Development of a New Technique in ROS for Mobile Robots Localization in Known-Based 2D Environments

Dr. Iyad Hatem*
Mhd Ali Alshikh Khalil**

(Received 11 / 8 / 2021. Accepted 14 / 12 / 2021)

□ ABSTRACT □

Adaptive Monte Carlo Localization (*amcl*) is the only standard package for mobile robots localization in Robot Operating System (ROS). In this research, a new particle filter based localization technique named general Monte Carlo Localization (*gmcl*) was developed by adding three particle filter algorithms to *amcl* in order to improve its performance, so the new versions of ROS could be better invested in systems that depend on the knowledge of the robot's pose.

In addition, we compared *amcl* and *gmcl* in terms of computational complexity and the ability of addressing the pose-estimation problem in a differential drive mobile robot equipped with a LiDAR sensor. The results showed that the new proposed technique outperformed *amcl* in the accuracy of estimating the pose when compared to the same maximal computational workload. *gmcl* was able to reduce the pose-error in pose tracking and also able to increase the success rate of robot's pose detection in the two problems of global localization and kidnapped-robot.

Keywords: Particle filters, Robot operating system, Pose estimation, Monte Carlo Localization, *amcl*, Localization in ROS.

* **Associate Professor, Mechatronics Department, Faculty of Mechanical and Electrical Engineering, Tishreen University, Lattakia, Syria, iyadhatem@tishreen.edu.sy**

** **Postgraduate Student (Master), Mechatronics Department, Faculty of Mechanical and Electrical Engineering, Tishreen University, Lattakia, Syria.**

تطوير تقنية جديدة في نظام تشغيل الروبوت (ROS) لتموضع الروبوتات المتحركة في البيئات ثنائية الأبعاد المعلومة

د. إياد محمد حاتم*

محمد علي الشيخ خليل

(تاريخ الإيداع 11 / 8 / 2021. قُبِل للنشر في 14/ 12 / 2021)

□ ملخّص □

إن تموضع مونتي كارلو المتكيّف (amcl) هو الحزمة القياسية الوحيدة لتحديد تموضع الروبوتات المتحركة في نظام تشغيل الروبوت (ROS). في هذا البحث ، تم تطوير تقنية تموضع جديدة تعتمد على مرشحات الجسيمات سمّيت بتموضع مونتي كارلو العام (gmcl) من خلال إضافة ثلاث خوارزميات لمرشحات الجسيمات إلى amcl لتحسين أدائه و بالتالي إمكانية استثمار نظام تشغيل الروبوت مستقبلاً بشكل أفضل في الأنظمة التي تعتمد على معرفة موضع الروبوت.
بالإضافة الى ذلك أجريت مقارنة بين amcl و gmcl من حيث درجة التعقيد الحسابي والقدرة على معالجة مشكلة تخمين الموضع لروبوت ذو نظام حركة تفاضلي مزود بحساس ليزري LiDAR. حيث اظهرت النتائج تفوق التقنية المقترحة الجديدة على amcl في دقة تخمين الموضع عند المقارنة مع تثبيت عبء التشغيل الحسابي الأعظمي. حيث استطاع gmcl تقليل خطأ الموضع في تتبع الموضع وزيادة معدل نجاح اكتشاف موضع الروبوت في مشكلتي التموضع الشامل والروبوت المختطف.

الكلمات المفتاحية: مرشحات الجسيمات، نظام تشغيل الروبوت ، تخمين الموضع ، تموضع مونتي كارلو، amcl التموضع في نظام تشغيل الروبوت.

* أستاذ مساعد، قسم الميكاترونيك ، كلية الهندسة الميكانيكية والكهربائية، جامعة تشرين، اللاذقية ، سورية.
iyadhatem@tishreen.edu.sy
** طالب دراسات عليا (ماجستير)، قسم الميكاترونيك، كلية الهندسة الميكانيكية والكهربائية، جامعة تشرين، اللاذقية ، سورية.

Introduction:

Robot Operating System (ROS), described as a flexible open source framework for writing robot software, is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. It uses synchronous nodes to carry out processes that represent programs, and messages to exchange data between nodes [1]. It is already widely used in aerial, ground, manipulator and marine robotic platforms [2]. ROS is characterized by the increasing presence of software developers contributing to modeling and to writing software for large number of robots of all types including surgical, humanoid, space robotics, and autonomous cars [3]. It also achieves great compatibility with the famous Gazebo simulator [4], where it is possible to model a virtual work environment, to test and analyze robot algorithms in virtual environment before applying them to a real robot. Its framework is also integrated with industrial robotics through ROS Industrial project [5].

Mobile robot localization is the problem of determining the pose position and orientation of the robot against a known map. it can be seen as a transformation problem between coordinates and can be described as the process of creating a link between the map global coordinate system and the robot local coordinate system [6] (Figure (1)). Localization is often called pose estimation since probabilistic approaches are used to solve it, (in our case through particle filters [7]). Robot localization is considered as a fundamental perception problem in robotics and it is a requirement for performing other tasks such as path planning (Figure (2)), synchronizing movement with the movement of other robots to perform a specific task, avoiding obstacles and making decisions in certain cases. Therefore, accurate robot localization makes the effectiveness of these tasks improved and reduces errors and the possibility of failure.

The package *amcl* [8], which stands for Adaptive Monte Carlo Localization, is the only standard package in ROS for estimating the pose of mobile robots within a known 2D environment. It can be observed from literature that the new localization techniques improve solving at the most two of the three localization problems: Pose Tracking, Global Localization and Kidnapped-Robot Problems. Thus, the importance of this research emerges from developing a new technique to improve solving the three problems together with the ability to work in real time.

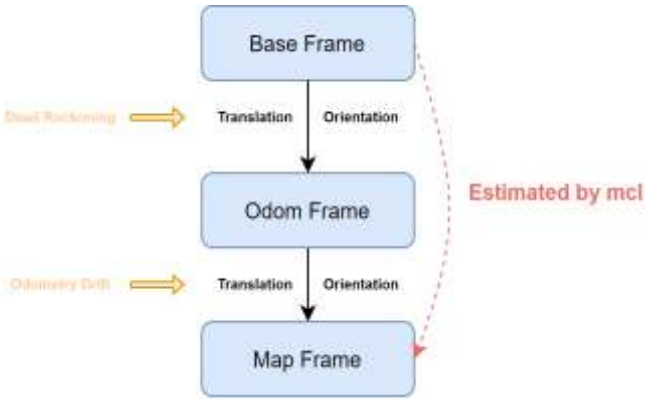


Figure (1) Monte Carlo Localization -*mcl*- estimating transformation between robot's base frame and map frame

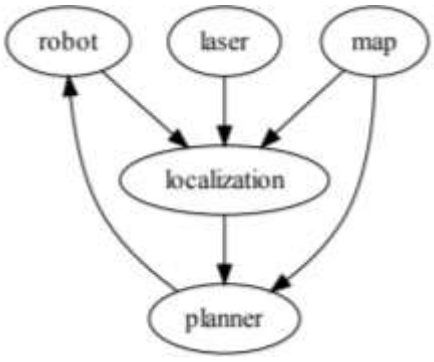


Figure (2) role of localization in path planning process

This article is divided into seven sections as follows: the first section provides an explanation of *amcl* technique, while the second contains comprehensive explanation of the proposed

gmcl technique with the modifications implemented on the selected particle filters. The third section presents an explanation of parameters added by *gmcl* to ROS parameter server, and in the forth section a complexity analysis of the new localization technique *gmcl* is presented. The experimental results of the two techniques -*gmcl* and *amcl*- with discussion are presented in fifth and sixth sections, respectively. The last section contains a conclusion to this article and recommendations.

