IMPROVED GLOBAL LOCALIZATION AND RESAMPLING TECHNIQUES FOR MONTE CARLO LOCALIZATION ALGORITHM

A THESIS SUBMITTED TO
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
KOCAELİ UNIVERSITY

BY

HUMAM ABUALKEBASH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHATRONICS ENGINEERING

KOCAELİ 2020

IMPROVED GLOBAL LOCALIZATION AND RESAMPLING TECHNIQUES FOR MONTE CARLO LOCALIZATION ALGORITHM

A THESIS SUBMITTED TO
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
KOCAELİ UNIVERSITY

BY

HUMAM ABUALKEBASH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHATRONICS ENGINEERING

Prof.Dr. Hasan OCAK		
Supervisor,	**Kocaeli University**	**..........................**
Prof.Dr. Hüseyin Metin ERTUNÇ		
Jury member,	**Kocaeli University**	**..........................**
Prof.Dr. Şeref Naci ENGİN		
Jury member,	**Yıldız Teknik University**	**..........................**

Thesis Defense Date: 17.08.2020

ACKNOWLEDGMENT

This thesis presents the global indoor localization problem for mobile robots. The created uncertainty is addressed using the Monte Carlo Localization (MCL) algorithm that uses the Particle Filter to estimate the robot's pose. This thesis report presents two novel approaches based on the MCL algorithm that can drastically reduce the sample set size and the amount of time it takes for the mobile robot to successfully localize itself in its environment.

First and foremost, I thank Almighty Allah for giving me the patience and endurance to successfully complete this work. With great appreciation, I thank my thesis advisor Professor Dr. Hasan Ocak for his continuous guidance and support that prompted me to overcome difficulties and encouraged me throughout my research work. I also would like to thank the lawyer Meryem Erol for her help in translating the thesis abstract to Turkish language.

I am very thankful for my parents and valuable brothers and sisters for their support, endless love, consistent encouragement, and prayers during my academic career.

Finally, thanks to all my friends for their support and motivation.

August – 2020 Humam ABUALKEBASH

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

α	: Motion noise parameters (robot-specific error parameters)
α_1	: Rotational error due to rotational motion
α_2	: Rotational error due to translational motion
α_3	: Translational error due to translational motion
α_4	: Translational error due to rotational motion
α_{fast}	: Exponential decay rate for the fast averages weights filter
α_{slow}	: Exponential decay rate for the slow averages weights filter
$bel(x)$	: Belief of robot's pose
$\overline{bel}(x)$	: Predicted belief of robot's pose
c	: Cumulative samples weight
D	: Diameter of the robot wheel, (m)
d	: Distance transform, (m)
δ_{rot1}	: Initial rotation in the odometry information, (radians)
δ_{rot2}	: Final rotation in the odometry information, (radians)
δ_{trans}	: translation in the odometry information, (m)
Enc_l	: Left encoder value, (degrees)
Enc_r	: Right encoder value, (degrees)
L	: Distance between robot wheels, (m)
LH	: Likelihood
m	: Occupancy grid map, (grid cells)
μ	: Mean value
N	: Number of samples in the sample set
$p(x_t \mid x_{t-1}, u_t)$	: Motion model (state transition probability)
$p(z_t \mid x_t, m)$	: Sensor model (measurement probability)
P_k	: Decision parameter, (pixel)
${}^{w}P_s$	: Position vector locates the origin of the s frame relative to the w frame, (m)
r	: Uniform random number in the interval [0,1]
R	: Radius of the desired area in the resampling step, (m)
$R_z(\theta)$	: Rotation matrix about the z-axis
σ	: Standard deviation
σ_{hit}	: Measurement noise, (m)
$t_{elapsed}$	: Elapsed time since the robot was confident about its pose, (sec.)
$t_{lastConv}$	: Real-time at the last filter convergence position, (sec.)
t_{real}	: Real-time, (sec.)
θ	: Yaw angle, (radians)
θ_{ep}	: Orientation Angle of the laser ray relative to the sensor direction, (radians)
θ_{est}	: Estimated heading direction of the robot relative to the x-axis of the world frame, (radians)
θ_s	: Orientation Angle of the sensor frame relative to the robot frame, (radians)

u_t	: Control data, ([m m radians])
v_l	: Rotational speed of the right and left wheels, (radians/sec)
V_l	: Velocity of the left wheel, (m/sec)
v_{max}	: Maximum velocity of the mobile robot, (m/sec)
v_r	: Rotational speed of the right wheel, (radians/sec)
V_r	: Velocity of the right wheel, (m/sec)
v_t	: Linear velocity of the mobile robot, (m/sec)
w_{avg}	: Samples average weight
w_{fast}	: Exponential filter of the importance weight over a short time
w_{hit}	: Probability of the expected measurement
w_{rand}	: Probability of the random measurement
w_{slow}	: Exponential filter of the importance weight over a long time
ω_t	: Angular velocity of the mobile robot, (radians/sec)
$w_t^{[n]}$	: Weight of the predicted sample n in discrete time steps
x	: Position coordinate in the x-axis direction, (m)
x_{est}	: Estimated x-coordinate of the actual robot pose, (m)
x_k	: x-axis of the first pixel in the grid cell, (pixel)
x_{sens}	: Position coordinate of the sensor in the x-axis of the robot frame, (m)
x_t	: Robot's pose vector in a 2D plane in discrete time steps, ([m m radians])
X_t	: Corrected sample set in discrete time steps, ([m m radians])
$\bar{X}_t$	: Predicted sample set in discrete time steps, ([m m radians])
y	: Position coordinate in the y-axis direction, (m)
y_{est}	: Estimated y-coordinate of the actual robot pose, (m)
y_k	: y-axis of the first pixel in the grid cell, (pixel)
y_{sens}	: Position coordinate of the sensor in the y-axis of the robot frame, (m)
z_{max}	: Maximum sensor range, (m)
z_t	: Sensor data in discrete time steps, (m)
z_t^b	: Individual beam range, (m)

Abbreviations

AGV	: Autonomous Guided Vehicle
AMCL	: Adaptive Monte Carlo Localization
ANT	: Autonomous Navigation Technology
AUV	: Autonomous Underwater Vehicle
CCD	: Charge-Coupled Device
CV	: Computer Vision
DBN	: Dynamic Bayes Network
DDA	: Digital Difference Analyzer
DGPS5	: Differential Global Positioning System
DOF	: Degrees of Freedom
EKF	: Extended Kalman Filter
GPS	: Global Positioning System
HMM	: Hidden Markov Model

HSM	: Hough Scan Matching
IC	: Integrated Circuits
IMU	: Inertial Measurement Unit
LiDAR	: Light Detection and Ranging
MCL	: Monte Carlo Localization
MHT	: Multi-Hypothesis Tracking
OGM	: Occupancy Grid Map
PDF	: Probability Density Function
PF	: Particle Filter
PMD	: Point-Mass Distribution

MONTE CARLO LOKALİZASYON ALGORİTMASI İÇİN GELİŞTİRİLMİŞ GLOBAL LOKALİZASYON VE YENİDEN ÖRNEKLEME TEKNİKLERİ

ÖZET

Global kapalı alan konumlandırma algoritmaları, robot ilk konumunu ve yönünü bilmediği durumlarda sensör ölçümlerini kullanarak robotun daha önceden haritalandırdığı ortamlardaki konumunu ve yönünü tahmin etmesini sağlar. Standart uyarlanır Monte Carlo Lokalizasyon (AMCL), global belirsizliklerle başarılı bir şekilde başa çıkabilen yüksek verimli bir konumlandırma algoritmasıdır. Global konumlandırma problemi, gezgin robotlar için çok önemli olduğundan, algoritmanın doğru konuma yakınsaması için geçen süreyi dikkate değer ölçüde azaltan yeni bir yaklaşım sunuyoruz. Hazırlanılan algoritma; verilen harita ve ilk tarama verilerini göz önüne alarak sensör modeline göre yüksek olasılıklı bölgeleri tespit eder. Sonuç olarak, önerilen örneklem dağılımı konumlandırma sürecini hızlandıracaktır. Biz bu çalışmada ayrıca sensörün görüş alanındaki haritalandırılmamış hareketli engeller sebebiyle örneklem ağırlıkları düştüğünde, robotun hızlı bir şekilde gerçek konumunu kestirmesini sağlayan ve kaçırılan robot problemleriyle başa çıkan etkili bir yeniden örnekleme stratejisi sunuyoruz. Hazırlanan teknik; en son başarılı konum bilgilerini kullanarak rastgele örneklemi robotun konumunun etrafında merkezlenen dairesel bir alana dağıtır. Örneklemler yüksek olasılıklı bölgelere dağıtıldığından, örneklemin gerçek konum ve yöne ulaşması daha az zaman almaktadır. Çalışma kapsamında elde edilen sonuçlar, küçük örneklem gruplarında bile, önerilen iyileştirmelerin konumlandırmadaki etkinliğini göstermektedir. Sonuç olarak, önerilen metotlar algoritmanın gerçek zamanlı performansını önemli ölçüde arttırmakta ve hesaplama maliyetini düşürmektedir.

Anahtar Kelimeler: AMCL, Global Konumlandırma, Olasılık, Örneklem Dağılımı, Yeniden Örnekleme.

IMPROVED GLOBAL LOCALIZATION AND RESAMPLING TECHNIQUES FOR MONTE CARLO LOCALIZATION ALGORITHM

ABSTRACT

Global indoor localization algorithms enable the robot to estimate its pose in pre-mapped environments using sensor measurements when its initial pose is unknown. The conventional Adaptive Monte Carlo Localization (AMCL) is a highly efficient localization algorithm that can successfully cope with global uncertainty. Since the global localization problem is paramount in mobile robots, we propose a novel approach that can significantly reduce the amount of time it takes for the algorithm to converge to true pose. Given the map and initial scan data, the proposed algorithm detects regions with high likelihood based on the observation model. As a result, the suggested sample distribution will expedite the process of localization. In this study, we also present an effective resampling strategy to deal with the kidnapped robot problem that enables the robot to recover quickly when the sample weights drop-down due to unmapped dynamic obstacles within the sensor's field of view. The proposed approach distributes the random samples within a circular region centered around the robot's position by taking into account the prior knowledge about the most recent successful pose estimation. Since the samples are distributed over the region with high probabilities, it will take less time for the samples to converge to the actual pose. The results demonstrate the high efficiency of the proposed scheme, even with small sample sets. Consequently, the proposed scheme significantly increases the real-time performance of the algorithm in terms of decreasing the computational cost.

Keywords: AMCL, Global Localization, Likelihood, Sample Distribution, Resampling.

INTRODUCTION

Robotics has been a huge hit so far in the industrial manufacturing world. Manipulators, or robot arms, which are installed on a fixed board to a specified location in the production lines, can move with high precision and speed to carry out repetitious functions such as drilling, assembly line for vehicles, spot welding, and painting [1]. In electronics manufacturing, robot arms install the Integrated Circuits (IC) with super meticulously, making laptops and mobile phones possible. Unfortunately, despite all their successes, these manipulators come up with a common drawback: lack of mobility, since the fixed robot arms bolted to a specified position, they have a restricted workspace. Contrariwise, mobile robots have the ability to roam around the plant, allowing their skills to be applied anywhere in the state space.

Tele-operated systems have been evolved in order to give a human operator the ability to execute complicated missions in inhospitable and hostile environments [2]. Mobile robots can be regarded as an example of these systems, as they can be remotely operated to carry out specific tasks. Tele-operated mobile robots are extensively utilized to perform challenging duties in dangerous environments; well-known examples are cleaning nuclear plants [3], demining operations [4], and underwater structures [5]. Plustech developed the first paradigm of walking robot [6], the leg coordination of Plustech’s walking robot is automated while the localization and navigation are still carried out by the human driver. Another example of a tele-operated mobile robot is the Autonomous Underwater Vehicle (AUV) [7]; it controls three propellers to maintain the stability of the underwater robot autonomously against water currents and turbulence, at the same time, the human operator manually directs the submarine to the targets. Based on the control scheme of these types of mobile robots which provide motion control, the human operator carries out the activities of localization and perception. So, the tele-operated mobile vehicles are compelling not on mobility reasons but on account of their autonomy. Besides, remote operations have proved to be tiring activities that need specialized training for the operator [2].

Therefore, the ability of mobile robots to perceive its environments and to navigate without any human interference is essential.

The first challenge for the technology of mobility is locomotion itself, how the mobile robot should move. Locomotion stands for movement capability, and in order to solve locomotion problems, mobile robotics should be able to understand kinematics and mechanisms, dynamics and control theory [8]. Mobile vehicles require mechanisms of locomotion that endow them unrestricted movement all over their environment. However, there is a vast diversity of ways for a mobile robot to move, such as swim, roll, skate, jump, fly, slide, run, and of course, walk. Most of those mechanisms of locomotion were inspired by their biological counterparts. So, the selection of a robot locomotion strategy is an essential part of mobile robot design. Since the natural environments are unstructured and rough terrain, it is understandable that the legged locomotion is preferred. Anyway, both of the outdoors and indoors human environment consists mostly of engineered smooth surfaces. Thus, it is also understandable that almost all mobile robotics industrial applications employ a form of wheeled locomotion.

Autonomous mobile robotics is more than only possess the ability to move. However, the autonomy of mobile robots indicates their ability to perceive the environment, to localize itself, and to navigate robustly from one place to another. As a result, the design of autonomous mobile robots comprises the integration of several different knowledge bodies, and this makes mobile robotics a multidisciplinary field. The mobile roboticist exploits the areas of signal processing and the specialized information structures to construct reliable perceptual systems, such as Computer Vision (CV), to utilize a multitude of sensing techniques properly. On the other side, localization and navigation require information theory, artificial intelligence, computer algorithms knowledge, and of course, probability theory.

From the above, we can summarize the control scheme of the autonomous mobile robot systems [6] as follows:

- Perception. This phase includes sensing, information extraction, and interpretation.
- Localization. The mobile robot will utilize the perceptual data to localize itself in the environment.

- Path planning. Given the robot’s current position and heading direction, the mobile robot generates the desired path to the target.
- Motion control. Based on the generated path, motion control will command the robot’s actuators.

Localization is the first point at which sensing and mobility must meet. As a result of the mobile robot control scheme, the mobile robot will be able to navigate autonomously throughout the environment.

Perception is a significant task for autonomous systems, where it enables mobile robots to get information about its environment. This process is done by acquiring measurements from different types of sensors and then extracting useful information from these measures. Therefore, perception is more than sense; it is also performing meaningful interpretations of perceptual data.

There is a broad set of sensors utilized in mobile robots. Some sensors are employed to detect simple values such as motors rotational speed or the temperature of the electronics parts in the robot; these sensors are called proprioceptive sensors. On the other side, exteroceptive sensors are being used to gather information about the environment where the robot navigates, or even to find the global location of a robot directly, for example, sound amplitude, light intensity, and distance measurements. Since autonomous mobile robots are concerned with interacting with the surrounding environment, we should take care of sensors utilized to extract meaningful information from the robot’s environment, where the robot frequently might face unexpected environmental features. Therefore, such sensing is crucial. Some sensors for mobile robots are listed below [9]:

- Motor/wheel sensors
- Heading sensors
- Accelerometers
- Inertial Measurement Unit (IMU)
- Active ranging
- Speed/motion sensors
- Vision sensors

Armed with mechanisms of locomotion and equipped with software and hardware for perception, mobile robots can perceive their environment and move. However, the mobile robot cannot navigate autonomously from one place to another without having accurate knowledge about its current position and heading direction in the world. As a result, we must take a step toward a high-level challenge by solving the localization problem for mobile robots [10], and of course, it is paramount to get autonomy.

In mobile robotics, the word pose stands for position and orientation. However, mobile robot localization or pose estimation is the problem of estimating a robot's position and orientation in real-time relative to an external reference frame given a sensor data and map of the environment [11]. In the past decade, the pose estimation problem has received the most research attention. And as a result, considerable progress has been created on this aspect.

In the case of an automobile, the Global Positioning System (GPS) sensor can be used to infer its pose, and in general, in the outdoor environments attaching such an accurate GPS sensor to a mobile vehicle, much of the positioning issue would be avoided. The exact location will be available for the vehicle via GPS; therefore, the answer to the question, "Where am I?" would always be available immediately. Unfortunately, the technology of GPS cannot be employed indoors or in unstructured areas, because the accuracy of existing GPS is provided only within a few feet. So, it is unreasonable to use the current GPS network for localizing miniature mobile robots like body-navigating nanorobots of the future and desk robots as well as human-scale robots. Nowadays, a high accuracy framework like Differential GPS (DGPS5) is utilized [12]; however, this system is too costly for the general objectives as well as it is also useless indoors.

