct the particle cloud, to represent the state of the robot more-accurately, it was discovered that larger sample sets were beneficial; where trialled and under influence of time constraints, one-thousand particles were found to be the most triumphal in localising the robot with an unknown pose, initially, compared to the five-hundred, two-hundred (configured) and one-hundred counts tested also. This is assumed to be in effect of more particles occupying a similar space, that can better and more densely confirm the actual state of the robot. However, as previously told and in acknowledgement of a non-adaptive scheme, there evidently exists a trade-off between the count of online particles and computational expense incurred also. For which, a slightly depreciated performance is opted for, to preserve computational resources during the algorithm's execution, given that MCL operates with a constant sample count that is computational unnecessary overtime and consequentially costly. Therefore, a constant value of two-hundred is settled, that allows the desired, initial, uniform dispersal of the particles to be achieved still, respective of the environments size; relative to the RVIZ visualisation tool, the robot can be observed to overcome ambiguities caused by structural symmetries, rendering the configuration submitted suitable for deployment, despite the particle count re[resenting a reduced amount.

Relevant to the motion model and resampling noise parameter configurations, the values of said parameters were initially trialled as constants and were insignificant by value, given that the robots initial pose estimate could be manually instructed and once localised, tracked adequately. However, the configuration was observed to be prone to symmetrical ambiguities in the robots environment, for which the convergence state of the particle cloud could not revert to dispersion, to overcome falsely representing the robots pose; as well, in the presence of computational malfunction and timestep inconsistences caused by extended computational resource consumption, the particles when posing with insignificant noise coefficients, were often found to lag behind the real state of the robot, overtime. Providing these observations, each parameters value was adapted to being sampled from an independent normal distribution, to inherit variance and reserve a degree of state trackability, overtime, for overcoming the local tracking problem of mobile robot localisation and ambiguities caused by structural similarities of the robot's environment. In which, the distributions were initially

configured large for the resampling noise parameters, to address the probabilistically equivalent potential of the robot existing within the structural arrangement of the environment, with all possible poses; this decision targeted the kidnapped robot problem mostly, whilst nullifying the perceived need to manually define the initial pose estimate of the robot. Inevitably, the configuration was successful, as the posterior distribution was able to converge overtime to achieve localisation, for which is why the distributions are preserved in the submitted state of the algorithm. This design decision was accordingly complemented by a value reduction scheme, as mentioned in prior sections, to emulate a gradual state of convergence; otherwise, vast distributions would prevent the particles from ever converging and thus, localisation would not be possible. Meanwhile, bearing relations to the motion model noise parameter values, they too became to be sampled from uniform distributions, to inherit variance, also for motion-tracking purposes. However, the range of the distributions were preserved to be small, for factoring the maximum velocity permitted by the robot's mechanical composition, when predicting the magnitude of displacement in the robots pose, during its posterior distribution being resampled. Collectively, the parameter values revealed previously, enable the localisation state of the MCL algorithm to behave as desired, such that all the mentioned problems sponsored by mobile robot localisation, are alleviated.

Lastly, in acknowledgement of the HAC technique most-accurately representing the robots pose estimate overtime, the threshold constant representing the cut height of the hierarchical clustering tree produced by the algorithm was modified, to observe the effects of a higher and lower presence of particle clusters, in determining the robots estimated state. Per the command shell and RVIZ visualisation tool outputs configured, it was observed that using a mediocre amount of particle clusters better stabilised and represented the state of the robot, in comparison to both the significant and insignificant amount alternatives tested. Which is assumed to be the cause of catering for increased counts of outlying and erroneous data types and inversely, averaging fewer centroids to yield a representative point of intersection, which consequentially offsets the estimate from the real pose of the robot, more-frequently overtime, given each alternatives sensitivity prospect to sudden displacements in the robot's state. Thus, a mediocre amount conforming to the value of '0.35' was configured, as the best coefficient found to mitigate the pose estimates sensitivity, to sudden displacements in the robot's position and orientation; given the bottom-up approach defining the clustering technique, a smaller cut height coefficient respectively generates more clusters from the particle cloud.

B. Robot State Representation

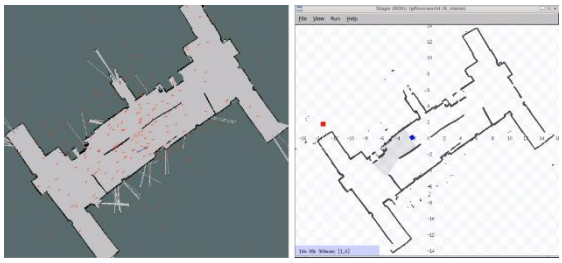

Figure 23: Particle filter initialisation visualisation, particles are uniformly distributed amongst the environment, initially.