1. Adaptive Monte Carlo Localization (*amcl*):

amcl is the standard package for mobile robot localization located in ROS Navigation Stack [9]. It utilizes *augmented_mcl* [10] with KLD-sampling particle filter [11]. *augmented_mcl* is used because *standard-mcl* algorithm [10] is unable to solve the kidnapped-robot problem. It is solved in *augmented_mcl* through spreading global particles randomly over the map in order to find the new robot's pose. The KLD-sampling particle filter is used to adapt the number of required particles to shape the pose-estimated distribution within acceptable error. This package has been used in many papers, such as [12, 13, 14, 15, 16, 17, 18, 19]. Table (1) has the pseudo code for this algorithm, which was derived from the attached code within the software package.

Table (1) *amcl* algorithm

1:	**Algorithm_amcl** (X_{t-1}, u_t, z_t, m)**:**
2:	$\bar{X}_t = X_t = \phi$
3:	for $m = 1$ to M_{t-1} do
4:	$x_t^{[m]}$ = sample_motion_model ($u_t, x_{t-1}^{[m]}$)
5:	$w_t^{[m]}$ = measurement_model ($z_t, x_t^{[m]}, m$)
6:	$\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7:	endfor
8:	compute w_{slow}, w_{fast}
9:	for $m = 1$ to M_t do
10:	with probability $max(0.0, 1.0 - w_{fast}/w_{slow})$ do
11:	draw $x_t^{[m]}$ from m with equal probability
12:	else
13:	draw $x_t^{[m]}$ from $\bar{X}_t$ with probability $\propto w_t^{[m]}$
14:	enddo
15:	add $x_t^{[m]}$ to X_t
16:	if $x_t^{[m]}$ falls into an empty bin b then
17:	$k := k + 1$
18:	$b := non - empty$
19:	$M_t = \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\right\}^3$
20:	endfor
21:	return X_t

The algorithm takes map m, control signals u_t, sensor reading z_t and particle set at previous time X_{t-1} as an input and outputs a new set of particles X_t. Computing of new pose and weight of particles are done through motion and measurement model, respectively (lines 3-7). *amcl* algorithm uses the *standard-mcl* approach in computing the new pose and weight of the particles. In (line 8) we compute w_{slow} and w_{fast}, which are short-term and long-term

averages of the measurement likelihood, respectively. The influence of theses variables can be seen in (line 10) where they are responsible of adding global particles to recover from kidnapped-robot problem. The calculation of these variables is shown in table (2), Whereα_{slow} , $\alpha_{fast}$are decay rates for the exponential filters. and they satisfy $0 \leq \alpha_{slow} \ll \alpha_{fast}$.

Resampling stage of *amcl* is in (lines 9-20). Particles of the new set$X_t$are sampled either with a random pose (line 10) or through roulette wheel of weights (line 13). KLD-sampling is implemented in (lines 16-19). KLD-sampling algorithm defines the number of required particles through maintaining the error value between true distribution and approximated distribution on a determinate distance called Kullback-Leibler Distance.

Table (2) computew_{slow}, w_{fast}algorithm

1: **Algorithm_compute$w_{slow}, w_{fast}(\bar{X}_t)$:**
2: static w_{slow}, w_{fast} 3: $w_{avg} = 0$ 4: for$m = 1$ toM_{t-1}do 5: $w_{avg} = w_{avg} + \frac{1}{M_{t-1}} w_t^{[m]}$ 6: endfor 7: $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$ 8: $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$ 9: returnw_{slow}, w_{fast}

2. General Monte Carlo Localization (*gmcl*):

The following paragraphs introduce our proposed technique for solving the localization problem.

2.1 Design Goals:

In the literature, many of particle filters have been developed to improve pose estimation. Every one of these filters improves at most two of the three localization problems. So we think that to improve localization for all three problems, two candidate filters at least are needed. In this article four different types of particle filters were adopted (three of them are new and one is already included in *amcl*) with the ability of turning one, two or all of them on or off when initializing *gmcl*.

The four candidate filters should solve all three localization problems in real time, where:

- Two candidate filters to improve pose tracking performance with possibility of increasing global localization success rate.
- One candidate filter to increase the success rate when dealing with global localization and kidnapped-robot problem.
- One candidate filter to reduce computational load by adapting the number of particles needed to represent the pose-estimated distribution.

Two candidates were chosen for solving the pose tracking problem. The reason emerges from the fact that pose tracking is the major problem in localization. In addition, after solving the two other problems, they transform to the pose tracking problem.

2.2 Candidates Selection Criteria:

In this research, several characteristics were adopted to select the candidate filters:

- Advanced and effective particle filters.

- Real-time particle filters.
- The ability to integrate filter algorithm with other filters.
- Possibility of applying filter to the global localization problem.

Taking into account the above characteristics, we selected the following filters:

- Optimal Particle Filter [20] and Intelligent Particle Filter [21] to improve pose tracking performance with the possibility of increasing global localization success rate.
- Self-Adaptive Particle Filter [22] to increase the success rate when dealing with global localization and kidnapped-robot problem.
- KLD-sampling Particle Filter to adapt the number of particles needed to represent the pose-estimated distribution.

We used the structure of *amcl* algorithm as a basis and then we added the remaining filters: Optimal, Intelligent, Self-Adaptive particle filters.

2.3 Modification and Integration of Candidate Filters:

Algorithms of Optimal, Intelligent and Self-Adaptive filters required modifications before integrating their algorithms with *amcl*. These modifications ensured the improvement of their computational load, took advantage of some *amcl* 's algorithms and accomodated filters for the localization problem.

2.3.1 Modified Optimal Particle Filter:

It can be seen from the algorithm presented in [20] that it uses rejection sampling method for concrete computing of particle pose in resampling stage. This method does not have a fixed run-time and, in the context of Localization, it is computationally expensive because the measurement model is calculated for each possible new pose of the particle in Δ.

$$\Delta = \frac{p\left(z_t|x_t^{[k]}\right)}{p_{max}(z_t|x_t)} \qquad (1)$$

The computational complexity imposed by the rejection sampling method is equal to $O(T_L \cdot R \cdot M)$, where:

T_L represents the computation time of the measurement model for the particle in its new potential pose.

R represents the number of attempts required to solve rejection sampling for one particle.

M represents the total number of particles.

For that reason, we replace rejection sampling method with a fixed-time computationally-inexpensive method by means of auxiliary particles. Equation (2) represents the pose of auxiliary particle $x_t^{[m,j]}$ that gives the largest measurement model value which represents the new pose of the particle $x_t^{[m]}$.

$$x_t^{[m]} = x_t^{[m,j]} \; ; \; y \in \{1,\dots,B\} \wedge \underset{y}{argmax}\left(p\left(z_t|x_t^{[m,y]}\right)\right) = \{j\} \qquad (2)$$

B is the number of auxiliary particles that each particle possesses.

2.3.2 Modified Intelligent Particle Filter:

The number of effective particles N_{eff}, which gives the index of the particle that contains the threshold weight W_T after arranging the particle weights in descending order, cannot be used with the localization problem. This is because the nature of the measurement model function dictates a small variance in particle weights, especially that resampling stage imposes equal weights on all drawn particles.