In fact, the pose estimation problem is sensor noise compensation [13], where the mobile robot has to estimate its pose from noisy and not directly observable information. Therefore, localization faces difficult challenges due to the incompleteness and inaccuracy of the sensors. Since the primary input for the perception process is the sensor measurements, the degree at which sensors can recognize the environment is crucial. Sensor noise limits the steadiness of measures in the same world state, and thus on the number of useful bits obtainable from every reading of the sensor. Often, sensor noise occurs because some ecological objects are not captured by the representation of the robot, and are therefore ignored. For instance, some mobile robots use a vision system such as a Charge-Coupled Device (CCD) video camera for indoor navigation; the robot navigates in its workplace by detecting color values using its CCD camera. The issue is when the clouds hide the sun, the lighting of the interior building changes due to the windows around the building. As a result, the values of the hue will not be constant. From the robot’s viewpoint, the CCD color seems noisy as if it is subject to random error. Therefore, the values of the hue acquired by the CCD video camera would be worthless unless the mobile robot can notice the location of clouds and the sun in its model.

In vision-based systems, illumination dependency is just one form of noticeable noise. Blurring, picture jitter, blooming, and signal gain are also considered as further noise sources, virtually these types of sensor noises reduce the helpful content of the color image.

In the case of sonar sensors (ultrasonic range finder sensors), as a sonar transducer transmit the sound wave toward an angled and reasonably smooth surface, much of the signal will reflect far away coherently, resulting in failure to produce an echo. Based on the properties of the material, some of the energy may return. When the level of recovered energy is close to the sonar sensor gain threshold, the sonar sometimes succeeds in perceiving the object, and at other times it fails.

In the motor/wheel sensors (Odometry) and heading sensors, the pose is updated based on proprioceptive sensors. The motion of the mobile robot, perceived with orientation sensors or wheel encoders or both, is integrated to determine the pose. Because the errors of sensor measurement are incorporated, over time, the pose error will accumulate. Therefore, the mobile robot’s pose should be continuously updated with other robust localization methodology. Otherwise, the mobile robot will not be able to estimate its pose in the long run accurately. The errors of sensor measurement might be deterministic, which means they can be removed by the calibration process. However, some other errors are random (non-deterministic) errors, and over time such errors increase uncertainties in pose estimation. From a geometric perspective, sensor errors are classified into three types [6]:

- Range error, incorporated distance of the robot movement (sum of the wheel movement).
- Turn error, related to the turns of the wheel (wheel motion difference)
- Drift error, the difference in the wheels error causes an error in the angular orientation of the robot.

The most straightforward localization problem is local position tracking [14], where the initial robot's pose is known. As a result, the uncertainty is local, and the problem is only to compensate for cumulative, small errors in the Odometry of the robot. More defying is the global localization problem [15]; this problem is raised when the initial robot's pose is unknown, but rather it must infer it from scratch. Here, the created uncertainty will be global, which leads to a more complicated problem. The error in the global pose estimation problem cannot be postulated to be small. In contrast, when a well-localized autonomous mobile robot is carried to an arbitrary location in operation mode without being told, the problem is extended to the kidnapped robot problem [16,17]. This problem is crucially more difficult than global localization where the robot might believe it knows where it is, while it does not. The kidnapped robot problem is mostly employed to test the ability of a mobile robot to recover from disastrous positioning failures.

In the robotic literature, there is a wide range of probabilistic approaches that used to solve indoor localization problem, including Grid-based algorithm [18,19], Extended Kalman Filter (EKF) [20,21], Monte Carlo Localization (MCL) [10,22], and some hybrid schemes [23,24]. The probabilistic pose estimation methodologies are part of probabilistic robotics. Based on mathematical statistics, probabilistic robotics provides autonomous mobile robots with a high level of robustness [25]. Undoubtedly, MCL is considered one of the subset approaches that can successfully deal with the created global uncertainty. And as a probabilistic approach, MCL can compute the instantaneous uncertainty of a mobile robot, and it is convenient to local and global pose estimation problem. Moreover, MCL is easy to implement and can solve the kidnapped and global localization problems in a very high robust and competent way. While other existing schemes cannot survive when the mobile robot is kidnapped or when localization failure occurs. Unlike the EKF which guarantees accuracy only for a linear system to which Gaussian noise is applied or for systems not dramatically nonlinear, MCL is capable of representing arbitrary distributions. Besides, MCL is more precise than algorithms based on the grid approach with fixed cell size, where the amount of memory needed is considerable.

The high efficiency of the MCL algorithm comes from the fact that it represents the uncertainty (robot’s pose) by a collection of particles, which are randomly generated over the robot poses in accordance to the posterior distribution. So, MCL uses a Particle Filter (PF) to cope with multi-modal distributions.

Unfortunately, conventional MCL still suffers from some shortcomings. The required number of particles is significant, where increasing the size of the particle set, the accuracy of MCL will increase. However, employing a higher number of particles will increase the complexity of MCL, and of course, the computational burden will increase. Therefore minimizing the number of needed particles is one of the main defies to the MCL algorithm [26]. Moreover, the MCL represents the global pose uncertainty by a weighted particle set distributed over the whole state space rather than focusing on the high-probability poses, and this is one reason why MCL needs more particles. Another challenge that faces this family of localization approaches is the resampling process. When the probabilities drop-down due to localization failure or unnatural sensor noise, augmented MCL tries to add random samples over the entire map to overcome these problems [25]. However, as MCL is a stochastic algorithm, drawing random particles over the space might discard all poses near the true robot’s pose. Definitely, this negatively affects the real-time performance.

In this thesis, we propose two novel methodologies to reduce uncertainty in the global indoor localization problem; both of these methodologies are an extension to the conventional MCL. The first improvement introduces an optimized scheme at the initialization step that detects mapped regions with high probabilities only based on the initial scan data. Given the robot’s environment map and only the initial scan data, the proposed algorithm detects regions with high likelihood based on the observation model to distribute particles there. As a result, the suggested particle distribution will expedite the process of localization. And it will significantly reduce the amount of time it takes for the samples to converge to true pose during the global localization problem. The second improved scheme presents an effective resampling strategy to deal with the kidnapped robot problem that enables the robot to recover quickly when the sample (particle) weights drop-down due to unmapped dynamic obstacles within the sensor's field of view. While the classical resampling method distributes random particles around the entire working space, the proposed scheme distributes the random samples within a circular region centered around the robot's pose by taking into account the prior knowledge about the most recent successful pose estimation. Since the particles are distributed over the region with high probabilities, it will take less time for the mobile robot to estimate its accurate pose.

1. BACKGROUND KNOWLEDGE

This chapter introduces the fundamental knowledge of the localization problem. First, we will discuss the uncertainty in robotic systems. Then, we will present some concepts in probabilistic robotics, includes the probabilistic generative laws and the recursive Bayesian filters. After that, the Markov localization approach will be discussed as the basic algorithm for the localization techniques. Finally, the most popular localization algorithms in the robotic literature, grid-based localization, and Monte Carlo localization as a variant of particle filter will be explained.

1.1. Uncertainty

The exploitation of computer-controlled devices for real-world manipulating and perceiving is called robotics [25]. The systems of robotics are employed in our real-world to act through physical forces, and to perceive the environmental features through sensors. Opening new frontiers for mobile robots and entering the autonomous world makes it imperative for mobile robots to adapt to the immense existence of uncertainty. There are a variety of factors that causes uncertainty in mobile robots. First, the mobile robot environment is generally unpredictable. The uncertainty in the organized world, such as production lines is limited. However, environments such as homes, offices, and thoroughfares are a dynamic world that results in high uncertainty. The second factor is the sensors themselves; they have their restrictions. The resolution and range of sensors depend on the noises and their physical constraints. Third, the actuation system of mobile robots includes mechanical motors that are unpredictable, at least to some degree. Uncertainty originates from impacts such as mechanical failure and control noise. Forth, uncertainty is also created by the software of a mobile robot as all the world’s internal models are approximate. Finally, more uncertainties are generated by approximations in algorithms. Sometimes, accuracy has to be sacrificed in a real-time system to get a timely response.

Dealing with created uncertainties is the primary concern for researchers and arguably the most crucial step towards robust mobile robot systems.

1.2. **Probabilistic Robotics**

Probabilistic Robotics addresses the uncertainty problem in robot systems by employing probability theory. The key idea is through utilizing probability distributions; the probabilistic robotics can represent information overall potential guesses rather than just a best single guess [25]. Probabilistic approaches are less dependent on the robot's model accuracy compared to classical methods, so it helps the programmer to be freer when building the model, instead of trying to develop the most accurate model. Besides, probabilistic robotics are less dependent on the accuracy of sensors. Based on mathematical statistics, in real environments, the probabilistic robotics gives a new level of robustness to autonomous mobile robots; and this is evident when mobile robots try to solve localization, mapping, planning, control, and even Simultaneous Localization and Mapping (SLAM) [27].

1.2.1. **State**

The mobile robot's world is characterized by state. And it is referred to as all characteristics of the mobile robot and its world which may affect the future. A state can be classified into two major types: static state represents the characteristics that remain constant over time, such as wall location. In contrast, the dynamic state represents all features that tend to modify their configurations or locations over time such as people, movable furniture, doors, and extend to the mobile robot's features itself like velocity, pose, the status of sensors, and so on.

Rigid mobile robot has six dynamic state variables relative to the world frame. Three of them are the cartesian coordinates used to specify the robot's position in space, while the others used to describe the robot's orientation (pitch, roll, and yaw). If we take the case of a rigid mobile robot that is restricted to the planer world, the coordinates will be just three, two of them is the location coordinates in XY-plane and the heading direction (yaw). These three coordinates are called the pose of the robot.

Moreover, many other states may affect the operation of the mobile robot. For instance, the battery charge level, the status of the sensor, whether broken or not. However, in robotics, we can summarize the state variables as follows [25]:

- Robot’s pose, it involves the 2D position coordinates and heading direction of the mobile robot relative to the external reference global frame.
- Kinematic state, in robot manipulation, variables of the robot’s actuators configuration are also included in the robot’s pose. For example, joint angles might be added to the state.
- Dynamic state, the velocity of the mobile robot, and its joints velocities.
- Characteristics and location of the environment objects are also state variables. The object can be a desk, wall, or tree. And features may be texture or color. In some cases, objects and features are treated as landmarks.
- Positions and velocities of moving individuals and objects are also considered as state variables.

In our thesis, the robot’s pose at time t was denoted as x_t. However, the time was held in discrete steps $t = 0,1,2$ where the zero time refers to the first operation point in the beginning. Equation (1.1) represents the mobile robot’s state, which is restricted to the planer world.

$$x_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{1.1}$$

1.2.2. Environment interaction

The mobile robot can interact with its environment in two fundamental forms, as shown in the following Figure 1.1 [28]. Through its actuators, the mobile robot can affect the environment. On the other hand, with the help of sensors, the mobile robot can collect information about the environment’s state.

Perceptional interaction may include a range scan or camera image, and the results are called measurements. However, the second form of interaction may involve object manipulation or even robot motion. In real life, control actions are executed continuously and simultaneously with the measurements. Besides, control actions and sensor measurements are different data streams, and the robot records this data concurrently.

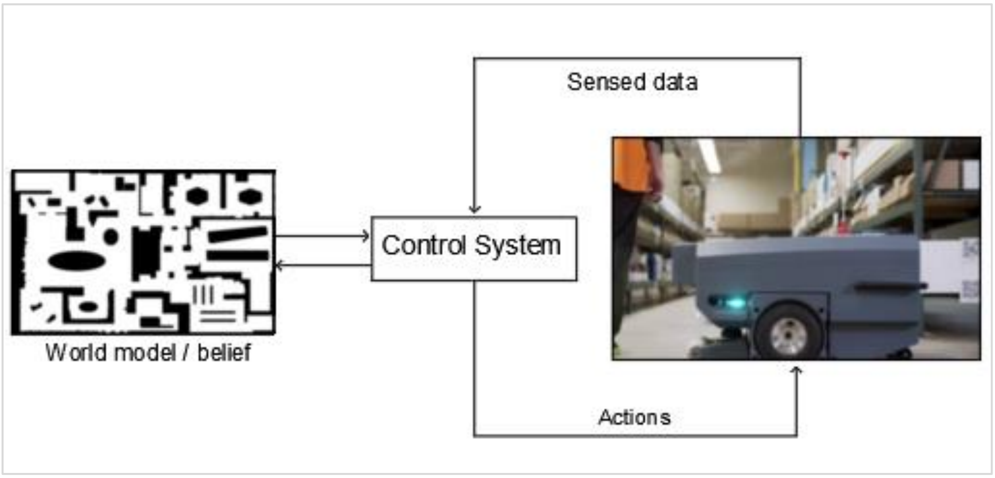


Figure 1.1. Mobile robot interactions with its environment

Measurement data provides a mobile robot with momentary information concerning the environment. We supposed that the mobile robot obtains one sensor data at a time. The following notation below stands for all perceptual data gathered between time t_1 and t_2.

$$z_{t1:t2} = z_{t1} + z_{t1+1} + z_{t1+2} + ... + z_{t2}, \quad \forall\ (t_1 \le t_2) \tag{1.2}$$

In the context of the localization problem of a mobile robot, control data refer to motion data or movement data, and it carries information related to the state change in the world. The velocity is considered as a typical example of control data. Setting the velocity of a mobile robot to 0.5 meters per second for ten seconds indicates that the pose of the robot will be around five meters ahead relative to the previous pose. Another example of control data is odometers. A sensor that measures the mobile robot’s wheels revolution is called odometer. Odometers provide information concerning state change. As a result, odometry data treated as control data, as they give information about the effect of control action. As we assumed in the case of measurement data, we also supposed that there is one odometry information at a time. When the mobile robot is in a stopped case, the legal odometry data will be “do nothing.”. Exactly as perceptual data, the sequence of control data from time t_1 to t_2 is denoted as below.

$$u_{t1:t2} = u_{t1} + u_{t1+1} + u_{t1+2} + ... + u_{t2}, \quad \forall\ (t_1 \le t_2) \tag{1.3}$$

Both control data and measurement data have essential roles in probabilistic robotics. The perceptual data increase the knowledge of the robot by providing information concerning the environment’s state. On the other side, as the environment of a mobile robot is stochastic, and due to noise inherent in the robot actuation, the control data tends to create a loss of knowledge. Once again, we emphasized that the control data and perceptual data are co-occurring. In probabilistic robotics, there are two separate models used to address these data. One is the motion model, while the other is the measurement model. Both of these two models will be discussed in the next chapter in detail.

1.2.3. **Probabilistic generative laws**

Measurements and state can be evolved in accordance with the probabilistic laws. The mobile robot state $\mathrm{x_t}$ is generated based on all previous states, controls, and measurements. Hence, the state evolution could be presented by a probability distribution as follows: $p(x_t \mid x_{0:t-1},\ z_{1:t-1},\ u_{1:t})$. We supposed first of all the mobile robot performs a control action u_t and then acquire sensor measurement z_t. The completed state x covers all previous events. Which means that the state x_{t-1} comprise all previous measurement data $z_{1:t-1}$, and control data $u_{1:t-1}$. So, the expression of the probability distribution that is in our hands can be expressed as follows.

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t) \tag{1.4}$$

This equality contains conditional independence. If the value of the conditioning variables such as x_{t-1} and u_t are known, then the variable such x_t is independent of other variables like $u_{1:t-1}$ and $z_{1:t-1}$. The probability density shown in Equation (1.4) is called motion model or action.

On the other hand, based on the generated state x_t we can model the procedure which generates the measurements. As we have a completed state x_t, we can get another crucial conditional independence that is called the perceptual model or the sensor model:

$$p(z_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t) \tag{1.5}$$

Once again, previous equality states that when the completed state x_t is present, no need for knowing the value of any other variables, such as previous states $x_{0:t-1}$, control data $u_{1:t}$, or even previous measurements $z_{1:t-1}$. In another way, the variable x_t is adequate to estimate the noisy measurement data z_t.

The probability distribution $p(x_t \mid x_{t-1},\ u_t)$ is termed as state transition probability, and it shows how the probability of robot control $\mathrm{u_t}$ to prompt a transition from the prior state x_{t-1} to the current state x_t the state $\mathrm{x_t}$ is generated based on the previous state x_{t-1} and the robot controls u_t. It is worth noting that the state transition probability is not a deterministic function; instead, it is a probability distribution. The probability distribution $p(z_t \mid x_t)$ is termed as measurement probability, and it specifies how the measurement z_t are produced from the environment state x_t. The measurement probability and state transition probability characterize the robot's dynamical stochastic system and its world. Figure 1.2 describes the development of states and measurements. The current state depends randomly on the prior state and recent control. And the most recent measurement data is randomly dependent on the current environment state. The following structure is also known as Dynamic Bayes Network (DBN) or Hidden Markov Model (HMM) [29,30].