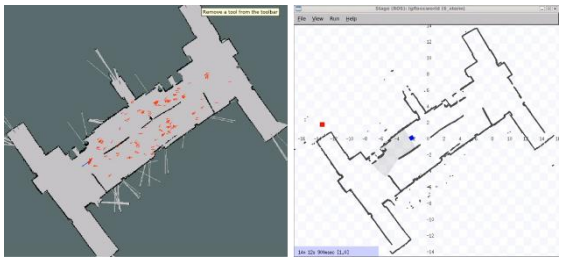

Figure 24: Particle filter converging visualisation, particles comprising the cloud form densities around regions promising to the robot's real state.

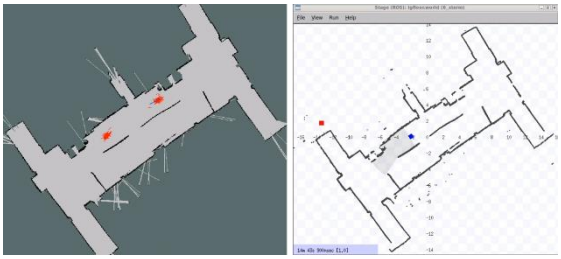

Figure 25: Particle filter ambiguity visualisation, particles formulate multiple densities reflecting uncertainty, concerning the structural symmetry of the robot's environment.

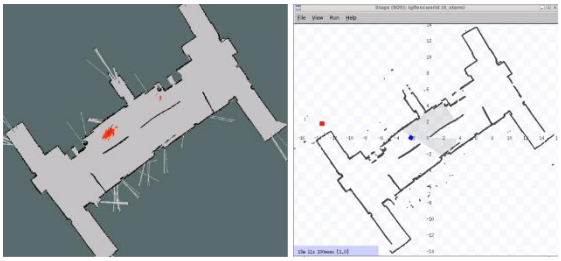

Figure 26: Particle filter localisation visualisation, particles converge to the most promising particle density overtime, becoming denser whilst the opposing density becomes more deprived.

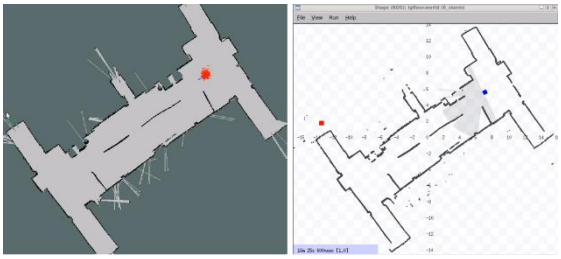

Figure 27: Particle filter local tracking visualisation, particles track the state of the robot upon converging to its pose estimate prior, as the state of the robot becomes approximately known.

V. Conclusion

This paper presented Monte Carlo Localisation (MCL); a probabilistic, sample-based algorithm appointed to the problem domain of mobile robot localisation and subsequently, target-tracking. Dissimilar to the prior approaches to mobile robot localisation submitted in the field, MCL applies a random sampling strategy equivalent to the procedure of a recursive Bayes filter, to represent the posterior belief of the robot, overtime. This presents a series of improvements over former methods, such that MCL typically poses a higher degree of accuracy within state representation by comparison, from the significant reduction in computational resource consumption, that once posed restraints on the number of observations or sensor readings factoring each pose estimate. As well, given its recognisably simple implementation, it is regarded a lot easier to implement, when compared to the prior Markov approaches to localisation. Providing the resampling scheme it incorporates, MCL overtime, can also, attractively favour likely states over unlikely ones, to achieve a convergence state that enables local tracking and a global estimate to be realised.

Aligned with the evaluation procedure and state representations explored for the implementation of the MCL algorithm presented, it is evident that the robot displays a competency to localise itself within a known environment. However, the particle filter also demonstrates vulnerability to environmentally-derived ambiguities, that consequently offset the robots pose estimate from its real state; if subjected to a dynamic and therefore partially-known environment, the robot is assumed inept of localising itself due to prospects of kidnapping.

In future works, it would be beneficial for adaptive schemes to have an increased presence, as well would three-dimensional models and other clustering techniques for better estimating the robot's state.

References

[1] Fox, D. and Burgard, W. and Dellaert, F (1999) Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In: *Proceedings of the National Conference on Artificial Intelligence, July 1999, Orlando, Florida, USA*. Berlin: ResearchGate, pp. 343-349.

[2] Fox, D. and Burgard, W. and Dellaert, F (2001) Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*. [Online] 128 (1-2). Available from: https://www.researchgate.net/publication/222559675_Robust_Monte_Carlo_Localization_for_Mobile_Robots [Accessed: 14/05/21].