To solve this problem, we assume a constant threshold weight which is equal to the particle weights after the resampling stage:

$$W_T = W_{const} = 1/M \qquad (3)$$

M is the total number of particles.

whatW_{const}represents in the context of grouping stage of particles into small and large weight sets, is that particles that gained weight through the measurement model. The ones who gained weight after normalization are considered to be large-weight particles, while particles that lost weight through the measurement model (after normalization) are considered small-weight particles.

Setting the threshold weight to a constant value speeds up the execution of the algorithm by canceling the particle weights sorting process whose computational complexity is equal to$O(M^2)$when using selection sort [23] for example.

After grouping particles into small- and large-weight particle sets, we take only one third of small-weight particles and apply crossover and mutation model to them. The reason for choosing one third of the small-weight particles is due to the fact that in many cases this number is large, especially at the beginning of the operation. Therefore, it imposes a large computational load. Also, the crossing of some particles to the actual pose of the robot leads to the transfer of a larger number of particles in the following time steps through resampling.

Crossover model modifies the pose of small-weight particles with the help of large-weight particles and can be described by Equation (4):

$$x_{tS}^{i} = \alpha x_{tL}^{i} + (1-\alpha)\, x_{tH}^{i} \qquad (4)$$

$\alpha \in [0,1]$is crossover amount.

x_{tL}^{i} is the pose of i^{th}particle from small-weight particle setX_{tL} .

$x_{tH}^{i}$is the pose of randomly selected particle from large-weight particle setX_{tH} .

x_{tS}^{i} is the new pose after applying crossover model.

Mutation model on the other hand moves the pose of small-weight particle to a new, unknown pose to search for new results that may be better. This may perform well in solving global localization and kidnapped-robot problem and is described by Equation (5):

$$x_{tM}^{i} = \begin{matrix} 2x_{tH}^{i} - x_{tS}^{i} \; if \; r \leq pM \\ x_{tS}^{i} \; if \; r > pM \end{matrix} \qquad (5)$$

$pM \in [0,1]$represents mutation probability, while$r \in [0,1]$is chosen randomly.

x_{tM}^{i}represents the new pose after applying mutation model.

2.3.3 Modified Self-Adaptive Particle Filter:

Since *amcl* depends in its structure on *augmented_mcl* to solve kidnapped-robot problem, two proposed modifications can be implemented. First is to substitute the sion$max(0.0, 1.0 - w_{fast}/w_{slow})$ by the expression $w_t^{max} < \zeta$, which determines whether the robot has been kidnapped or not. This expression infer the suspicion of being kidnapped when it is greater than zero. The second modification is to cancel the ratio α, which contributes in defining a fixed number of global particles$N_G$in relation to the total number of particles N_T .In modified algorithm, the potential number of global particles is equal to that produced by *augmented_mcl* algorithm and it satisfies the expression:

$$N_G \approx max(0.0, 1.0 - w_{fast}/w_{slow}) \times M \qquad (6)$$

Where *M* is the total number of particles.

We consider this as a modified *augmented_mcl* where instead of spreading random pose particles over the free space of the map, the modified algorithm spreads particles on the Similar Energy Region (SER). As mentioned by authors, SERrepresents a set of energy cells that their energy is similar to senor reading's energy and can be calculated through algorithm mentioned in table (3).

Table (3)SERcalculator algorithm

1: **Algorithm_SER_calculator ($\bar{E}, z_t$):**
2: $SER = \phi$
3: for laser beam$i = 1$to$I$do
4: $a_i = 1 - z_t^i / z_{max}$
5: endfor
6: $e = \sum_{i=1}^{I} a_i$
7: normalize$e = \frac{1}{I} e$
8: for energy map cell$k = 1$to$K$do
9: if$\| e - e_k \| \leq \delta$then
10: $SER = SER + \langle x_k \rangle$
11: endfor
12: return SER

Where:

$e$is sensor reading's energy.

$e_k$is energy stored ink^{th}energy cell.

δ is the threshold value that consider if an energy cell is insideSERor not.

$x_k$is embedded pose ink^{th}energy cell.

2.3.4 Integration of Candidates:

After applying the modifications on the algorithms, we integrate them with *amcl*. This is done by exploiting Optimal and Intelligent particle filters for computing new pose and weight of particles, whereas Self-Adaptive and KLD-sampling particle filter are used in the resampling stage. Pseudocode of *gmcl* algorithm is given in table (4). The algorithm takes same *amcl* input parameters with energy map$\bar{E}$as an input and outputs a new set of cles X_t .Implementation of Optimal particle filter to compute new pose and weight of particles is done in (lines 4-11) with the help of auxiliary particles of number$B$.

Optimal particle filter in this context replaces *standard-mcl* approach used in *amcl* algorithm for computing new pose and weight of particles. Intelligent particle filter implementation can be seen in (lines 12-25), where classifying the particles into small-weight set$X_{tL}$and large-weight set$X_{tH}$is represented in (lines 12-18), and crossover and mutation models are represented in (lines 19-25). Crossover and mutation models represent the implementation of Equation (4) and (5) on one third of small-weight particlesM_t^l / 3, respectively. Based on the fact that the pose of particle has been changed, a re-computation of weight is necessary (line 23). We computew_{slow}andw_{fast}in (line 26) in similar way to *amcl*. Resampling stage of *gmcl* is in (lines 27-39). First the calculation of $SER$is done, which is explained in table (3), and the rest of the stage is done in similar manner as in *amcl*. Not to forget that *gmcl* adds global particles with poses drawn fromSER(line 30).

Table (4) *gmcl* algorithm

1: **Algorithm_gmcl ($X_{t-1}, u_t, z_t, m, \bar{E}$):**
2: $\bar{X}_t = X_t = X_{tH} = X_{tL} = \phi$
3: $w_{avg} = 0$
4: for$m = 1$toM_{t-1}do
5: for$n = 1$to$B$do
6: $x_t^{[m,n]}$ = sample_motion_model ($u_t, x_{t-1}^{[m]}$)
7: $p\left(z_t \mid x_t^{[m,n]}\right)$ = measurement_model ($z_t, x_t^{[m,n]}, m$)
8: endfor

9: $x_t^{[m]} = x_t^{[m,j]}\ ;\ y \in \{1,\dots,B\} \wedge \underset{y}{argmax}\left(p\left(z_t \mid x_t^{[m,y]}\right)\right) = \{j\}$
10: $w_t^{[m]} = p\left(z_t \mid x_t^{[m]}\right) = \sum_{n=1}^{n=B} p\left(z_t \mid x_t^{[m,n]}\right)/B$
11: endfor
12: for$m = 1$toM_{t-1}do
13: if $w_t^{[m]} < W_{const}$then
14: $X_{tL} = X_{tL} + \langle x_t^{[m]}, w_t^{[m]}\rangle$
15: else
16: $X_{tH} = X_{tH} + \langle x_t^{[m]}, w_t^{[m]}\rangle$
17: $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]}\rangle$
18: endfor
19: for$m = 1$ to$M_t^l\ /\ 3$do
20: draw$x_{tH}^{[m]}$from$X_{tH}$with equal probability
21: $x_{tS}^{[m]}$ = crossover_model ($x_{tL}^{[m]}, x_{tH}^{[m]}$)
22: $x_{tM}^{[m]}$ = mutation_model ($x_{tS}^{[m]}, x_{tH}^{[m]}$)
23: recompute$w_{tM}^{[m]}$
24: $\bar{X}_t = \bar{X}_t + \langle x_{tM}^{[m]}, w_{tM}^{[m]}\rangle$
25: endfor
26: computew_{slow}, w_{fast}
27: SER = SER_calculator ($\bar{E}, z_t$)
28: for$m = 1$ to$M_t$do
29: with probability$max\left(0.0, 1.0 - w_{fast}/w_{slow}\right)$ do
30: draw$x_t^{[m]}$from$SER$with equal probability
31: else
32: draw$x_t^{[m]}$from$\bar{X}_t$with probability $\propto w_t^{[m]}$
33: endwith
34: add$x_t^{[m]}$toX_t
35: if$x_t^{[m]}$falls into an empty bin$b$then
36: $k := k + 1$
37: $b := non - empty$
38: $M_t = \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\}^3$
39: endfor
40: returnX_t