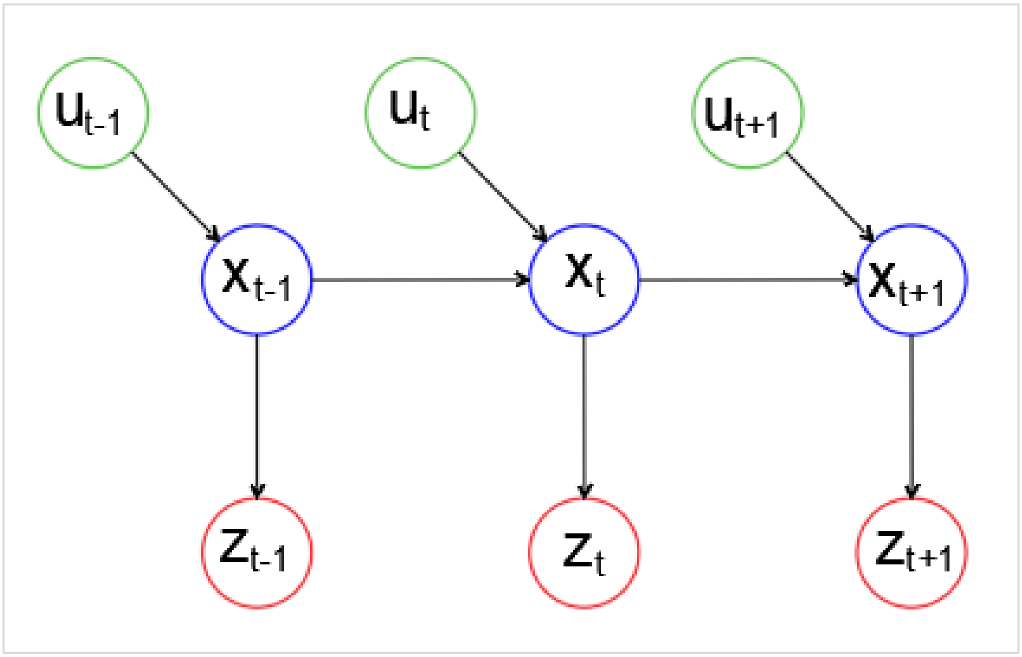


Figure 1.2. The hidden Markov model that describes the development states, and measurements

1.2.4. **Belief**

The internal knowledge of a mobile robot regarding the environment state is called belief. However, the environment state is not directly measurable, so the robot should estimate its belief from the collected data. The representation of belief is done using conditional probability distributions [25]. As a result of this representation, the distribution of belief allocates a density value (probability) to each possible hypothesis state regarding the actual state. The following symbolization below referred to the belief value at time t.

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}) \tag{1.6}$$

The previous equation of belief represents the posterior probability distribution over the environment state, conditioned on all previous perceptual data and all prior robot's control data. Equation (1.6) presumes that the belief is calculated after integrating the measurement data. On occasion, it is vital to compute the posterior before integrating the most recent perceptual data. Such a belief is often called prediction, and it represents as follows.

$$\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{1.7}$$

Only based on the prior posterior, the above prediction can estimate the environment state at a time t before integrating the most recent perceptual data. In probabilistic robotics, computing the posterior $bel(x)$ from the prediction $\overline{bel}(x)$ is called measurement update or correction [25].

1.2.5. **Bayes filter**

Bayes filter is considered as the fundamental algorithm for computing the posterior using control and perceptual data. This filter is recursive; it estimates the belief at time t from the previous belief one-time step before [11,25]. Table 1.1 [25] shows the primary Bayes Filter with only a single iteration (update rule). The inputs for the Bayes Filter are the prior belief, most recent control data, and most recent perceptual data, while its output is the most recent belief at the current time.

Table 1.1. Bayes Filter

```
1:  **Bayes_Filter_Algorithm** $(bel(x_{t-1}), u_t, z_t)$
2:      for all $x_t$ do
3:          $\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)\, bel(x_{t-1})\, dx_{t-1}$
4:          $bel(x_t) = \mu\, p(z_t|x_t)\, \overline{bel}(x_t)$
5:      end for
6:      return $bel(x_t)$
```

The calculations of the current belief using Bayes Filter are done in two important stages: prediction (control update) (line3) and correction (measurement update) (line4). In the prediction step, Bayes Filter computes the belief over the current environment state based on the state transition probability and last belief over the prior environment state. Notably, the predicted belief is calculated via integration (summation) of the product of two probability densities. On the other side, the correction phase addresses the probability of observing the most recent sensor data at the current state, and this is done by multiplying the measurement probability by the predicted belief. However, as the continuous probability density integrates to one, the corrected belief could not integrate to one. So, the returned belief is normalized by the normalization factor.

Bayes filtering algorithm invokes the initial belief at zero time to calculate the next belief recursively. So, the boundary condition bel(x_0) should be provided to the algorithm as input at initialization. if the initial state x_0 is determined with certainty, then the initial posterior belief distribution initialized around the actual initial state as point mass distribution. At the same time, zero probability should be assigned elsewhere. On the other hand, if the initial state x_0 is unknown, based on the uniform distribution, the initial posterior density should be initialized over all possible states in state space. Finally, the belief is voiced by nonuniform distributions when the initial state is partly known.

1.3. **Markov Localization**

Probabilistic pose estimation methodologies are variants of the Bayes filter. In localization problems, Bayes filter is however called Markov localization [31]. The basic Markov algorithm illustrated in Table 1.2 [25] is developed from the Bayes filter algorithm depicted in Table 1.1. As we note, the map of the robot environment has been added to the algorithm arguments and incorporated in the measurement model and (but not always) plays a role in the motion model.

Table 1.2. Markov localization

```
1:  **Markov_Localization_Algorithm** $(bel(x_{t-1}), u_t, z_t, m)$
2:      for all $x_t$ do
3:          $\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t, m)\, bel(x_{t-1})\, dx_{t-1}$
4:          $bel(x_t) = \mu\, p(z_t|x_t, m)\, \overline{bel}(x_t)$
5:      end for
6:      return $bel(x_t)$
```

In analogy with the Bayes filter, the Markov localization algorithm estimates the beliefs recursively. Moreover, Markov localization can solve the localization problems (position tracking, global localization, and kidnapped robot problem) in static worlds. However, the initial belief is initialized in three forms based on the localization problem, and it represents the initial knowledge of the robot's state.

- Position tracking. When the initial state of the mobile robot is known, then the initial belief is initialized by a discrete distribution such as Point-Mass Distribution (PMD), which does not have a density. Suppose $\tilde{x}_0$ indicates the actual initial robot's pose, then the value of initial belief will be:

$$bel(x_0) = \begin{cases} 1, & x_0 = x_0 \\ 0, & otherwise \end{cases} \tag{1.8}$$

However, in practical applications, the robot's initial state is roughly known. Therefore, the initial belief is typically initialized through such a narrow Gaussian distribution with a mean $\tilde{x}_0$. Normal Gaussian distribution for a scalar value x with a mean $\tilde{x}_0$ and variance σ^2 is given by Equation (1.9). Often, x is a multidimensional vector. In such a case, the normal distribution is called Multivariate normal distribution, as shown in Equation (1.10). Figure 1.3 below illustrates the Normal Gaussian distribution curve.

$$p(x) = \left(2\pi\,\sigma^2\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{\left(x - x_0\right)^2}{\sigma^2}\right\} \tag{1.9}$$

$$p(x) = \det\left(2\pi\Sigma\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(x - x_0\right)^T \Sigma^{-1}\left(x - x_0\right)\right\} \tag{1.10}$$

Here $\sum$ stands for the covariance matrix (positive semidefinite matrix) of the initial state uncertainty.

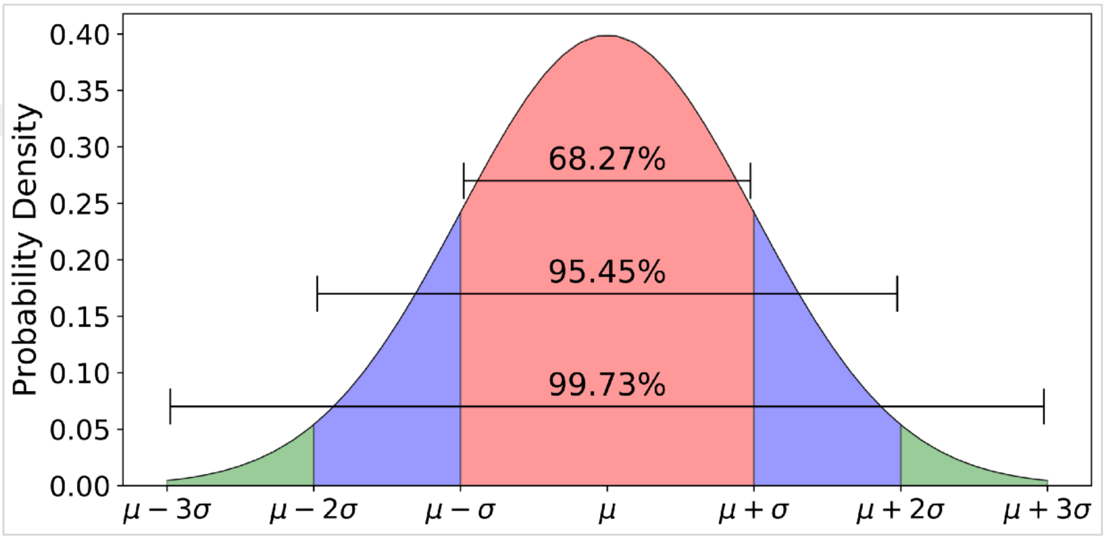


Figure 1.3. Normal Gaussian distribution (68-95-99.7 Rule)

- Global localization. This problem deals with the unknown initial state of the mobile robot. In this case, the initial belief is initialized via uniform distribution over all possible legal states in the robot's state space. So, the initial belief value is given by:

$$bel(x_0) = \frac{1}{V} \tag{1.11}$$

V refers to the volume of all states in the world of the mobile robot.

- Partial knowledge regarding the robot's pose. Sometimes the robot knows the characteristic of its current place, but cannot specify its exact pose, such areas as if the robot starts near a door or starts in a corridor. In this case, the initial belief can be represented by multi distributions. The uniform distribution is used to describe the belief in the expected areas while the density will be zero anywhere else.

From the above, we noticed that the Markov localization algorithm is independent of the state space representation, and it can be realized utilizing various techniques of state representation. In the next section, we will discuss the most effective practical algorithms from the robotic literature that can perform real-time pose estimation. We will start with the grid representation (discrete representation); after that, we will address an algorithm that employs particle filter for state representation.

1.4. **Localization Algorithms**

Localization problem has received great attention in the robotic literature, as it was considered the key and first question to the Autonomous Guided Vehicle (AGV) navigation problem and one of the essential challenges in autonomous mobile robots [32]. The control scheme of the autonomous mobile robot systems shown in Figure 1.4 [6] illustrates how the mobile robot can navigate autonomously in its environment. As we see, the localization problem should be solved first to answer the question (Where am I?) where the robot after that can start the navigation process.

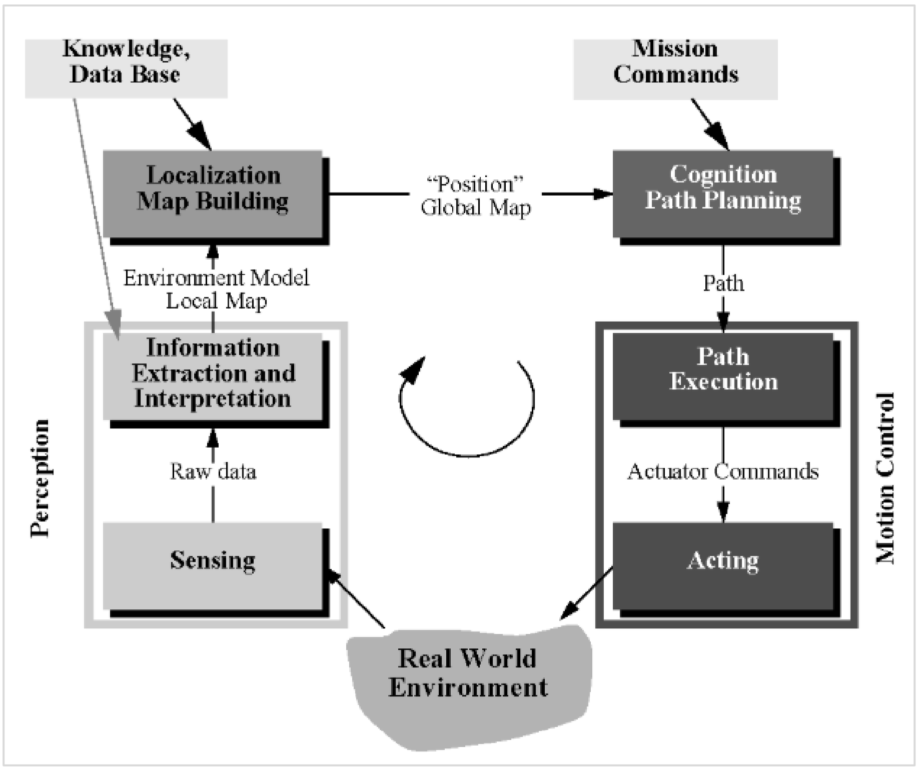


Figure 1.4. Control scheme for mobile robot systems

For instance, the AGV robot shown in Figure 1.5 [6] (a) below by SWISSLOG used to carry motors from one assembly line to another. This robot uses electrical guidewires built-up on the floor as a localization technology that allows the robot to know its position in real-time. AGV robot shown in Figure 1.5 (b) outfitted with BlueBotics Autonomous Navigation Technology (ANT). Instead of relying on electrical guidewires, this robot utilizes onboard safety lasers to do localization within a pre-defined map. Figure 1.5 (c) shows the Helpmate AGV used to transport medication and food in the hospital. Helpmate has different on-board sensors that help it to navigate autonomously through corridors. The installed lights on the ceiling play a role as landmarks, where the robot's camera can detect these landmarks for localization purposes.

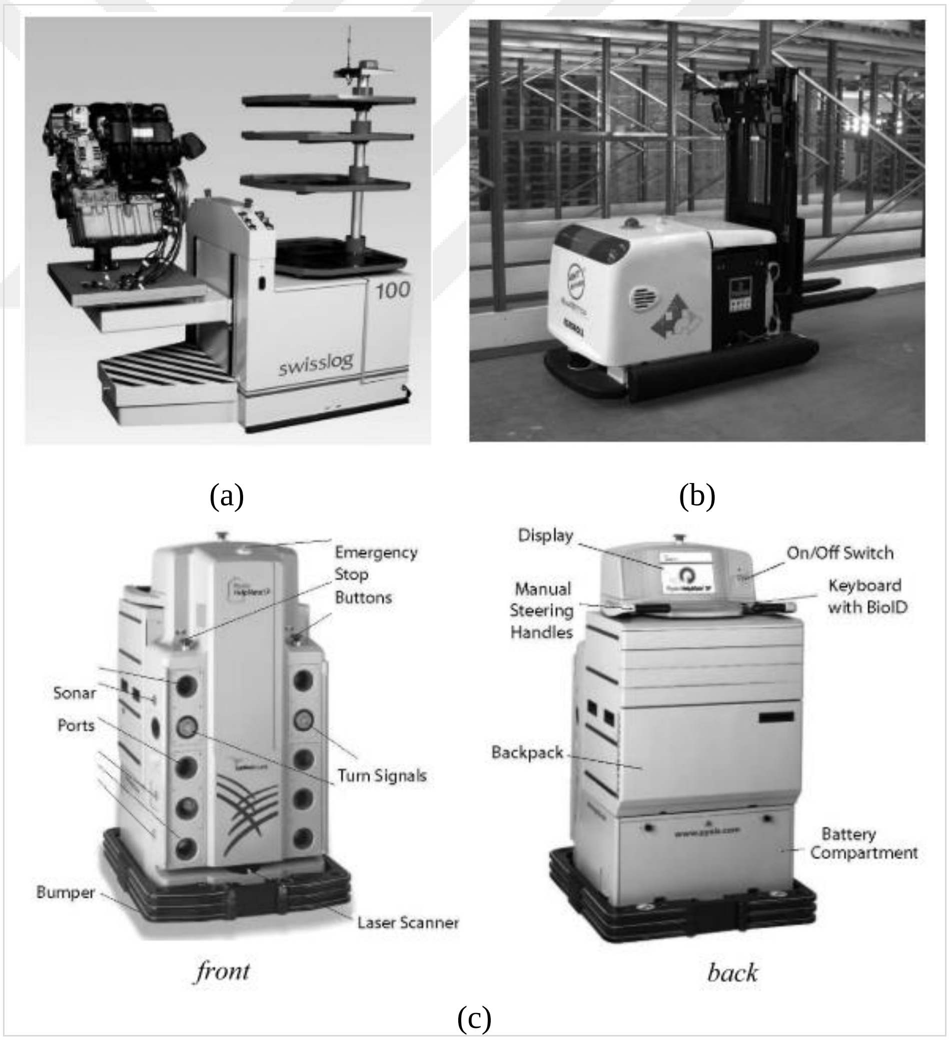


Figure 1.5. Different designs of autonomous guided vehicles

Localization methodologies are variants of the Markov localization algorithm. In the probabilistic context, we can find several localization algorithms that can help mobile robots to be autonomous. One of these algorithms uses Gaussian filters, such as EKF. As the Gaussian distribution is a uni-modal, the Kalman filter can handle the assumption of local uncertainty. Thus, EKF only solves the position tracking problem where the initial robot's pose is known in approximate. EKF guarantees accuracy for the system to which Gaussian noise is applied and only for a linear system or systems not dramatically nonlinear. On the other side, the global localization problem requires multi-modal distribution to represent the created uncertainty. However, EKF cannot cover the global localization problem. Multi-Hypothesis Tracking (MHT) [33] was introduced to overcome Gaussian limitations. MHT represents the robot's pose belief by employing multiple normal Gaussian distributions. As a result, multiple Gaussians can address the global localization problem, but this will come up at a very high computational cost.