[3] Génération Robots (2021) *Pioneer P3-DX mobile robot*. [Online] Génération Robots. Available from: https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html [Accessed: 14/05/21].

[4] ROS (2021) *ROS*. [Online] ROS. Available from: https://www.ros.org/ [Accessed: 14/05/21].

[5] ROS (2021) *Adept MobileRobots Pioneer and Pioneer-compatible platforms*. [Online] ROS. Available from: http://wiki.ros.org/Robots/AMR_Pioneer_Compatible#Laser_Rangefinders [Accessed: 14/05/21].

[6] Hanzel, J. and Kľúčik, M. and Jurišica, L. and Vitko, A. (2012) Range Finder Models for Mobile Robots. *Procedia Engineering*. [Online] 48. Available from: https://www.sciencedirect.com/science/article/pii/S1877705812045675 [Accessed: 14/05/21].

[7] Fabrizi, E. and Ulivi, G. (1997) Sensor Fusion for a Mobile Robot with Ultrasonic and Laser Rangefinders. *IFAC Proceedings Volumes*. [Online] 30 (7). Available from: https://www.sciencedirect.com/science/article/pii/S1474667017432903 [Accessed: 14/05/21].

[8] Dellaert, F. and Fox, D. and Burgard, W. and Thrun, S. (1999) Monte Carlo localization for mobile robots. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), Detroit, MI, USA, May 1999*. New York: IEEE, pp. 1322-1328.

[9] Bukhori, I. and Ismail, Z. (2017) Detection of kidnapped robot problem in Monte Carlo localization based on the natural displacement of the robot. *International Journal of Advanced Robotic Systems*. [Online] 14. Available from: https://www.researchgate.net/publication/318354950_Detection_of_kidnapped_robot_problem_in_Monte_Carlo_localization_based_on_the_natural_displacement_of_the_robot [Accessed: 14/05/21].

[10] Rekleitis, I. (2004) A particle filter tutorial for mobile robot localization. [Online]. Available from: https://www.researchgate.net/publication/244978679_A_particle_filter_tutorial_for_mobile_robot_localization [Accessed: 14/05/21].

[11] ROS (2021) *map_server*. [Online] ROS. Available from: http://wiki.ros.org/map_server [Accessed: 14/05/21].

[12] ROS (2021) *rviz*. [Online] ROS. Available from: http://wiki.ros.org/rviz [Accessed: 14/05/21].

[13] NumPy (2021) *numpy.random.vonmises*. [Online] NumPy. Available from: https://numpy.org/doc/stable/reference/random/generated/numpy.random.vonmises.html [Accessed: 14/05/21].

[14] NumPy (2021) *numpy.random.uniform*. [Online] NumPy. Available from: https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html [Accessed: 14/05/21].

[15] Hayes, A. (2021) Empirical Rule. [Weblog] *Investopedia*. 5th March. Available from: https://www.investopedia.com/terms/e/empirical-rule.asp [Accessed: 14/05/21].

[16] Golan, E. (2014) Why Abstraction is Really Important. [Weblog] *DZone*. 3rd April. Available from: https://dzone.com/articles/why-abstraction-really [Accessed: 14/05/21].

[17] De Luca, G. (2020) Roulette Selection in Genetic Algorithms. [Weblog] *Baeldung*. 19th October. Available from: https://www.baeldung.com/cs/genetic-algorithms-roulette-selection [Accessed: 14/05/21].

[18] Martinelli, A. and Siegwart, R. (2003) Estimating the Odometry Error of a Mobile Robot during Navigation. [Online]. Available from: https://www.researchgate.net/publication/37441222_Estimating_the_Odometry_Error_of_a_Mobile_Robot_during_Navigation [Accessed: 14/05/21].

[19] Newcastle University (2021) *Roulette wheel selection*. [Online] Newcastle University. Available from: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php [Accessed: 14/05/21].

[20] Data Novia (2021) *Hierarchical Clustering in R: The Essentials*. [Online] Data Novia. Available from: https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/ [Accessed: 14/05/21].

[21] Maklin, C. (2018) Hierarchical Agglomerative Clustering Algorithm Example in Python. [Weblog] *Towards Data Science*. 31st December. Available from https://towardsdatascience.com/machine-learning-algorithms-part-12-hierarchical-agglomerative-clustering-example-in-python-1e18e0075019 [Accessed: 14/05/21].

[22] SciPy.org (2021) *scipy.cluster.hierarchy.linkage*. [Online] SciPy.org. Available from: https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage [Accessed: 14/05/21].

[23] SciPy.org (2021) *scipy.cluster.hierarchy.median*. [Online] SciPy.org. Available from: https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.cluster.hierarchy.median.html [Accessed: 14/05/21].