3. General Monte Carlo Localization (*gmcl*) in ROS:

The proposed *gmcl* adds new parameters to ROS parameter server. These parameters are:

- $use\_optimal\_filter$(type: bool, default value: **false**)

When set to **true**, *gmcl* will compute new pose and weight of particles through auxiliary particles of the Optimal particle filter, otherwise it will compute pose and weight through *standard-mcl* approach.

- $use\_intelligent\_filter$(type: bool, default value: **false**)

When set to **true**, *gmcl* will crossover and mutate the pose of one third of small- weight particles through crossover and mutation models.

- $use\_self\_adaptive$(type: bool, default value: **false**)

When set to **true**, *gmcl* will spread global particles randomly on SER, otherwise it will spread global particles randomly over the free space of map.

◆ $use\_kld\_sampling$(type: bool, default value: **true**)

When set to **true**, *gmcl* will adapt the number of its particles, otherwise it will fix the number of particles to parameter $max\_particles$.

◆ $N\_aux\_particles$(type: int, default value: **10**)

Defines the number of auxiliary particles. Used in Optimal particle filter.

◆ $crossover\_alpha$(type: double, default value: **0.5**)

Specifies the amount of effect that pose of large-weight particle does to pose of small-weight particle. Used in crossover model in Intelligent particle filter.

◆ $mutation\_probability$(type: double, default value: **0.1**)

Specifies the occurrence probability of a mutation to pose of small-weight particle. Used in mutation model in Intelligent particle filter.

◆ $energy\_map\_resolution\_x$(type: double, default value: **0.2 meters**)

X-axis resolution of energy map. Used in Self-Adaptive particle filter.

◆ $energy\_map\_resolution\_y$(type: double, default value: **0.2 meters**)

Y-axis resolution of energy map. Used in Self-Adaptive particle filter.

◆ $energy\_threshold\_value$(type: double, default value: **0.05**)

Defines the upper limit in energy difference between energy map cells and sensor reading's energy to shape SER. Used in Self-Adaptive particle filter.

◆ $publish\_ser$(type: bool, default value: **false**)

When set to **true**, *gmcl* will displaySERin Rviz as a PoseArray.

The following flow chart in Figure (3) describes how the first four parameters affects the flow of *gmcl* algorithm:

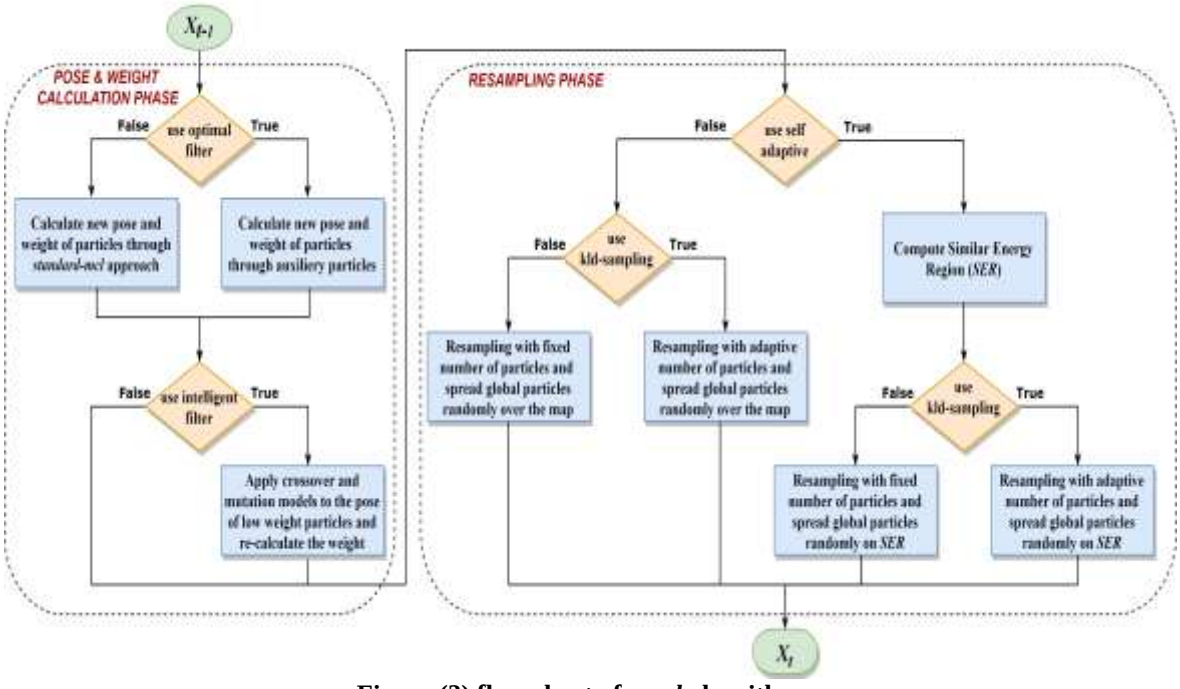


Figure (3) flow chart of *gmcl* algorithm

From this flow chart, we find that *gmcl* can run up to 16 different filter types, as shown in table (5). *gmcl* contains two other localization techniques and for that it is named General Monte Carlo Localization.

Table (5) name of localization technique based on embedded filter types

name	Optimal particle filter	Intelligent particle filter	elf-Adaptive particle filter	LD-sampling particle filter
augmented-mcl	0	0	0	0
amcl	0	0	0	1
gmcl	x	x	x	x

$x$stands for "Don't care" state.

Installation instructions, examples, *gmcl* requirements to run on ROS, *gmcl* subscribed and published topics, services and links to package code can be seen in Wiki page of *gmcl* located in ros.org site and can be accessed through link*.

4. Complexity Analysis:

It is essential when developing a new algorithm to study its computational complexity. We infer the time complexity as a Big O notation in table (6) for both *amcl* and *gmcl* algorithms given in table (1) and table (4), respectively.

Table (6) computation complexity of *amcl* & *gmcl*

Type	Big O notation
amcl	$O(T_L \cdot M_{t-1} + T_k \cdot M_t + T_o(N_t \cdot log(M_{t-1}) + (M_t - N_t)))$
gmcl	$\left(T_L\left(B \cdot M_{t-1} + M_t^l/3\right) + T_e \cdot K_e + T_k \cdot M_t + T_o(N_t \cdot log(M_{t-1}) + (M_t - N_t))\right)$

Where:

T_L Time required to compute measurement model for a particle.