In contrast, Grid-based localization and MCL algorithms can deal with the multi-modal distribution and address the created global uncertainty with high performance. Therefore, these algorithms are appropriate for global localization and kidnapped robot problems.

1.4.1. **Grid-based localization**

Grid localizer is a modified version of the discrete Bayes filter. This localizer utilizes the histogram filter to estimate the beliefs over the grid of state space. The following Figure 1.6 [25] depicts a grid decomposition example. The robot's map is decomposed into several grids. Each cell denotes a robot's state in its world, and each grid layer denotes a different robot head direction.

Table 1.3 [25] shows the grid-based localization algorithm. Here, x_k stands for the grid cell, and as the grid localizer is based on the discrete Bayes filter, it handles a belief as a set of discrete values of probability $bel(x_t) = \{p_{k,t}\}$, where $p_{k,t}$ itself is the probability of a mobile robot to be in grid cell k in real-time. $\mathrm{mean}(x_k)$ denotes the mass center of the grid cell. For inputs, as a recursive filter, the localizer asks for the prior discrete probability set $\{p_{k,t-1}\}$, the current sensor data z_t, control data $\mathrm{u_t}$, and the robot's environment map m. At each iteration, the algorithm updates all grid cell's probability. The motion model in line 3 includes the control data, while the sensor data incorporated in the measurement model in line 4.

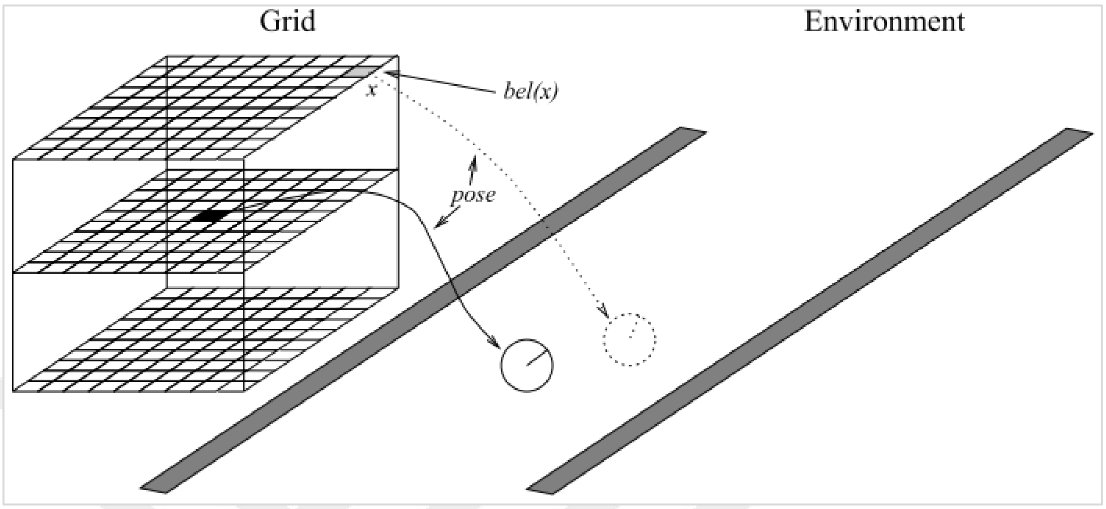


Figure 1.6. Grid decomposition example over the robot's state space. (appears just three orientations)

Table 1.3. Grid localization algorithm

```
1:  Grid_Localization_Algorithm ({$p_{k,t-1}$}, $u_t$, $z_t$, $m$)
2:  for all $k$ do
3:      $\bar{p}_{k,t} = \sum_i p_{i,t-1}\textbf{motion\_model}(mean(x_k), u_t, mean(x_i))$
4:      $p_{k,t} = \mu\, \bar{p}_{k,t}\textbf{measurement\_model}(z_t, mean(x_k),, m)$
5:  end for
6:  return {$p_{k,t}$}
```

Two challenges face grid localization during implementation. The first is the trade-off between the results accuracy and processing time. The accuracy of the returned value from the algorithm relies on the grid cell resolution. With a fine grid, the results are very accurate, but the fine grid imposes a higher computational burden leads to a very slow localization. However, the coarse grid leads to losing some information through discretization, and the results will be inaccurate. The second challenge is related to the motion model, especially when the algorithm employs a high-resolution sensor model with coarse grid cells. The results may be poor as the grid localizer describes the grid cell by its center mean(x_k).

1.4.2. **Monte Carlo Localization (MCL)**

1.4.2.1. **Monte Carlo Localization methodology**

MCL is a localization technique based on PF [11]. It provides a new methodology to cope with multi-modal distribution, where it can localize the mobile robot in a global environment. With this technology, the created uncertainty in the robot pose estimation is represented by addressing a particle set which is randomly inferred from PF [34]. Most of the problems in the real world are nonlinear systems, so rather than describing the robot world by Probability Density Function (PDF), MCL represents such an arbitrary distribution shown in Figure 1.7 by a bunch of particles. Each particle created on the state space represents a hypothesis state of the mobile robot.

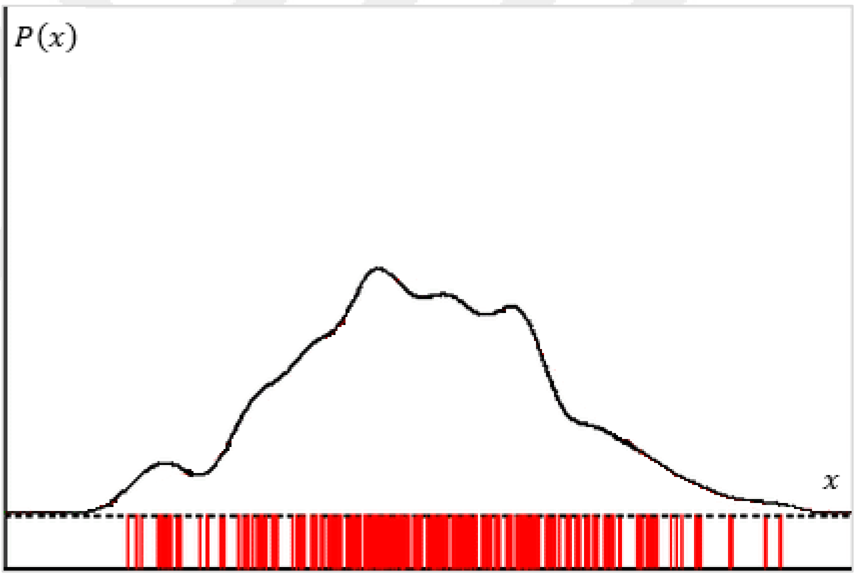


Figure 1.7. An arbitrary distribution represented by a set of particles

Representing the belief $bel(x_t)$ by a set of particles allows the MCL the ability to solve position tracking problem, global localization, and even kidnapped robot problem. So, when the initial robot's pose is unknown, MCL tries to estimate the robot's state by generating equal-probability particles uniformly distributed all over the robot space. Although MCL is a comparatively new approach in the world of autonomous mobile robots, it has already become one of the most popular pose estimation techniques in robotic systems. Moreover, the implementation of the MCL algorithm is easy. And in order to understand how the MCL algorithm works and how it is implemented, first and foremost, we will explain the particle filter in the next section.

1.4.2.2. **Particle Filter (PF)**

PF is an advanced form of recursive Bayes filter. It represents the belief distribution by a collection of random particles estimated from this distribution, as shown in Figure 1.7. Each particle generated from the corresponding posterior distribution represents a potential estimated state in real-time. So, PF is used by the MCL technique to estimate the robot’s pose in global environments and when the robot has a kind of unreliable sensor. The key idea of PF is the samples, where it uses multiple samples to represent any arbitrary distribution from our real life. Therefore, the sampling procedure is called PF. Each sample has its weight; the higher the weight means higher the probability of the corresponding area, and the density of the particles within a specified area describes the density of distribution. Equation (1.12) depicts a set of weighted samples, where each sample m includes a state hypothesis $x_t^{[n]}$ and its importance weight $w^{[n]}$. The more samples in the sample set, the better the approximation.

$$X = \left\{ \left\langle x^{[n]}, w^{[n]} \right\rangle \right\}, \qquad n=1,...,N \tag{1.12}$$

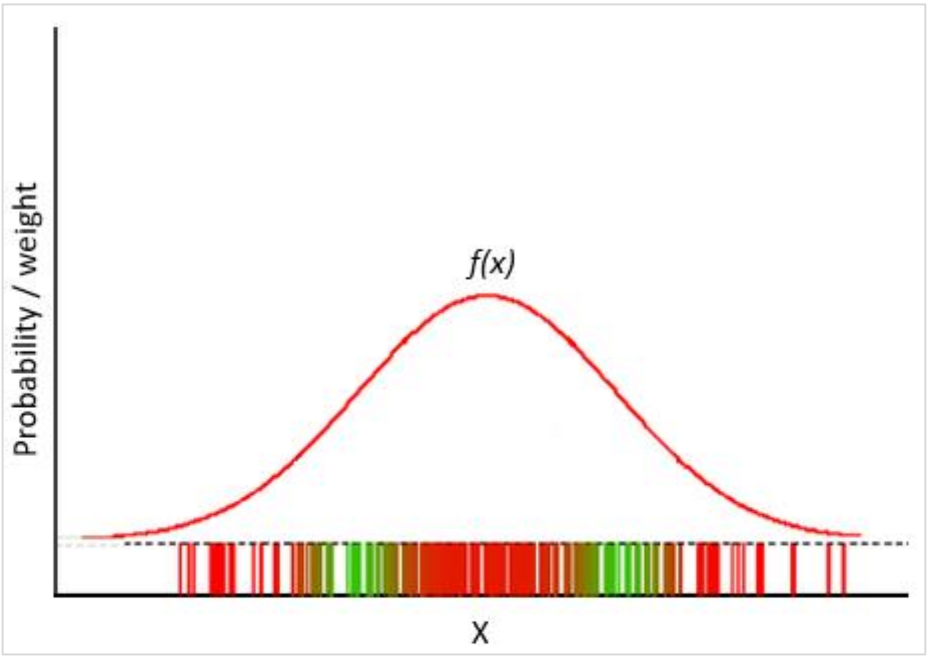


Figure 1.8. Generating samples from Gaussian distribution by closed-form sampling

For Gaussian distributions, such as one illustrated in Figure 1.8, the samples are generated using the closed-form sampling method [35], as shown in Equation (1.13) below. The technique is by taking 12 random numbers between minus plus of the standard deviation of Gaussian distribution, then sum all of them up and divide it by two.

$$x = \frac{1}{2}\sum_{i=1}^{12} rand\left(-\sigma,\sigma\right) \tag{1.13}$$

However, closed-form sampling is only applicable for a few distributions, such as Gaussian distributions. Unfortunately, in most cases, the target distribution has an arbitrary form where we cannot generate samples by closed-form sampling. PF uses importance sampling principle to generate samples from an arbitrary distribution. As shown in Figure 1.9 below, PF uses a different distribution called proposal distribution, such as Gaussian to generate samples from the target distribution. This process is called sampling. Then, in the importance process, PF accounts for the difference between the target and proposal distributions by calculating the importance weight for each generated sample. MCL algorithm utilizes motion model $p(x_t \mid x_{t-1},\ u_t)$ as a proposal function and measurement model $p(z_t \mid x_t)$ to calculate the importance weight for each sample.

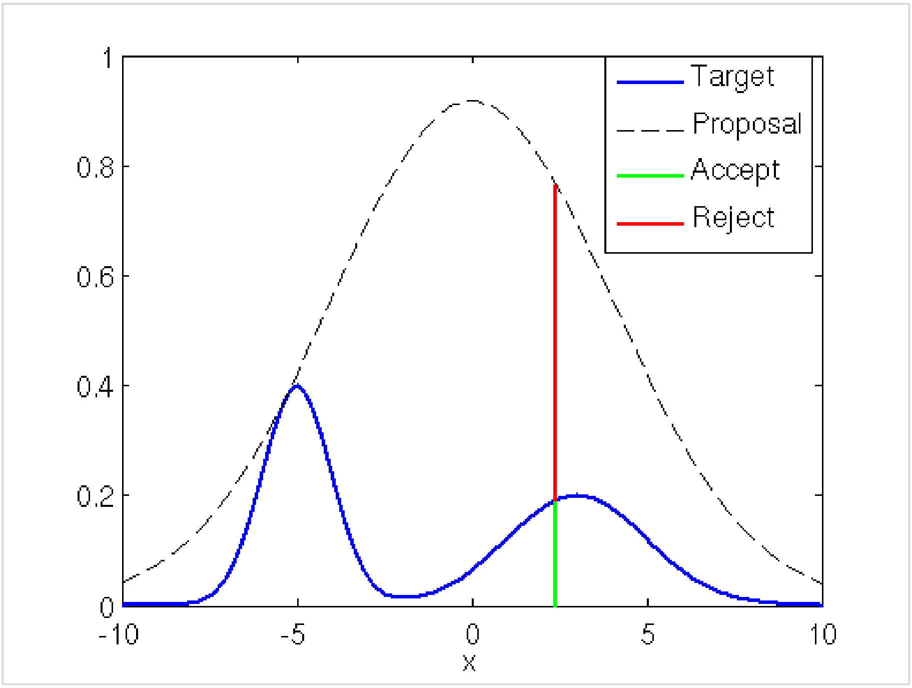


Figure 1.9. Importance sampling principle

Through the following steps below, PF can estimate the actual state precisely [36]. Steps 2 to 5 are frequently executed to infer the pose of the mobile robot. In other words, it is a technique of estimating the robot's pose by updating the distribution of the samples, which represents the probability of the robot's pose based on the perceptual data.

- Initialization. In global localization, samples are randomly distributed over all possible states. While the robot's pose is unknown, the weight of all samples is sum up to one.
- Prediction (sampling). Based on the motion model that uses the previous sample set, PF predicts the current new samples.
- Update (Importance). PF uses the measurement model to correct the pose estimate by calculating the weight of each sample.
- Pose estimation. The pose and weight of all samples are used to calculate the maximum weight value, median value, and the average weight for estimating the robot's pose.
- Resampling. PF gets rid of less weighted samples and generates new samples that inherit the characteristic of samples with a high importance weight. The weight of all samples is sum up to one again.

After two iterations of prediction and correction, the filter will resample the particles. And after a complete filter iteration, the algorithm gives a new bunch of weighted samples (new distribution) that represent the probability of the robot's pose.

Table 1.4 [25] below, illustrates the PF algorithm. As the PF constructs the next posterior recursively from the previous posterior, the inputs include the previous sample set X_{t-1} along with the current control data u_t and most recent sensor data z_t. The algorithm iterates over all samples in the previous sample set, where in the line 4 the proposal function (action model) is used to generate a new sample. In line 5, PF uses the sensor model to compute the importance factor (weight) for each generated sample. The state hypothesis and its weight were added together in line 6 to create the predicted sample set. Line 8 to 11 represent the resampling process, where PF keeps tracking the actual pose. In fact, resampling is the trick of PF. If the number of the used samples is infinite, the filter does not care if there are a large number of samples going to unlikely regions, as long as it has enough samples in the areas with a high likelihood. But in all practical applications, the number of used samples that we can represent in the computers is finite. So, resampling will let the bad samples (samples with low weight) die out and use the sample actually to represent the high likelihood areas. In practice, the resampling step is essential because if the PF does not use the resampling, it will not be able to track the pose of a robot over long periods as the PF will diverge. Every sample in the sampling process makes an error. These errors will accumulate. So, there is no way to recover it unless having an infinite number of samples or do a resampling.

Table 1.4. Particle filter algorithm

```
1:   Particle_Filter_Algorithm ($X_{t-1}, u_t , z_t$)
2:       $\bar{X}_t = X_t = \emptyset$
3:       for $n = 1$ to $N$ do
4:           Sample:  $x_t^{[n]} = p(x_t|x_{t-1}^{[n]},\ u_t)$
5:           Correct:  $w_t^{[n]} = p(z_t|x_t^{[n]})$
6:           $\bar{X}_t = \bar{X}_t + \langle x_t^{[n]}, w_t^{[n]}\rangle$
7:       endfor
8:       for $n = 1$ to $N$ do
9:           draw $i$ with probability $\propto\ w_t^{[i]}$
10:          add $x_t^{[i]}$ to $X_t$
11:      endfor
12:      return $X_t$
```

Finally, PF is a non-parametric approach, which means it does not utilize a particular function to represent the belief. Instead, it uses random samples to represent the posterior distribution.

2. MONTE CARLO LOCALIZATION ALGORITHM

MCL algorithm is considered one of the subset approaches that can successfully deal with the created global uncertainty. And as a probabilistic approach, MCL can compute the instantaneous uncertainty of a mobile robot, and it is convenient to local and global pose estimation problem. Moreover, MCL is easy to implement and can solve the kidnapped and global localization problems in a very high robust and competent way. The high efficiency of the MCL algorithm comes from the fact that it represents the uncertainty (robot pose) by a collection of particles, which are randomly generated over the robot poses in accordance to the posterior distribution. So, MCL uses PF to cope with multi-modal distributions and achieving pose tracking for autonomous mobile robots against a known map.

MCL employs the term sample rather than a particle, where each sample represents the predicted robot pose over the state space. In analogy with PF, MCL proceeds in these primary stages:

- Initialization. All samples are randomly scattered all over the state space via uniform distribution. The total weight of the samples is one.
- Prediction phase. Samples are distributed based on the change in the robot's state and the motion model.
- Correction phase. Based on the probability of receiving the sensor reading for each sample, the weight of the samples is assigned.
- Pose estimation. Based on the calculated importance weights, the robot can estimate its pose. The set of samples with the highest weight is utilized to infer the robot pose.
- Resampling. It is a crucial key for adapting to changes and maintain relevant samples to estimate the robot's pose.

The algorithm of MCL is derived from the PF algorithm that is explained in Table 1.4 above. Besides, MCL incorporates the map of the robot environment in inputs and utilizes motion model function $p(x_t \mid x_{t-1},\ u_t)$ as a proposal function to calculate the predicted hypothesis states (line 4), and measurement model function $p(z_t \mid x_t)$ to calculate the importance weight for each sample (line 5). Table 2.1 [25] below illustrates the algorithm of the MCL technique.

Table 2.1. Monte Carlo Localization algorithm

```
1:   **MCL_algorithm** $(X_{t-1}, u_t, z_t, m)$
2:       $\bar{X}_t = X_t = \emptyset$
3:       for $n = 1$ to $N$ do
4:           $x_t^{[n]} =$ **sample_motion_model** $(x_{t-1}^{[n]}, u_t)$
5:           $w_t^{[n]} =$ **measurement_model** $(z_t, x_t^{[n]}, m)$
6:           $\bar{X}_t = \bar{X}_t + \langle x_t^{[n]}, w_t^{[n]} \rangle$
7:       endfor
8:       for $n = 1$ to $N$ do
9:           draw $i$ with probability $\propto w_t^{[i]}$
10:          add $x_t^{[i]}$ to $X_t$
11:      endfor
12:      return $X_t$
```

In the next two sections, the motion and measurement models used in our thesis will be explained in detail. We will focus totally on mobile robots that working on the planer world. Moreover, differential drive mobile robots are used in our work.

2.1. **Probabilistic Motion Models**

Motion models or probabilistic kinematic model includes the state transition probability $p(x_t \mid x_{t-1},\ u_t)$. And it plays a crucial role in the prediction phase of the MCL. The motion model function is a representation of the kinematics of a robot and can predict how samples change their poses over time. Without the probabilistic kinematic model, it is difficult to predict the next move. There are two probabilistic motion models in the robotic literature. The first is the velocity motion model, while the other is the odometry motion model. Both of them are utilized for mobile robots moving in flat environments. The velocity motion model supposes that the control data u_t provides velocity commands to the motors of the mobile robot. Most of the industrial mobile robots, such as synchro drive or differential drive, are driven by independent rotational and translational velocities. However, the odometry motion model assumes that the control data $\mathrm{u_t}$ contains the relative motion information which has access to the odometer data. Odometry information provided by kinematic information (travel distance and the angle turned).

While both motion models suffer from drift and slippage, practical experience suggests the odometry motion model over the velocity motion model to solve localization problems. The reason, the accuracy of the odometry model is higher than the velocity model, where the velocity model also suffers from a mismatch between the actual motion controllers and its mathematical model. In other words, in most mobile industrial robots, the velocity measured by the revolution of the mobile robot's wheels does not fully match the executed velocity command. However, odometry information is only obtainable in retrospect, which means after the motion command is executed. Therefore, the odometry model cannot be employed for planning algorithms. So, the velocity motion models are utilized for probabilistic motion planning, whereas odometry motion models are applied for localization.

2.1.1. **Odometry motion model**

In our thesis, we assumed a differential drive mobile robot moves on a planer environment. The mobile robot is outfitted with incremental encoders that measure the rotation of the wheels, but not directly the position and orientation of the robot to a fixed world frame. Wheel sensors, such as incremental optical encoders, are a type of proprioceptive sensors that measure the dynamics and internal state of a robot. Wheel sensors used in mobile robots have a low cost and high quality that provide excellent resolution. A map-free odometry motion model is utilized to calculate the motion of the mobile robot over time. The odometry motion model is typically created by incorporating the motion information from wheel encoders. Therefore, the robot's odometer is sensor measurements, but we treated it like controls.

Based on PF, odometry motion model generates random samples from the state transition probability $p(x_t \mid x_{t-1},\ u_t)$. The following Figure 2.1 illustrates the robot motion from a state x_{t-1} to state $\mathrm{x_t}$. However, odometry employs relative motion information in pose estimation. Here, the motion information (control u_t) is given by the two successive poses as below:

$$u_t = \begin{bmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{bmatrix} \tag{2.1}$$

The bar sign above poses demonstrates that the odometry measurements are given relative to the internal coordinate system of the robot whose relationship to the world frame is unknown.

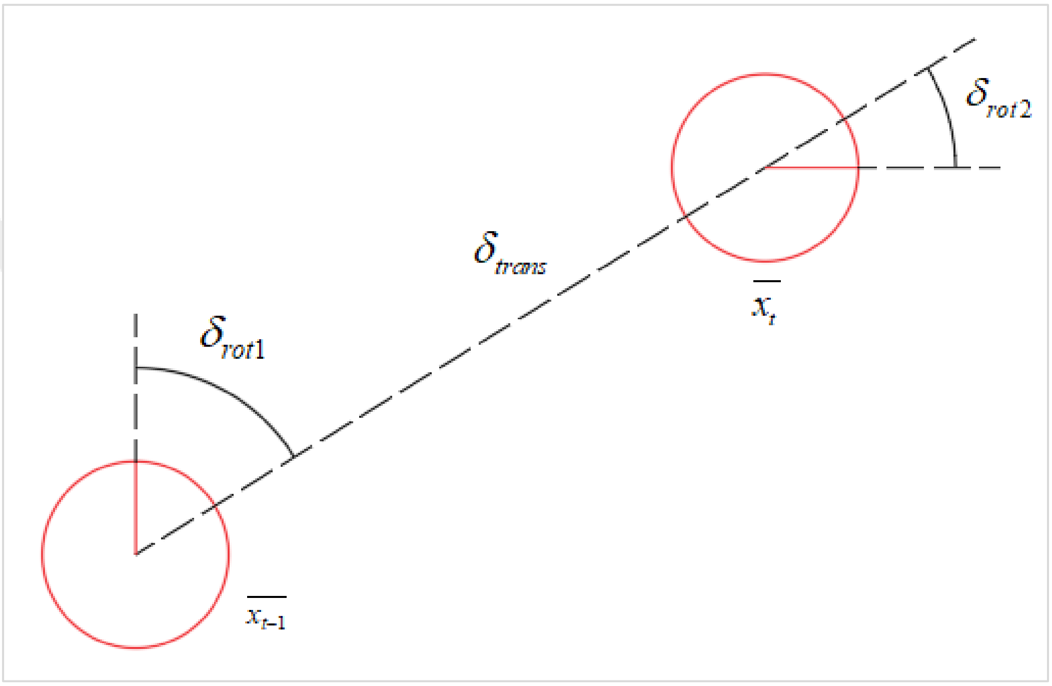


Figure 2.1. Odometry information from sensor measurements

Odometry information is represented by a sequence of three parameters as a result of the relative difference between two successive states. Initial rotation δ_{rot1} followed by translation δ_{trans}, and finally, the second rotation δ_{rot2}, as shown in Figure 2.1 above. Equation (2.2) below shows how to extract these three parameters from odometry reading u_t.

$$\begin{bmatrix} \delta_{rot1} \\ \delta_{trans} \\ \delta_{rot2} \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}\left(\overline{dy}, \overline{dx}\right) - \bar{\theta} \\ \sqrt{(\overline{dx})^2 + (\overline{dy})^2} \\ \overline{d\theta} - \delta_{rot1} \end{bmatrix} \tag{2.2}$$

Odometry information is noise-free. And the probabilistic approach postulates that there is independent noise incorporated with the odometry information. For example, if the Odometry says that the mobile robot moves one meter forward, the pose estimation algorithm takes a sample out from the previous sample set and move it a meter forward by applying control u_t. Then may add some sampling noise around this sample, where the measured motion is given by the actual motion corrupted with noise. So, modeling the motion error is done by subtracting or adding independent noise ϵ_{σ^2} to the odometry information. We can generate noise samples from Gaussian normal distribution with zero mean and variance σ^2 by using the closed-form principle shown in Equation (1.13). Equation (2.3) demonstrates noise modeling for odometry information.

$$\begin{bmatrix} \delta_{rot1} \\ \delta_{trans} \\ \delta_{rot2} \end{bmatrix} = \begin{bmatrix} \delta_{rot1} - \varepsilon_{\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2} \\ \delta_{trans} - \varepsilon_{\alpha 3 \delta_{trans}^2 + \alpha 4 (\delta_{rot1}^2 + \delta_{rot2}^2)} \\ \delta_{rot2} - \varepsilon_{\alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2} \end{bmatrix} \tag{2.3}$$

Where, α_1 to α_4 stand for the motion noise parameters (error parameters). α_1 refers to the rotational error due to rotational motion, α_2 indicates the rotational error due to translational motion, α_3 indicates the translational error due to translation motion, and α_4 refers to the translational error due to rotational motion.

As a variant of recursive Bayesian filters, the estimated robot's pose x_t is obtained from the robot's pose one step time back x_{t-1} by the first rotation δ_{rot1}, followed by a straight motion δ_{trans}, finally by the second rotation δ_{rot2}. Thus, the odometry motion model suggests the estimated real robot's pose, as shown in the following Equation (2.4).

$$\overrightarrow{x_t} = \overrightarrow{x_{t-1}} + \begin{bmatrix} \delta_{trans} \ \cos\left(\theta + \delta_{rot1}\right) \\ \delta_{trans} \ \sin\left(\theta + \delta_{rot1}\right) \\ \delta_{rot1} + \delta_{rot2} \end{bmatrix} \tag{2.4}$$

The following Table 2.2 [25] demonstrates the odometry motion model algorithm. The algorithm takes the previous robot's pose and the most recent control as input, and it returns the estimated robot's pose at the current time as output. Lines 2 to 4, represent free noisy Odometry measurement, lines 5 to 7 represent noise modeling for odometry information. The function sample (σ^2) generates a random sample from a zero centered distribution with variance σ^2. Finally, lines 8 to 10 calculates the estimated robot's pose at the current time.

Table 2.2. Sample odometry motion model

```
1:    Sample_Odometry_Motion_Model_Algorithm ($x_{t-1}, u_t$)
2:        $\delta_{rot1} = atan2(\overline{dy}, \overline{dx}) - \theta$
3:        $\delta_{trans} = \sqrt{(\overline{dx})^2 + (\overline{dy})^2}$
4:        $\delta_{rot2} = \overline{d\theta} - \delta_{rot1}$

5:        $\hat{\delta}_{rot1} = \delta_{rot1} - \textbf{sample}(\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2)$
6:        $\hat{\delta}_{trans} = \delta_{trans} - \textbf{sample}(\alpha_3 \delta_{trans}^2 + \alpha_4 (\delta_{rot1}^2 + \delta_{rot2}^2))$
7:        $\hat{\delta}_{rot1} = \delta_{rot2} - \textbf{sample}(\alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2)$

8:        $\acute{x} = x + \hat{\delta}_{trans} \; cos\,(\theta + \hat{\delta}_{rot1})$
9:        $\acute{y} = y + \hat{\delta}_{trans} \; sin\,(\theta + \hat{\delta}_{rot1})$
10:       $\acute{\theta} = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
11:       return $x_t = [\acute{x} \quad \acute{y} \quad \acute{\theta}]^T$
```

2.1.2. Robot forward kinematics model

Most of the industrial mobile robots, such as differential drive shown in Figure 2.2 below, are driven by independent rotational and translational velocities. Thus, the orientation and position of mobile robots can be calculated using forward kinematics.

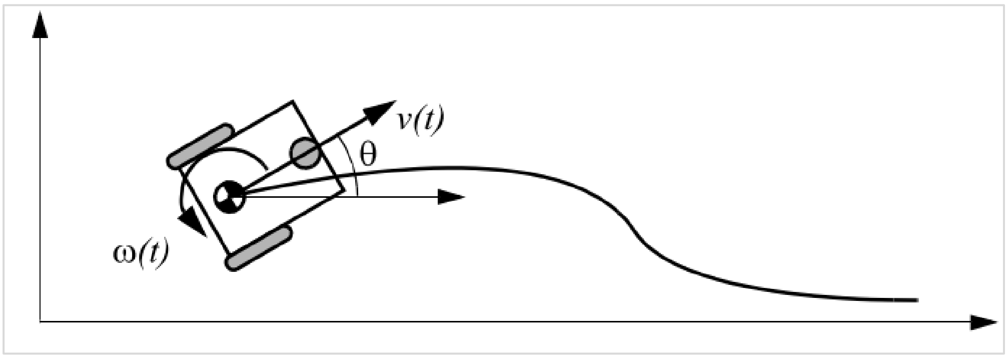


Figure 2.2. Differential drive mobile robot driven by rotational and translational velocities.

Our robot has 3 Degrees of Freedom (DOF), and can only move in XY-plane. However, the differential drive system is non-holonomic, where some of the DOFs are uncontrollable [37]. Differential drive mobile robot changes its heading orientation by differentiating the relative rotation rate of wheels. So, for example, differential drive is unable to move sideways. While in the holonomic system, the number of controllable DOFs equals the total number of DOFs. By utilizing mecanum or omni wheels, mobile robots can change their locations without changing the direction of their orientation [38]. However, our differential drive mobile robot is equipped with incremental encoders that measure the rotation of the wheels. The motion information u_t used in the algorithm of the odometry motion model shown in Table 2.2 is obtained based on the robot's wheels rotation measurements.