T_e Time required to decide if an energy cell is within the Similar Energy Region or not.

T_o Time required to decide if the particle is the desired one or not.

T_k Time required to determine if the particle falls into an empty pin or not.

K_e Number of energy map cells.

M Total number of particles.

M^lTotal number of small-weight particles.

N Number of particles sampled via roulette wheel of weights.

B Number of auxiliary particles.

The following is a description of Big O notation for *amcl* and *gmcl* .

- **Complexity of *amcl*:**

In new pose and weight computation stage, the computational complexity lies in computing new weights for all particles, which is done through *standard-mcl* approach $O(T_L \cdot M_{t-1})$.

In resampling stage, the complexity of KLD-sampling lies in re-counting of empty pins after every newly added particle to the new particle set$O(T_k \cdot M_t)$ .The complexity of *augmented-mcl* lies in sampling local and global particles. Local particles are sampled via roulette wheel of weights and since roulette wheel implies Binary search algorithm, the complexity in sampling local particles is$O(T_o(N_t \cdot log(M_{t-1})))$ .On the other hand, global particles are sampled randomly from free space of the map and have complexity of$O(T_o(M_t - N_t))$.

- **Complexity of *gmcl*:**

In new pose and weight computation stage, the computational complexity lies in computing new weights for all particles in similar way to *amcl*. However, Optimal particle filter uses auxiliary particles to compute the weight of its particles, thus the computational complexity becomes$O(T_L \cdot B \cdot M_{t-1})$ .Also Intelligent particle filter adds complexity through re-

* **http://wiki.ros.org/gmcl**

computing the weight of small-weight particles that have taken part in crossover and mutation models, so the added complexity is$O(T_L \cdot M_t^l/3)$.
In resampling stage, the calculation ofSERin Self-Adaptive filter imposes complexity$O(T_e \cdot K_e)$. The rest of resampling stage is the same as in *amcl*, but global particles are sampled randomly from$SER$and have complexity of$O(T_o(M_t - N_t))$.

5. Experimental Results:

For experiments, the latest ROS 1 LTS version (Noetic Ninjemys) at the time of the research is used along with the virtual environment of the simulation program Gazebo on the famous Turtlebot 3 [24]. These experiments are conducted in the Mechatronics Department labs at the Faculty of Mechanical and Electrical Engineering at Tishreen University. Execution of software and performance evaluation are done on a computer with an Intel core i7 3rd generation processor and under the Ubuntu 20.04 operating system.
For fair comparison in performance of both techniques, we try as much as possible to equalize the maximum CPU usage that each demand while conducting the experiments. This is achieved by selecting parameters in Big O notation that approximate their algorithm computation time for both. The biggest factor that really decide computation time in Big O is timeT_L because *beam-range-finder* algorithm is implemented in both *gmcl* and *amcl* to compute particle weights. After neglecting other factors the approximation becomes:

$$O(T_L \cdot M_{t-1})_{amcl} \approx O\left(T_L\left(B \cdot M_{t-1} + M_t^l/3\right)\right)_{gmcl} \quad (7)$$

It could be observed, while operating *gmcl* at Turtlebot, that the number of small-weight particles that take part in crossover and mutation models is approximately15%of total number of particles:

$$M_t^l/3 \approx 0.15 \cdot M_{t-1} \quad (8)$$

The Equation (7) becomes:

$$O(T_L \cdot M_{t-1})_{amcl} \approx O\left(T_L \cdot M_{t-1}(B + 0.15)\right)_{gmcl} \rightarrow O(T_L \cdot M_{t-1})_{amcl} \approx O(T_L \cdot M_{t-1} \cdot B)_{gmcl} \quad (9)$$

To fix as much as possible the maximal computational workload we should fulfill:

$$M_{amcl} = M_{gmcl} * B \quad (10)$$

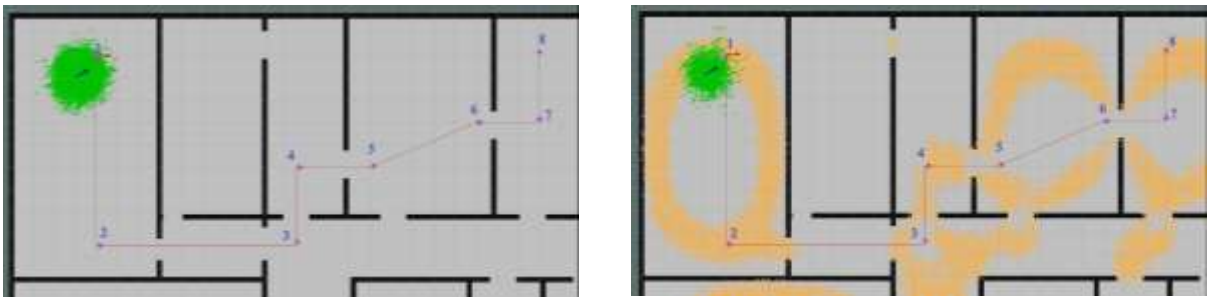
In our experiments, we define $M_{amcl}^{max} = 10000$, $M_{gmcl}^{max} = 2000$ and $B = 5$.
In the following paragraphs, we study the performance of *gmcl* and *amcl* in solving the three types of localization problems.

5.1 Pose Tracking:

The study of pose tracking problem implies some knowledge about the initial pose of the robot. The experiment entails running the robot along the path shown in Figure (4) from point 1 to 8, and studying how well did *gmcl* and *amcl* track the robot's pose. Tracking performance is evaluated by means of Root Mean Square (RMS) value for both error and variance of position and orientation for the whole experiment.
It is clear that conducting one experiment is not enough to evaluate the performance of both techniques, so we repeat the experiment several times and take the mean of the RMS value for both the error and the variance. The number of necessary repetitions follows the Standard Error of the Mean (SEM) [25]:

$$\bar{\sigma} = \frac{\sigma_1}{\sqrt[1]{n}} \quad (11)$$

σrepresents the standard deviation of RMS value set, where each RMS value results from an experiment iteration.
$n$represents number of experiment iterations.



(a) (b)

Figure (4) particles at start time when studying pose tracking. Path of experiment is presented by line connecting the eight points. Red arrow represents the true pose of robot, while blue arrow represents pose estimated by green particles. (a) represents *amcl* particles, (b) represents *gmcl* particles.