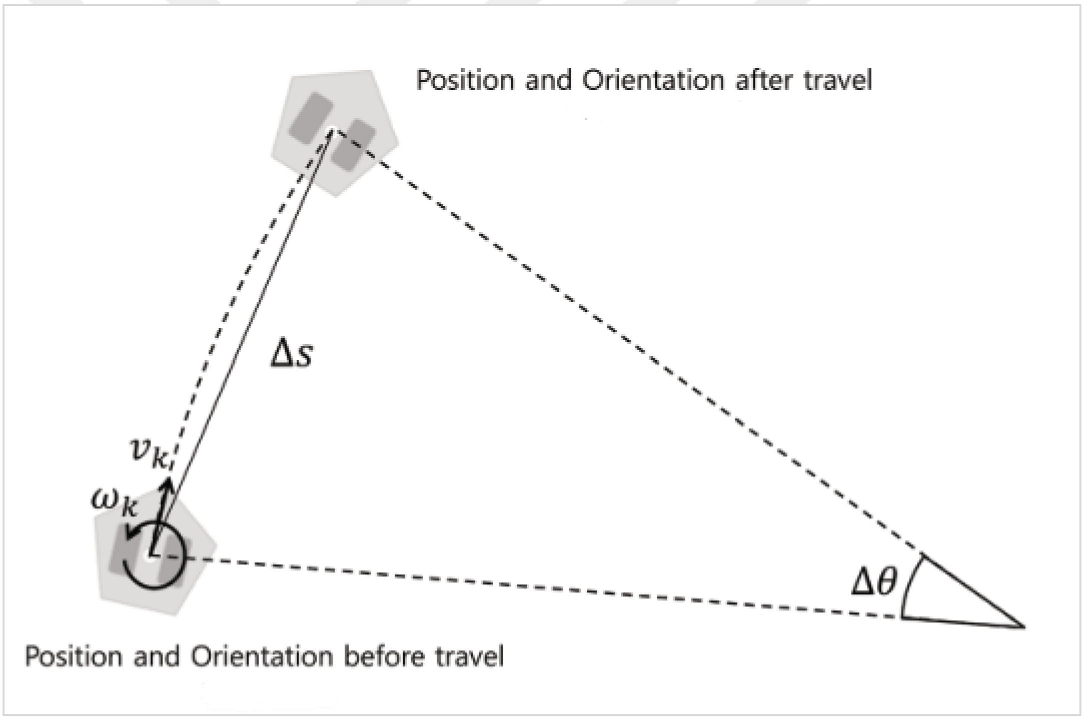

Position and Orientation after travel
Δs
v_k
ω_k
$\Delta\theta$
Position and Orientation before travel

Figure 2.3. Differential drive mobile robot changes its pose during the time.

For a discrete system with a fixed sampling interval Δt, we assumed the movement of a robot as incremental distances. The mobile robot illustrated in Figure 2.3 above, traveled Δs distance in a short time Δt. Relying on the robot's odometer, Equations (2.5) and (2.6) calculates the rotational speed of the right and left wheels respectively.

$$v_r = \frac{\left(Enc_{r,t} - Enc_{r,t-1}\right)}{\Delta t} \cdot \frac{\pi}{180} \tag{2.5}$$

$$\nu_l = \frac{\left(Enc_{l,t} - Enc_{l,t-1}\right)}{\Delta t} \cdot \frac{\pi}{180} \tag{2.6}$$

Where, $(Enc_{r,t}, Enc_{r,t-1})$ is the current and previous right encoder values respectively. while $(Enc_{l,t}, Enc_{l,t-1})$ are the current and previous left encoder values respectively. The rotational speed of wheels in (radian/sec).

If the diameter of the wheels is D and the distance between two wheels is L, then the velocity of the right and left wheels is calculated as in Equations (2.7) and (2.8) respectively. And the linear velocity and angular velocity of the mobile robot can be calculated as shown in Equations (2.9) and (2.10).

$$V_r = \nu_r \cdot \frac{D}{2} \tag{2.7}$$

$$V_l = \nu_l \cdot \frac{D}{2} \tag{2.8}$$

$$\upsilon_t = \frac{(V_r + V_l)}{2} \tag{2.9}$$

$$\omega_t = \frac{(V_r - V_l)}{L} \tag{2.10}$$

Equations (2.11) and (2.12) calculates the path traveled during the last time duration.

$$\Delta s = \upsilon_t \cdot \Delta t \tag{2.11}$$

$$\Delta\theta = \omega_t \cdot \Delta t \tag{2.12}$$

Finally, Equations (2.13) calculates the robot's pose. The result of the following equation is considered as the motion information u_t for the odometry motion model.

$$\overrightarrow{x_t} = \overrightarrow{x_{t-1}} + \begin{bmatrix} \Delta s \cdot \cos\left(\theta_t + {}^{\Delta\theta}\!/_{2}\right) \\ \Delta s \cdot \sin\left(\theta_t + {}^{\Delta\theta}\!/_{2}\right) \\ \Delta\theta \end{bmatrix} \tag{2.13}$$

2.2. Probabilistic Measurement Models

Measurement models or observation models solve for the measurement probability $p(z_t \mid x_t,\ m)$. It plays an essential role in the correction phase of the MCL directly after the motion model, and it directly affects the robustness and performance of the mobile robot. Since PF uses proposal distribution rather than the target distribution to generate samples as illustrated in Figure 1.9 above, the observation model takes an essential role to account for the differences between these two distributions by calculating the importance factor (weight) for each created sample n.

$$w_t^{[n]} = \frac{\text{target\_distribution}}{\text{proposal\_distribution}} = p(z_t \mid x_t, m) \tag{2.14}$$

In the above equation z_t is the sensor measurements at time t, x_t indicates the predicted robot's pose, and m is the robot's environment map. So, the correction is done through the observation model. Global localization assumes that all samples are distributed normally over the state space with the same weight. However, in the updating phase, the weight of the samples will be updated according to the sensor measurements. Therefore, the sum of the weights after the correction will not be equal to one.

Probabilistic approaches provide different models to perform the measurement probability, such as beam-based model, likelihood field, and landmarks [25]. However, the characteristic of the measurement model relies on the sensor used. In industrial applications, there are a wide variety of sensors utilized by mobile robots, such as sonar sensors, the best model for these sensors is by depicting the reflection of the sound wave on surfaces. In contrast, the best model for visual sensors is done by projection geometry.

MCL algorithm typically uses the odometry motion model as proposal distribution. Therefore, the importance weight is corrected by the observation likelihood model. This model ignores the physical properties of the beam and just uses the endpoints to check for the obstacles; it is highly efficient and smooth to small changes in the robot's pose.

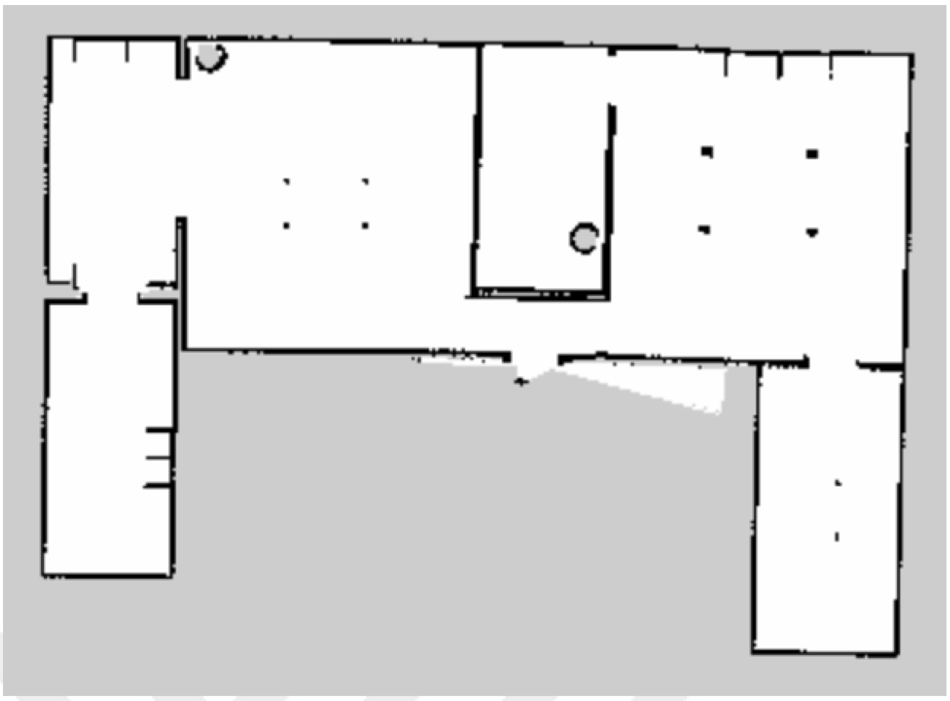


Figure 2.4. Occupancy grid map

The robot’s environment map is a table of objects in the world and their positions. A known map, such as Occupancy Grid Map (OGM) [39] shown in Figure 2.4 above, is used to help the mobile robot to localize itself. The white area in the map indicates the free space where the mobile robot can move, occupied area denoted by the black color, and the gray area stands for the unknown region.

2.2.1. Likelihood field range finder model

The likelihood field model uses 2D Light Detection and Ranging sensor (LiDAR) [40] to perceive the robot’s environment. 2D LiDAR is a laser distance sensor that employs the light to find the ranges from the sensor to the features around based on the time it takes for the laser to reflect. LiDAR sensor collects data ranges at given angle intervals. The measurement data obtained from the LiDAR sensor at a time t is a collection of b measurements, and it is indicated as follows:

$$z_t = \left[z_t^1, z_t^2, z_t^3, \ldots, z_t^b \right] \tag{2.15}$$

where z_t stands for the most recent sensor data, b refers to the individual beam from the current scan. However, the individual measurements are independent and the single beam z_t^b has a range value from zero to the maximum sensor range z_{max}.

Rather than accounting for the physical properties of the laser rays, the likelihood field model just uses the endpoints of the beams to check for the obstacles there. And the weight of each sample is allocated based on the calculated likelihood. The key idea is to match the obtained laser scan with the pre-mapped environment. The likelihood field represents the probability of obstacle detection as a function of global XY coordinates, as illustrated in Figure 2.5 (b).

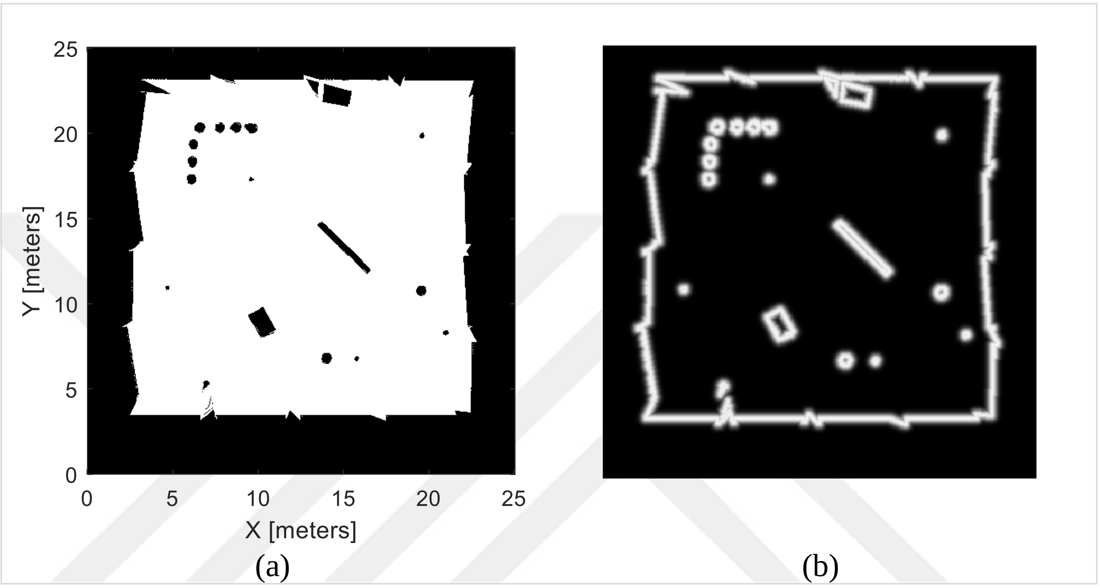


Figure 2.5. (a) OGM of the robot’s environment. (b) Likelihood field

Figure 2.5 (b) illustrates the likelihood field representation extracted from the OGM of the environment. The black regions of the map demonstrate that an obstacle is unlikely to be perceived there. The likelihood of the beam’s endpoint to hit an obstacle is calculated based on the distance transform, which indicates the minimum distance from the endpoint of the ray to the nearest obstacle. Relying on the normal Gaussian distribution shown in Equation (1.9) with a zero center, Equation (2.16) computes the likelihood of a single beam’s endpoint as follows.

$$likelihood\left(\mathrm{endpoint}^{b}\right)=\left(\frac{1}{\sigma\sqrt{2\pi}}\right)\cdot\exp\left(-\frac{d_b^2}{2\sigma^2}\right) \tag{2.16}$$

d_b refers to the distance transform of the beam b, while the standard deviation σ stands for the measurement noise. However, since the measurement noise is constant, the first term of Equation (2.16) can be normalized for all samples. Now, the likelihood of the individual beam is given in the following Equation (2.17).

$$p(z_t^b \mid x_t^{[n]}, m) = \exp\left(-\frac{d_b^2}{2\sigma^2}\right) \tag{2.17}$$

Equation (2.17) demonstrates that the likelihood of the beam's endpoint to hit an obstacle will decrease as the distance transform increases. However, the value of the distance transform at obstacles is zero; this leads to a high likelihood value (almost one) for the relevant beam's endpoint.

Now, the measurement probability $w_t^{[n]}$ of the expected robot's pose obtained from the odometry motion model can be calculated by multiplying the likelihood values of the scan endpoints, as shown in the following Equation (2.18).

$$p(z_t \mid x_t^{[n]}, m) = \prod_{b=1}^{b} p(z_t^b \mid x_t^{[n]}, m) \tag{2.18}$$

where $p(z_t \mid x_t^{[n]},\ m)$ is the weight of the sample n. The environment map m used to compute the distance transform for each grid cell. While the state $x_t^{[n]}$ used to apply scan data on each generated sample n. Since the obtained scan data is relative to the sensor pose, the endpoints of the scan should be projected into the world coordinate system. LiDAR sensor is installed on the robot, and its pose is assumed to be fixed over time. The following Figure 2.6 describes the sensor coordinate system relative to the global frame, and the general transformation mapping [41] of the sensor frame to the world frame is defined by Equation (2.19) below.

$${}^{w}P_{\text{sens},ORG} = {}^{w}P_{r,ORG} + {}^{w}_{r}R_z(\theta)\ {}^{r}P_{\text{sens,ORG}} \tag{2.19}$$

${}^{w}P_{sens,ORG}$ is the position vector locates the origin of the sensor frame relative to the world frame, ${}^{w}P_{r,ORG}$ refers to the position vector that describes the origin of the robot's frame relative to the world frame, ${}^{w}_{r}R_z(\theta)$ stands for the rotation matrix about z-axis which describes the orientation of the robot's frame relative to the world frame, and ${}^{r}P_{sens,ORG}$ is the position vector locates the origin of the sensor frame relative to the robot's frame.

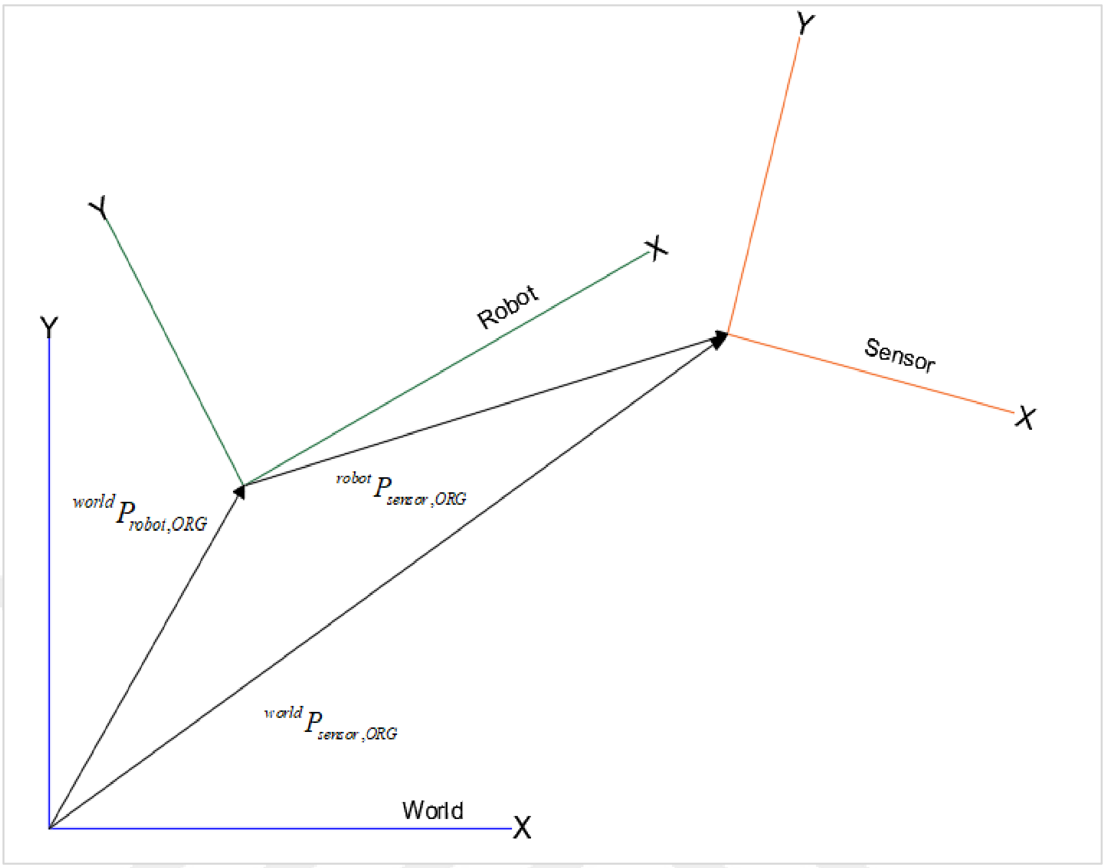


Figure 2.6. Sensor and robot coordinates in the world coordinate system, where each of these coordinates has a rotation about z-axis of the world frame

Now, Equation (2.20) describes the sensor frame relative to the world frame. where $[x, y, \theta]^T$ is the predicted state in world frame, and $[x_{sens}, y_{sens}]^T$ is the position coordinates of the sensor in the robot's frame.

$$^{w}P_{sens,ORG} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{sens} \\ y_{sens} \end{bmatrix} \tag{2.20}$$

Equation (2.21) describes the rotation of the sensor coordinates in the world coordinate system.

$$_{sens}^{w}R_z = {}_{r}^{w}R_z(\theta)\ {}_{sens}^{r}R_z(\theta_s) \tag{2.21}$$

$_{sens}^{W}R$ stands for the rotation matrix about z-axis which describes the orientation of the sensor frame relative to the world frame, while $_{sens}^{r}R_z(\theta_s)$ describes the orientation of the sensor frame relative to the robot's frame.

So,

$$
{}^{w}_{sens}R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta_s) & -\sin(\theta_s) \\ \sin(\theta_s) & \cos(\theta_s) \end{bmatrix}
$$

$$
= \begin{bmatrix} \cos(\theta+\theta_s) & -\sin(\theta+\theta_s) \\ \sin(\theta+\theta_s) & \cos(\theta+\theta_s) \end{bmatrix} \tag{2.22}
$$

θ_s denotes the angular orientation of the sensor relative to the heading direction of the mobile robot.

The target in the likelihood field representation is the scan endpoints. So, the following Equation (2.23) describes the endpoint coordinates in the world frame.

$$
{}^{w}P_{\mathrm{ep},ORG} = {}^{w}P_{sens,ORG} + {}^{w}_{sens}R \; {}^{sens}P_{\mathrm{ep,ORG}} \tag{2.23}
$$

${}^{sens}P_{ep,ORG}$ is the position vector locates the scan endpoint relative to the sensor coordinate system.

Now,

$$
{}^{w}P_{\mathrm{ep},ORG} = \left(\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{sens} \\ y_{sens} \end{bmatrix} \right) +
$$

$$
\begin{bmatrix} \cos(\theta+\theta_s) & -\sin(\theta+\theta_s) \\ \sin(\theta+\theta_s) & \cos(\theta+\theta_s) \end{bmatrix} \begin{bmatrix} z_t^b \cos(\theta_{ep}) \\ z_t^b \sin(\theta_{ep}) \end{bmatrix} \tag{2.24}
$$

from Equation (2.24), Equation (2.25) describes the trigonometric transformation used to map the scan endpoint into the world coordinate system.

$$
\begin{bmatrix} x_{z_t}^b \\ y_{z_t}^b \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{sens} \\ y_{sens} \end{bmatrix} + z_t^b \begin{bmatrix} \cos(\theta+\theta_s+\theta_{ep}) \\ \sin(\theta+\theta_s+\theta_{ep}) \end{bmatrix} \tag{2.25}
$$

$[x_{z_t}^b, y_{z_t}^b]^T$ stands for the endpoint coordinates in the global frame, θ_{ep} denotes the angular orientation of the laser ray relative to the sensor direction. Notice that the coordinates computed in Equation (2.25) are meaningless in the real world if the beam range has a maximum value of the sensor range $z_t^b = z_{max}$. Simply, the likelihood model rejects readings with maximum range.

We took for granted three different types of uncertainty and noise sources. The first one is the noise which arises when calculating for the distance to the nearest obstacle, which describes the probability of getting a correct measurement $\mathrm{p_{hit}}$ and it is modeled by a Gaussian with zero-centered. Besides we assumed another noise source as a result of readings with a max range which have a significant likelihood, PMD distribution used to model this type of uncertainty $\mathrm{p_{max}}$. Moreover, random measurements increase uncertainty in sensor readings, so the random noise modeled by the uniform distribution $\mathrm{p_{rand}}$. These distributions should be integrated to obtain the desired likelihood $p\left(z_t^b \mid x_t^{[n]},\ m\right)$ [25].

Table 2.3. Likelihood field range finder model

```
1:  **Likelihood_Field_Range_Finder_Model_Algorithm** $(z_t,\ x_t, m)$
2:  $LH = 1$
3:  for all $b$ do
4:      if $z_t^b \neq z_{max}$
5:          $x_{z_t^b} = x + x_{sens}\, cos\,\theta - y_{sens}\, sin\,\theta + z_t^b\, cos(\theta + \theta_s + \theta_{ep})$
6:          $y_{z_t^b} = y + x_{sens}\, sin\,\theta + y_{sens}\, cos\,\theta + z_t^b\, sin(\theta + \theta_s + \theta_{ep})$
7:          $dist = min\left\{\sqrt{\left(x_{z_t^b} - \acute{x}\right)^2 + \left(y_{z_t^b} - \acute{y}\right)^2}\ \middle|\ \langle \acute{x}, \acute{y} \rangle \text{occupied in } m\right\}$
8:          $LH = LH * \left[w_{hit}\ .\ prob(dist, \sigma_{hit}) + \frac{w_{rand}}{z_{max}}\right]$
9:  return $LH$
```

The algorithm of the likelihood field model used for computing the measurement probability by utilizing the distance transform is summarized in Table 2.3 [25]. $prob(dist, \sigma_{hit})$ is the distance transform probability computed by Gaussian distribution under a zero center with a measurement noise σ_{hit}. w_{hit} denotes the expected measurement weight, and w_{rand} denotes the random measurement weight. Line 7 calculates for the minimum distance between the beam endpoint and the nearest obstacle, where $\langle \acute{x}, \acute{y} \rangle$ indicates the occupied cell coordinates around the endpoint.

Since the computations of the Euclidean distance transform is smooth, the likelihood model is smoother than any other models used to achieve the measurement probability. Moreover, the precomputations of the Euclidean distance done in a 2D table rather than 3D tables. However, there are three shortcomings to this model. First, the likelihood field model does not model dynamic objects that might cause a drop in probabilities. Second, it does not see obstacles that encounter the beam when mapping to the samples as it looks just for the endpoints. Third, the likelihood field model does not model the uncertainty in the map itself.

2.3. Pose Estimation

According to the Odometry motion model, previous samples move to new states on the state space as a result of the motion control, which is applied to the last sample set. After that, the likelihood field range finder model assigns the weight of each new sample generated. The updated weights describe the probability of a sample to match the actual robot's pose on the map. Now, based on the Bayes rule [42] shown in the Equation (2.26) the estimated pose can be found. Bayes rule provides an efficient technique to estimate the posterior $p(x|z)$ by utilizing the inverse conditional probability $p(z|x)$ and the prior probability $p(x)$, where $p(z)$ is the evidence.

$$P(x \mid z) = \frac{P(\mathrm{z} \mid x)\, P(x)}{P(\mathrm{z})} \tag{2.26}$$

Based on the Bayes rule, Equations (2.27) and (2.28) used to compute the estimated coordinates (x_{est}, y_{est}) of the actual pose.

$$x_{est} = \frac{\sum_{n=1}^{n} \omega_t^n x_t^n}{\sum_{n=1}^{n} \omega_t^n} \tag{2.27}$$

$$y_{est} = \frac{\sum_{n=1}^{n} \omega_t^n y_t^n}{\sum_{n=1}^{n} \omega_t^n} \tag{2.28}$$

Equations (2.29) to (2.31) utilized to estimate the mobile robot’s heading direction relative to the x-axis of the world frame.

$$\cos\theta_{est} = \frac{\sum_{n=1}^{n} \omega_t^n \cos\theta_t^n}{\sum_{n=1}^{n} \omega_t^n} \tag{2.29}$$

$$\sin\theta_{est} = \frac{\sum_{n=1}^{n} \omega_t^n \sin\theta_t^n}{\sum_{n=1}^{n} \omega_t^n} \tag{2.30}$$

$$\theta_{est} = \text{atan2}(\sin\theta_{est},\ \cos\theta_{est}) \tag{2.31}$$

2.4. **Resampling**

The resampling phase in MCL is essential since all practical applications use a limited number of samples. During the sampling phase, every sample makes an error, and these errors will accumulate over time, leading to a divergence in the filter. As a result, the estimator will not be able to track the actual state over long periods. Resampling methodology was utilized to prevent the degeneration of the samples [43,44] through sampling a new equally weighted sample set (corrected sample set) from the degenerate samples (predicted sample set). Here, each sample will survive based on its importance weight obtained from the measurement model. Generally, the resampling process will prompt low weighted samples to inherit the pose of high weighted samples. Therefore, the probability of losing the filter to track the pose will be reduced. Moreover, since all samples will be redistributed over regions with a high likelihood, the computational effort of the MCL algorithm will also focus on the areas with high probability. However, some low-weight samples are useful in handling disturbances. Thus, the resampling algorithm should preserve low-weight samples [45].

2.4.1. **Multinomial resample**

Multinomial resampling methodology utilizes a single random number to resample from the predicted sample set $\bar{X}_t$. However, the probability of a sample being regenerated is proportional to its importance weight. This resampling process is accomplished by generating a uniform random number in the period [0,1].

Table 2.4. Multinomial resampling algorithm

```
1:   Multinomial_resampler_algorithm ($\bar{X}_t,\ W_t$)

2:   $X_t = \emptyset$
3:   $c = w_t^{[1]}$
4:   $i = 1$
5:   for $n = 1$ to $N$
6:       $r = \mathrm{rand}(0,1)$
7:       while $r > c$
8:           $i = i + 1$
9:           $c = c + w_t^{[i]}$
10:      endwhile
11:      add $x_t^{[i]}$ to $X_t$
12:  endfor
13:  return $X_t$
```

The algorithm shown in Table 2.4 describes the multinomial resampling process, where the predicted sample set $\bar{X}_t$ is transformed into the corrected sample set X_t with the same size. Normalized weights are passed to the algorithm as input. The variable c in line 3 represents the cumulative sum of the normalized weights; line 4 stands for the index variable of the selected state, r is a uniform random number. Usually, the returned corrected sample set X_t holds many duplicated samples, since samples are inferred by replacement.

The loop in line 7 performs two tasks; it checks if the cumulative weight at the index i exceeds the random number r, and it adds the weight of sample i to the cumulative weights. This while loop creates complexity in the algorithm. Processing time is essential when handling the localization problem. Thus, the binary search method [46] can be used to perform the search for a sample index that its corresponding sample exceeds the generated random number.

In summary, the weight of the samples was initialized by one and then corrected multiplicatively under the observation model. After two update iterations, the resampling phase takes place, and again the weights of all samples sum up to one.

2.4.2. **Augmented MCL**

The previous resampling process keeps the samples to track the robot's pose by prompting low weight samples to inherit the state of high weight samples. Unfortunately, this method cannot survive samples in global localization failures, or when the robot kidnapped, since over time, all samples will converge to a specific area in the robot's space and no alternative poses can be envisaged by the algorithm. Moreover, since the MCL algorithm is stochastic, during the resampling phase, it may unintentionally ignore all samples around the actual robot's pose. This problem can be solved by adding random samples to the sample set to represent random poses over the state space. Even if the probabilities still high, random samples provide an extra level of robustness.

At each iteration, one may introduce a certain number of random samples. However, too many random samples will weaken localization capability, and also a few random samples will have no significant impact on the algorithm's efficiency. The probabilistic approach suggests adding samples based on the performance of localization through monitoring measurement probability [25]. One thought is by checking for the average weight of the samples. However, other reasons may cause dropping in measurement probability, such as a high LiDAR noise, or the samples may still be scattered all over the state space during a global localization step. Therefore, the trick to compute the number of needed random samples is by observing the fast average weight and the slow average weight of the measurement probability. The desired measurement probability, long-term average, and short-term average are calculated in Equation (2.32), Equation (2.33), and Equation (2.34), respectively.

$$w_{avg} = \frac{1}{N}\sum_{n=1}^{N} w_t^{[n]} \tag{2.32}$$

$$w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow}) \tag{2.33}$$

$$w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast}) \tag{2.34}$$

w_{avg} stands for the empirical measurement likelihood, w_{slow}, and w_{fast} refer to the exponential filters of the importance weight over a reasonably long and short time, respectively. The parameters α_{slow}, and α_{fast} are the decay rates of these filters, which infer the slow weight and fast weight averages. This technique discussed above is known as augmented MCL, and its illustrated in Table 2.5 [25] below. augmented MCL works well if $0 \leq \alpha_{slow} \ll \alpha_{fast}$ where a sudden descent in short-term average's value compared to long term average's value, signposts a decrease in localization efficiency.

Table 2.5. Augmented MCL algorithm

```
1:   **Augmented_MCL_Algorithm** $(X_{t-1}, u_t, z_t, m)$
2:       Static $w_{slow}, w_{slow}$
3:       $w_{avg} = \bar{X}_t = X_t = \emptyset$
4:       for $n = 1$ to $N$ do
5:           $x_t^{[n]} =$ **sample_motion_model** $(x_{t-1}^{[n]}, u_t)$
6:           $w_t^{[n]} =$ **measurement_model** $(z_t, x_t^{[n]}, m)$
7:           $\bar{X}_t = \bar{X}_t + \langle x_t^{[n]}, \omega_t^{[n]} \rangle$
8:           $w_{avg} = w_{avg} + \frac{1}{N} w_t^{[n]}$
9:       endfor
10:      $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
11:      $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
12:      for $n = 1$ to $N$ do
13:          With probability $\max\{0.0,\ 1.0 - w_{fast}/w_{slow}\}$ do
14:              Add random pose to $X_t$
15:          else
16:              draw $i$ with probability $\propto\ w_t^{[i]}$
17:              add $x_t^{[i]}$ to $X_t$
18:          endwith
19:      endfor
20:      return $X_t$
```

The key idea of augmented MCL found in line 13, where the random samples are added to the sample set based on the ratio w_{fast}/w_{slow}. However, if the short-term average is better than the long-term average, the algorithm will keep tracking of samples with high measurement likelihood. The random sample added in line 14 is generated according to the uniform distribution on the robot's state space.

3. PROPOSED SCHEMES

3.1. Motivation

We proposed two novel methodologies that can significantly reduce uncertainty in the global indoor localization problem; both of these methodologies are an extension to the standard MCL. The first improvement introduces an optimized scheme at the initialization step that detects mapped regions with high probabilities only based on the initial scan data. Given the robot's environment map and only the initial scan data, the proposed algorithm detects regions with high likelihood based on the observation model to distribute samples there. As a result, the suggested sample distribution will expedite the process of localization. And it will significantly reduce the amount of time it takes for the robot to find its actual pose when it starts moving. The second improved scheme presents an effective resampling strategy to deal with the kidnapped robot problem that enables the robot to recover quickly when the sample weights drop-down due to unmapped dynamic obstacles within the sensor's field of view. While the conventional resampling method propagates random samples around the entire working space, the proposed scheme distributes the random samples within a circular region centered around the robot's pose by taking into account the prior knowledge about the most recent successful pose estimation. Since the samples are distributed over the region with high probabilities, it will take less time for the mobile robot to infer its accurate pose.

3.2. Improved Global Localization Algorithm

The initial belief $bel(x_0)$, as discussed in the Markov Localization in section 1.3 can be initialized in three forms based on the localization problem we maintain, and it represents the initial knowledge about the actual pose of the mobile robot. The standard MCL algorithm solves the global localization problem by introducing the uniform distribution over all possible legal states in the pose configuration space to represent the initial belief. However, in our proposed scheme, we can treat the global localization problem as a partial problem regarding the robot's pose. Based only on the initial

LiDAR scan z_1 collected in the immobile state at the initialization phase, the mobile robot gets knowledge about the features of its current place. Still, it cannot specify its pose precisely because of the uncertainty in the OGM map itself, symmetrical areas in the robot's environment, and unnatural sensor noise. In this case, the initial belief can be represented by multi distributions (multi-modal belief). The uniform distribution is used to describe the belief in the expected areas while the density will be zero anywhere else.