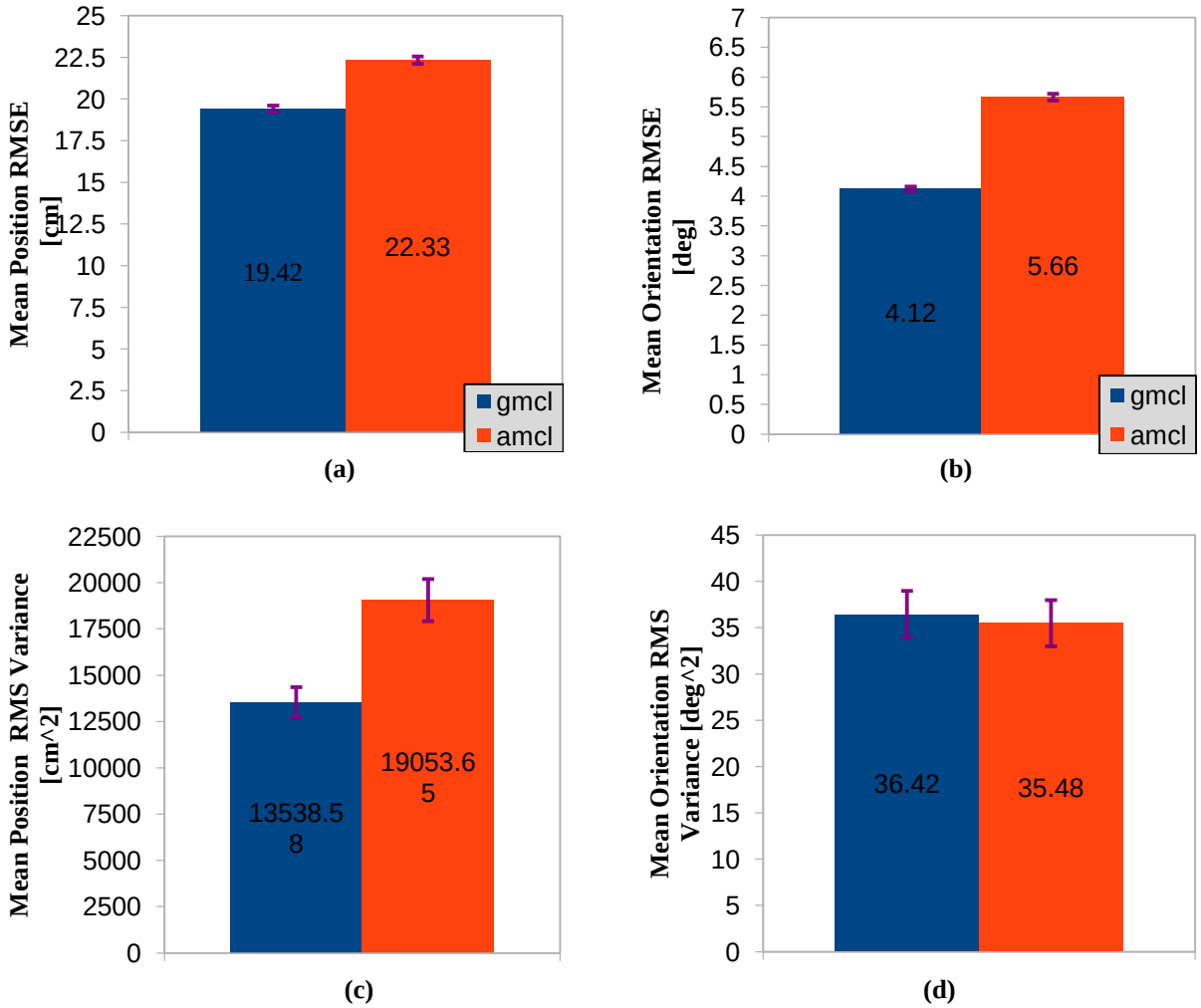


Figure (5) mean RMS of the 30 experiments for both techniques when studying pose tracking problem. Purple line represents margin of error

For experiment repetitions of 30 times, the margin of error in mean RMS of error and variance for both position and orientation at a confidence level of90%were:

- Margin of error in mean RMS for position error in percentage±1.03% .
- Margin of error in mean RMS for orientation error in percentage±1.07% .
- Margin of error in mean RMS for particle position variance in percentage±6.39% .

- Margin of error in mean RMS for particle orientation variance in percentage$\pm$7.08% .

Although the margin of error in mean RMS for particle position and orientation variance is relatively large, not much can be done about it due to the large randomness in RMS values, where a large number of experiment iterations is needed to reduce it.

After conducting the experiment 30 times, the performance of both *amcl* and *gmcl* during the entire study of pose tracking can be seen in Figure (5). The figure expresses the mean RMS values of error and variance for each of the position and orientation for *amcl* and *gmcl* taking into account the margin of error (colored in purple).

From Figure (5), we could infer the percentage of error and variance that *gmcl* improves compared to *amcl* when used in solving pose tracking problem. Values in Figure (6) represent relative percentage improvement of error and variance taking into account the margin of error (colored in purple). These values are calculated by substituting the mean RMS and variance of error for the position and orientation for both *gmcl* and *amcl* in the following equation:

$$\%RelativeDecrease(x_{amcl}, x_{gmcl}) = \frac{x_{amcl} - x_{gmcl}}{x_{amcl}} \times 100 \qquad (12)$$

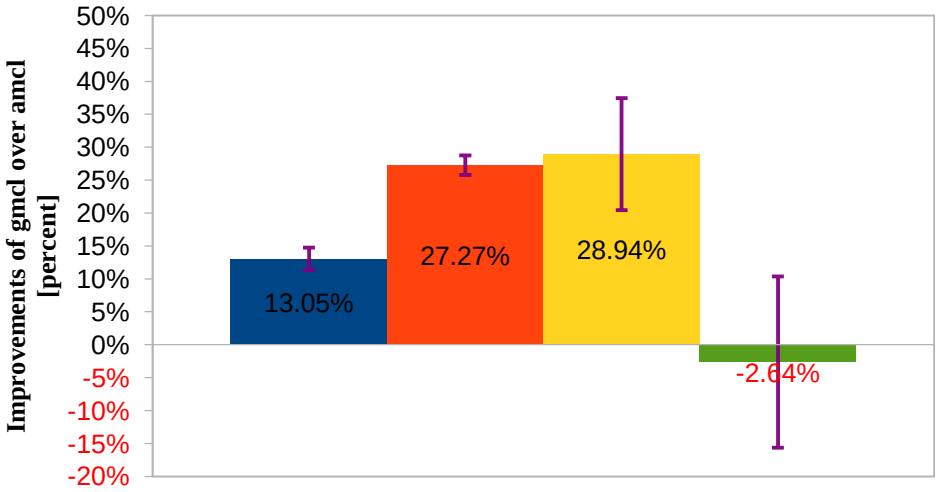


Figure (6) reduction of mean RMS that *gmcl* achieved relative to mean RMS of *amcl*

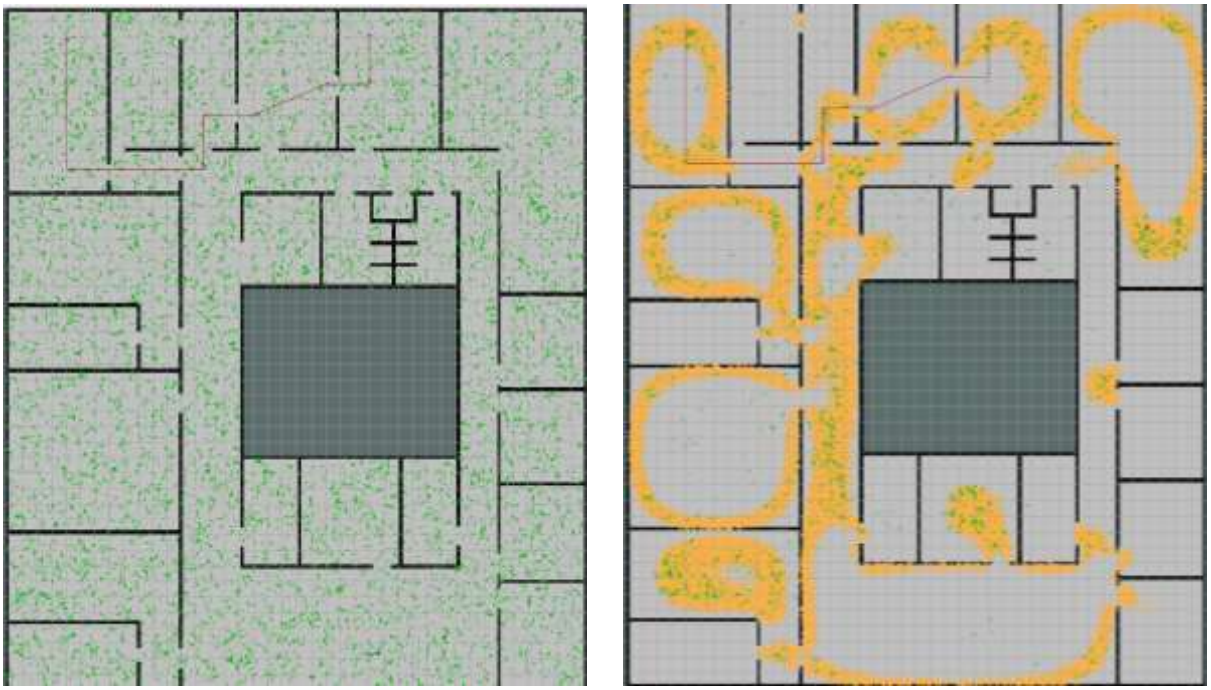
5.2 Global Localization:

In this experiment, the initial pose of the robot at start time is unknown, so particles in *amcl* are uniformly distributed over the free cells of the map. On the other hand, in *gmcl*, the particles are uniformly distributed on SER. The particle distribution of both *amcl* and *gmcl* can be seen in Figure (7).

It can be seen that some of *gmcl* particles in Figure (7-b) are outside SER. The reason is that *gmcl*, after spreading the particles at start time for global localization study, executes the algorithm once. During execution, Intelligent particle filter performs crossover and mutation on one third of small-weight particles trying to move them to the pose of large- weight particles, so these particles appear as a result.

Performance evaluation function for global localization study can be described by a two-value function (0 if robot's pose has not been discovered, 1 if robot's pose has been discovered) as follows:

$$f(x) = \begin{matrix} 1\ if\ x = success \\ 0\ if\ x = failure \end{matrix} \qquad (13)$$



(a) **(b)**

Figure (7) particles at start time when studying global localization, green pose array represents particles that compose the set of pose estimates (a) represents *amcl* particles, (b) represents *gmcl* particles, orange pose array represents SER.

We consider that *gmcl* and *amcl* have discovered the robot's pose, if they are able to (before the robot reaches the path point 3):

- Make the error value of the position and orientation less than $50cm$and $10deg$,respectively.
- Set particles in the converged state.

After iterating the experiment 30 times, it was found that the margin of error in the percentage of success rate in discovering robot's pose was$\pm$13.5with a confidence level of 90% .

The large margin of error is due to the nature of performance evaluation function, as two-value functions impose a large standard deviation, which makes SEM large. When the experiment is repeated 100 times, the margin of error in the percentage of success rate in discovering robot's pose is equal to $\pm$7.2. Eventually, to reduce the margin beyond this value too many iterations are required.

After conducting the experiment 100 times, the performance of both *gmcl* and *amcl* in the study of global localization is shown in Figure (8-a), expressed with the percentage success rate in discovering robot's pose taking into account the margin of error (colored in purple).

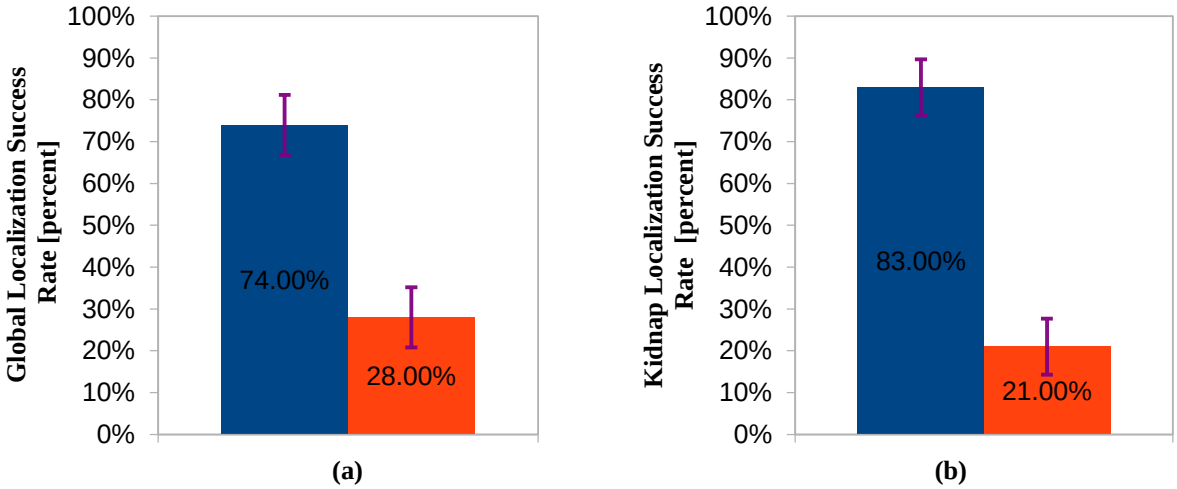


Figure (8) percentage success rate in discovering robot's pose for both *amcl* and *gmcl*. (a) for global localization study, (b) for kidnapped-robot problem study

5.3 Kidnapped-Robot Problem:

In this experiment, the robot is operated in similar manner of the pose tracking study. However, once the robot reaches path point 2 in simulation environment, it is teleported (Kidnapped) to path point 4 in the simulation environment. Here, *gmcl* and *amcl* have two tasks, the first is to detect that the robot has been kidnapped, and the second is to find the new pose of the robot.

Performance evaluation function for the study of kidnapped-robot problem is the same as in global localization (Equation (13)).

After being kidnapped, we consider that *amcl* and *gmcl* have discovered the new robot's pose if they -before the robot reaches the end of the path, which is point 8- are able to:

- Make the error value of the position and orientation less than $50cm$and $10deg$,respectively.
- Set particles in the converged state.

Just like in global localization study, we conduct the experiment 100 times, the performance of both *gmcl* and *amcl* in the study of kidnapped-robot problem can be seen in Figure (8-b), which expresses the percentage success rate in discovering robot's pose for both techniques taking into account the margin of error which is equal to$\pm$6.7when confidence level90%was adopted (colored in purple).

Results and Discussion:

After conducting the experiments with both *gmcl* and *amcl* to solve the three localization problems,we present here discussion for the results of these experiments. (A sample of each experiment for the three problems is posted on YouTube*)

1. Pose Tracking Results:

Taking advantage of Optimal and Intelligent particle filter, *gmcl* was able to reduce position error by13%and orientation error by 27%. We also observed a decrease in position variance by29% .

The drop in position variance is due to the fact that both *gmcl* and *amcl* may sense, as a result of changing in sensor reading during experiment, that a sudden change in robot's pose has been occurred, i.e. a kidnapping of the robot. Although in fact, the robot has not been

*** https://youtu.be/J9ZcCon6k-gv**

kidnapped. As a result of this sensing, global particles are added, which increases the variance. Here *gmcl* is superior to *amcl* as it spread particles on SER due to the use of Self-Adaptive particle filter rather than over the whole map, which makes the position variance less than that of *amcl*.

gmcl can not reduce orientation variance. It rather increases it by 2.6%. This is due to the fact that even when *gmcl* spread particles on SER, these particles have a uniform angular distribution similar to *amcl*, which cannot be improved currently in *gmcl*.

2. Global Localization Results:

It is known that in the case of global localization, a large number of particles is required to increase the chances of successful detection of robot's pose. As we mentioned earlier that *amcl* uses *ten* thousands particles in the experiments while *gmcl* uses only *two* thousands particles.

Nevertheless, the results shows the superiority of *gmcl* due to Self-Adaptive particle filter, which distributes the particles in time$t = 0$on SER. Intelligent and Optimal particle filters also have a role in increasing the success rate of robot's pose detection.

The increase in percentage success rate for both techniques is directly related to the maximum number of particles that they could possess.

3. Kidnapped-Robot Problem Results:

Both *gmcl* and *amcl* use the same modification of *standard-mcl* (*augmented-mcl)* to detect if a robot has been kidnapped. However, the approach of finding the new robot pose is done differently, in *gmcl* global particles are spread randomly on SER, while in *amcl* global particles are spread randomly over the free space of the map. Here it can be emphasized that *gmcl* was able to solve the kidnapped-robot problem with a greater percentage success than the global localization. Although, it is known that dealing with the kidnapped-robot problem is more difficult than global localization.

This is because in our global localization experiment, the robot's pose in time$t = 0$has symmetrical poses in the map with respect to sensor reading. This makes it necessary to form particle clusters in these poses. Also, since we use a small number of particles, a particle cluster may not be formed near robot's pose which leads to failure in global localization. While in the kidnapped robot problem the robot is teleported to a pose in map that does not have any symmetrical poses with respect to sensor reading, making it more detectable and thus a higher percentage success rate. However, in *amcl* the percentage success rate in kidnapped-robot problem study was still lower than the percentage success rate in global localization study.

Conclusions and Recommendations:

The experimental results conclude that:

Adding a selected number of filters to work with *amcl* for solving the localization problem produce good results with an acceptable computational complexity that enables it to work in real time.

Size of the map plays a role in determining the number of particles only when dealing with global localization and kidnapped-robot problem and does not play role in pose tracking.

In *gmcl*, attention should be paid to the number of auxiliary particles used in the Optimal particle filter. Increasing the number of these particles gives better performance in pose tracking, but it increases computation time linearly $O(T_L \cdot B \cdot M_{t-1})$.

Changing the values of the crossover alpha and mutation probability in *gmcl* does not affect the computational complexity. It plays an important role in changing the estimated pose by

gmcl. Since there is no fixed rule for choosing these values, it must be experimentally selected for the values that gives the best performance.

References:

[1] Quigley, M. ; Gerkey, B. ; Conley, K. ; Faust, J. ; Foote, T. ; Leibs, J. ; Berger, E. ; Wheeler, R. ; Andrew, N. *ROS: An open-source Robot Operating System*. In ICRA Workshop on Open Source Software, 2009, pp. 1-3.

[2] n.d. *List of robots using ROS*, 2021, http://robots.ros.org.

[3] Zhang, L. ; Merrifield, R. ; Deguet, A. ; Yang, G. *Powering the world's robots—10 years of ROS*. Science Robotics. 2(11), 2017.

[4] Koenig, N. ; Howard, A. *Design and use paradigms for Gazebo, an open-source multi-robot simulator*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Japan, 2004, pp. 2149–2154.

[5] Edwards, S. M. ; Lewis, C. L. *ROS-Industrial — Applying the Robot Operating System (ROS) to Industrial Applications*. International Conference on Robotics and Automation/Robot Operating System Developer Conference (ICRA/ROSCon), Minnesota, May 2012.

[6] Filliat, D ; Meyer, J.-A. *Map-based Navigation in Mobile Robots. A Review of Localization Strategies*. In Cognitive Systems Research, 4(4),2003, pp243-282.

[7] Del Moral, P. *Non Linear Filtering: Interacting Particle Solution.* Markov Processes and Related Fields. 2 (4), 1996, 555–580.

[8] Gerkey, B. *amcl – ros wiki*, 2021, http://wiki.ros.org/amcl

[9] Marder-Eppstein, E. *navigation –ros wiki*, 2021 http://wiki.ros.org/navigation

[10] Thrun, S. ; Burgard, W. ; Fox, D. *Probabilistic robotics.* Cambridge, MIT Press. September 2005.

[11] Fox, D. *Adapting the Sample Size in Particle Filters Through KLD-Sampling.* International Journal of Robotics Research, 22, 2003.

[12] Zaman, S., ; Slany, W. ; Steinbauer, G. *ROS-based Mapping, Localization and Autonomous Navigation using a Pioneer 3-DX Robot and their Relevant Issues*. Electronics, Communications and Photonics Conference, 2011.

[13] Pajaziti, A. ; Avdullahu, P. *SLAM – Map Building and Navigation via ROS.* International Journal of Intelligent Systems and Applications in Engineering, 2014.

[14] Takaya, K. ; Asai, T. ; Kroumov, V. ; Smarandache, F. *Simulation Environment for Mobile Robots Testing Using ROS and Gazebo*. 20th International Conference on System Theory, Control and Computing, 2016.

[15] Singh, D. ; Trivedi, E. ; Sharma, Y. ; Niranjan, V. *TurtleBot: Design and Hardware Component Selection*. International Conference on Computing, Power and Communication Technologies, 2018.

[16] Diddeniya, I. ; Wanniarachchi, W. K. ; Silva, P. & Ganegoda, N. *Efficient Office Assistant Robot System: Autonomous Navigation and Controlling Based on ROS*. International Journal of Multidisciplinary Studies, 6.(1), 2019.

[17] Stahl, T. ; Wischnewski, A. ; Betz, J. ; Lienkamp, M. *ROS-based localization of a race vehicle at high-speed using LIDAR*. In Proceeding of E3S Web of Conferences, 2019.

[18] Talwar, D. ; Jung, S. *Particle Filter-based Localization of a Mobile Robot by Using a Single LiDAR Sensor under SLAM in ROS Environment*. 19th International Conference on Control, Automation and Systems, 2019.

[19] Thale, S. ; Prabhu, M. ; Thakur, P. ; Kadam, P. *ROS based SLAM implementation*

for Autonomous navigation using Turtlebot. ITM Web of Conferences, 2020.
[20] Blanco, J. ; Gonzalez, J. ; Fernandez-Madrigal, J. *An optimal filtering algorithm for non-parametric observation models in robot localization*. IEEE International Confereence on Robotics and Automation (ICRA), 2008, pp. 461–466.
[21] Yin, S. ; Zhu, X. *Intelligent particle filter and its application on fault detection of nonlinear system.* IEEE Transactions on Industrial Electronics, 2015, pp.3852-3861.
[22] Zhang, L. ; Zapata, R. ; Lepinay, P. *Self-Adaptive Monte Carlo for Single-Robot and Multi-Robot Localization*. Proceedings of the IEEE International Conference on Automation and Logistics, 2009.
[23] Knuth, D. *The Art of Computer Programming, Volume 3: Sorting and Searching.*2^{nd}.ed., Addison–Wesley Professional, 1998, 800.
[24] n.d. *Turtlebot 3 Documentation*. 2021, https://github.com/ROBOTIS-GIT/emanual/tree/
master/docs/en/platform/turtlebot3
[25] Taylor, J. R. *An Introduction to Error Analysis*.2^{nd}. ed., University Science Books, 1996, 327.