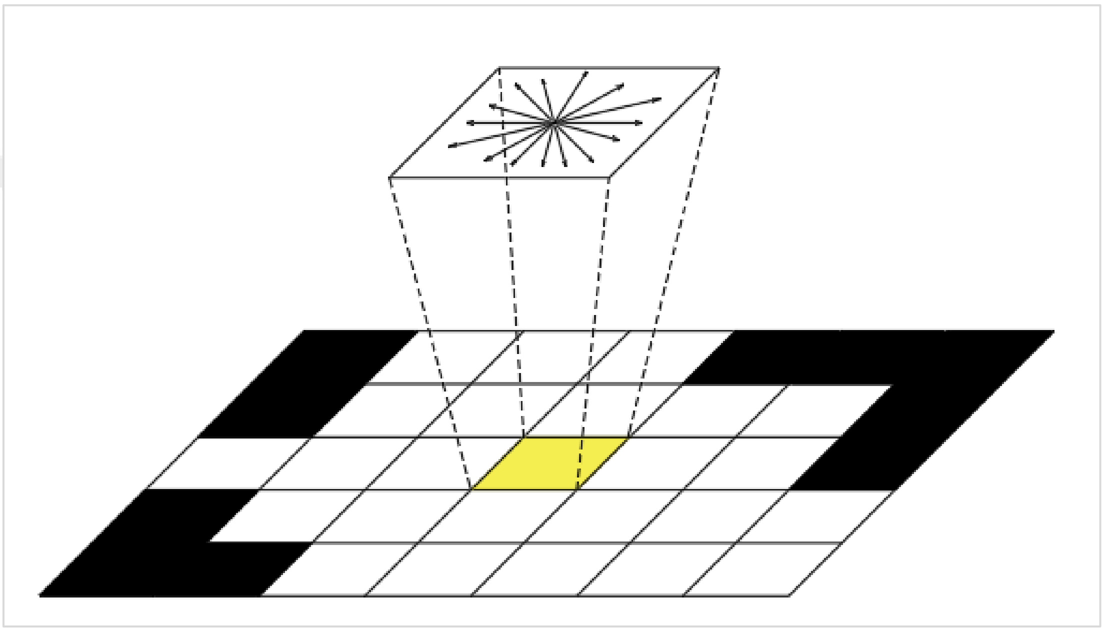


Figure 3.1. OGM with all possible poses over the pose configuration space. Black cells represent occupied areas, while the white cells represent free areas

Each free grid cell in the OGM depicts the robot's position coordinate (x, y) with different orientations θ, where $(0 \leq \theta < 2\pi)$ as illustrated in Figure 3.1 above, which means different hypothesis states in one grid cell. Given a map of the robot's environment m and the initial perceptual data z_1, the proposed algorithm searches for the poses with high likelihood within state space according to the observation model $p(z_t \mid x,\ m)$ discussed in the section 2.2.1 above and illustrated again in Equation (3.1) below.

$$p(z_1 \mid x_1^{[s]}, m) = \prod_{b=1}^{b} \left(w_{\text{hit}} . \exp(-\frac{{d_b}^2}{2\sigma^2}) + w_{rand} \frac{1}{z_{\max}} \right) \tag{3.1}$$

The measurement probability calculations iterate over all possible hypothesis states $x^{[s]}$. However, to reduce the computational effort, instead of searching the robot's configuration space over all grid cells, we skipped some cells on the map (i.e., in our case we skipped every 3 cells) resulting in a 15cm search resolution in x and y directions. Equivalently, 3 degrees resolution is adapted for the angle search space in each grid cell. Consequently, for a map of size 15m x 15m, 101x101x120 different pose values are evaluated. During this process, the robot should be in an immobile state. And as a result of these calculations, the proposed method will nominate only the first N high probability states to represent the initial belief as shown in Equation (3.2), where N indicates the number of samples employed to track the actual pose of the mobile robot.

$$\overrightarrow{poses} = \overrightarrow{poses}\left[\max\left(p(z_1 \mid x_1^{[1\ldots s]}, m)\right)\Big|_1^N\right] \tag{3.2}$$

In this case, the generated samples can be drawn directly into the high likelihood regions within a map. Moreover, the suggested initial distribution as samples concentrated on a small portion of space with a high likelihood will reduce uncertainty at start-up.

Now, once the mobile robot starts its motion, the MCL filter incorporates the proposed initial posterior with the motion model $p(x_t|x_{t-1}, u_t)$. As a result, the odometry motion model discussed before in the section 2.1.1 will focus on the entire continuum of states concentrated around the expected robot’s pose. After only a few resampling steps, all the particles are expected to converge to the actual robot’s state.

3.3. **Effective Resampling Strategy for Augmented MCL**

The probability of sensor data is measured given a particle’s pose and a known map $p(z_t \mid x_t^{[n]},\ m)$. As discussed in section 2.2.1, the endpoints of sensor scans are projected into the global coordinate space of the given map. A likelihood value is computed for each ray in the scan, which is inversely proportional to its end point’s distance to nearest mapped obstacle point, as illustrated in Equation (2.17). Assuming independence, these probabilities are then multiplied to obtain the measurement probability for the scan as presented in Equation (2.18). When the mobile robot encounters unmapped dynamic obstacles, the probabilities of individual rays that hit the dynamic obstacle will drop based on the distance between the dynamic obstacle and the closest mapped obstacles. The higher this distance, the lower the probabilities will be. When the probabilities drop, the robot will think that it is getting lost (although it is not) and random samples will be added to the sample set based on the divergence between fast and slow average weights to estimate the correct distribution accurately. However, the conventional resampling method draws these random samples over the pose configuration space without taking into account any prior knowledge about the last confident robot's state. So, this might result in total randomization of the samples over the entire map region turning this temporary case into the global localization problem. Global localization problem requires thousands of samples, especially with large maps which increases the computational burden negatively affecting real-time performance.

The proposed resampling strategy deals with the kidnapped robot problem that enables the robot to recover quickly when the sample weights drop-down due to unmapped dynamic obstacles within the sensor's field of view. While the random samples are added to the sample set based on the divergence between fast and slow average weights, the proposed algorithm infers their poses in accordance with the uniform normal distribution over a specified area rather than drawing them over all possible poses on the map. Our proposed algorithm takes place when the filter diverges after it was in convergence mode. In this case, the mobile robot knows its last confident pose and can predict where it can reach as time progresses based on its maximum velocity, as shown in Equation (3.3) below. R represents the predicted traveled distance by the robot from the last confident pose, v_{max} denotes the maximum velocity of the robot, and $t_{elapsed}$ is the elapsed time since the last time the robot was confident about its pose estimate.

$$R = v_{\max} \, t_{elapsed} \tag{3.3}$$

Thus, random samples will be confined to a circular region centered around the last filter convergence position by taking into account the prior knowledge about the most recent successful pose estimation, as shown in Figure 3.2 below. Since the random samples are distributed over the region with high probabilities, it will take less time for the mobile robot to infer its accurate pose.

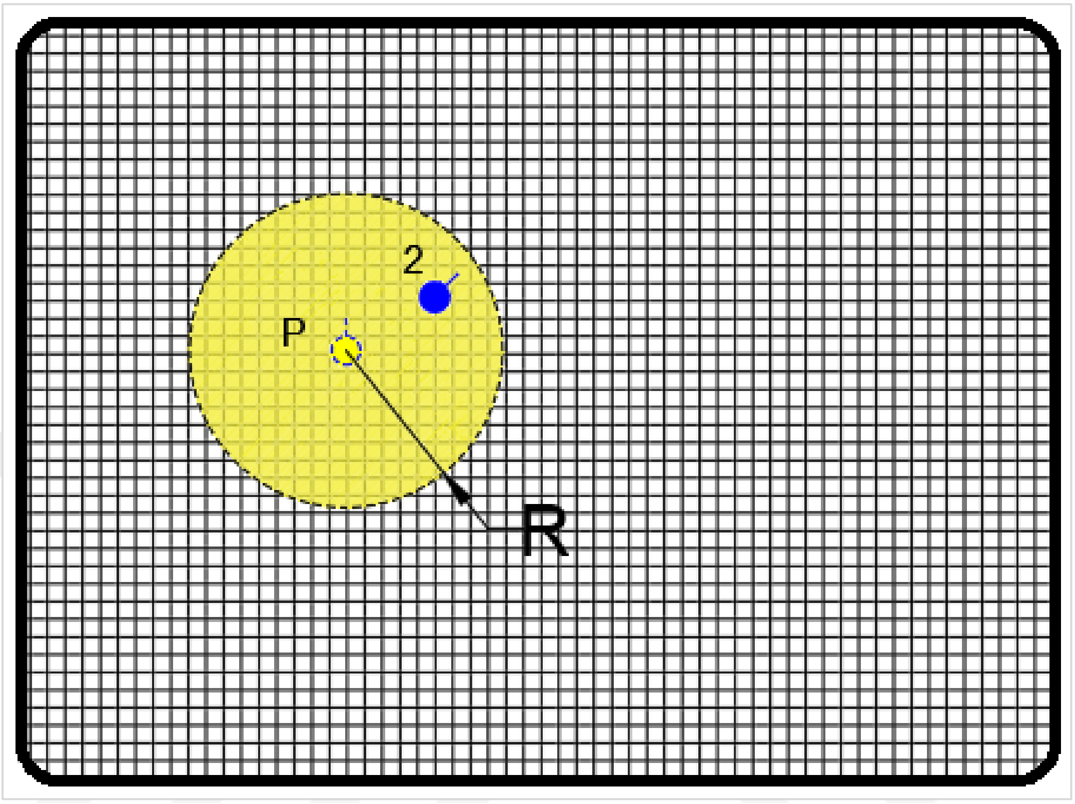


Figure 3.2. The specified area around the last confident position (P) of two-dimensional coordinates where random samples can be added. The second robot's pose represents the unknown new pose after the filter gets diverged

Table 3.1. Effective resampling technique that finds the desired certain region around the last robot's confident pose

```
1:   **Desired_Specified_Area_Algorithm** $(m, P, v_{max}, t_{lastConv})$

2:   $A = \emptyset$
3:   $t_{elapsed} = t_{real} - t_{lastConv}$
4:   $R = v_{max} * t_{elapsed}$
5:   for $x = 1$ to length$(m(x))$
6:       for $y = 1$ to length$(S(y))$
7:           if $\{[m(x) - P(x)]^2 + [m(y) - P(y)]^2\} \leq R^2$
8:               add $\langle x, y \rangle$ to $A$
9:           endif
10:      endfor
11:  endfor
12:  return $A$
```

Table 3.1 shown above describes the proposed resampling algorithm utilized to determine the exact certain area within the pose configuration space (m) where the robot might be. The Radius (R) of the desired region where the random samples will be added depends on the elapsed time from the last filter convergence position to the current robot's position (line 4), where t_{real} is the current real-time, and $t_{lastConv}$ is the real-time at the last filter convergence position. The desired certain area increases as time increase. The algorithm checks all possible positions from the pose space to judge if the position will be included in the desired region (line 7). In line 8, the algorithm adds the correct position to the specified area (A) which contains a set of two-dimensional poses.

The proposed approach also takes into account the measurement probability to adjust the desired area. The desired region gradually increases as the probabilities continue to the dropdown. If the localization performance keeps down, the samples will be totally randomized over the pose configuration space. Finally, the problem turns again to be a global localization problem. Through this approach, fewer samples will be enough to re-infer the true belief of the mobile robot's state. Therefore, the real-time performance of the algorithm will increase in terms of decreasing the computational burden. Besides, since the samples are distributed over a specified region with high probabilities, the samples will take less time to converge to the actual robot's pose.

4. SIMULATION SET-UP AND RESULTS

This chapter demonstrates the simulation results we conducted for all of the algorithms previously explained in chapter 2 utilized to achieve the augmented MCL filter and outlines the simulation results and some analysis for evaluating the efficacy of our proposed methodologies, which are described in chapter 3. Besides, we presented a fair comparison between our proposed schemes and the conventional approaches for augmented MCL when applied to solve robot positioning problems in terms of sample set size, traveled distance, and time required for the robot to localize itself in its world.

The conducted experiments were performed in a simulated robot environment. MATLAB software is widely used for scientific research approaches and utilized in this thesis to design, analyze, and evaluate the proposed schemes. Besides, we wrote and tested the presented standard augmented MCL algorithm in MATLAB software. All tests were examined on a Laptop with 2.53 GHz (4 CPUs).

Section 4.1 presents the implementation details to perform pose estimation in a simulated environment based on the augmented MCL method. We discussed how we generate our robot's environments (OGM maps). Moreover, the Bresenham line algorithm [47] that is used to simulate ray tracing, which provides range data for the sensor model has been explained. Also, we have made clear the method we utilized to make the results more authentic. Section 4.2 provides the results and discussions for the models of standard augmented MCL, such as the odometry motion model and the likelihood measurement model. Besides, we discussed the term convergence to judge the localization performance of the mobile robot. Also, we discussed the results of the resampling phase in MCL. In section 4.3, we presented the results of our proposed schemes compared to the conventional methods of augmented MCL. Also, the results obtained will be discussed in parallel.

4.1. **Implementation Details**

The goal of the augmented MCL algorithm presented in Table 2.5 is to localize the differential drive mobile robot outfitted with incremental encoders and LiDAR sensor in its pre-mapped world. In this section, we will discuss the essential simulation parts needed to implement our proposed algorithms.

4.1.1. **Occupancy grid map**

2D binary OGM map utilized to describe and visualize the mobile robot's environment, as well as obstacles. We simulated different indoor OGM maps of different sizes. First, we drew the indoor environments using 2D AutoCAD software and then converted them to 2D binary OGM using MATLAB software. Figure 4.1 depicts our prepared OGM environments. Different maps of different sizes enable us to evaluate the performance of our proposed approaches.

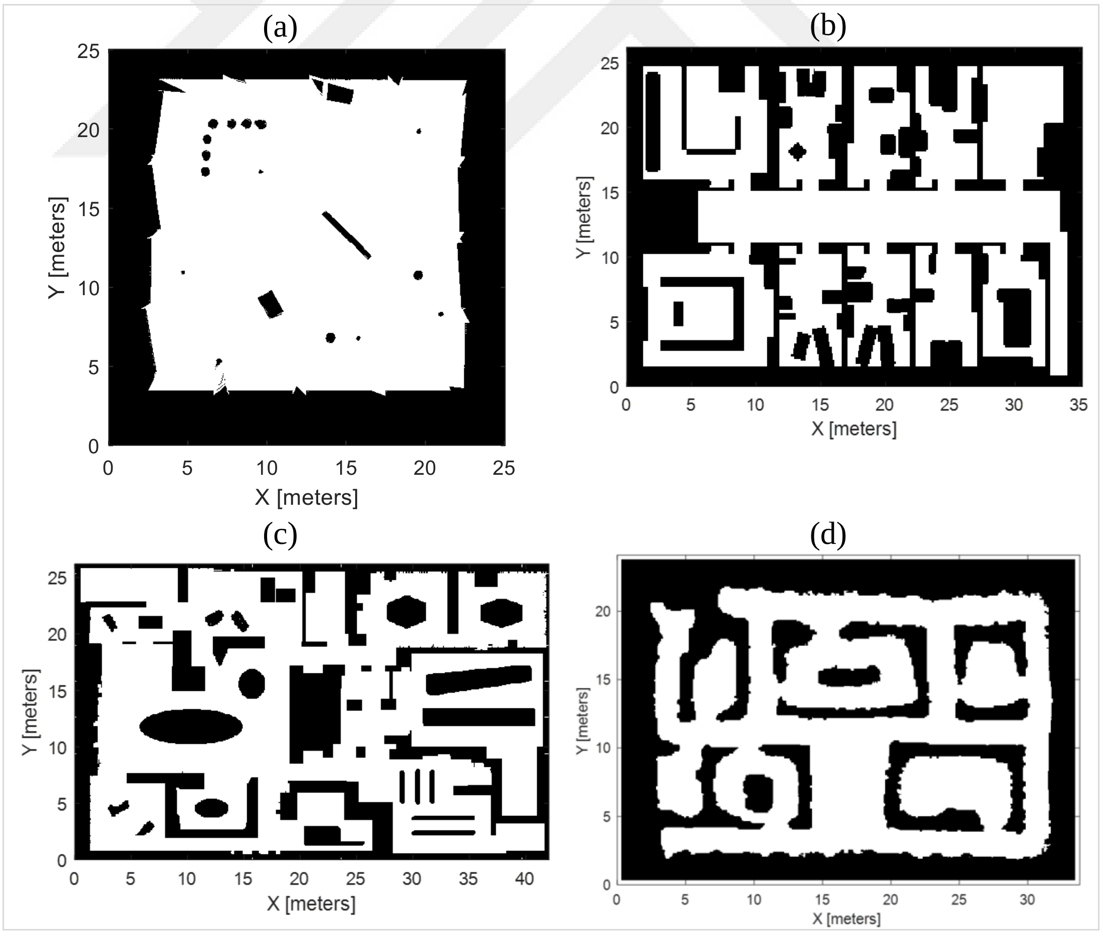


Figure 4.1. OGM indoor environments. The resolution for each is 5cm per cell. (a) simple map, (b) symmetric map, (c) factory map, (d) maze map

All the prepared OGM maps come with 20 cells per meter resolution. The simple map is a small and simple environment, while the symmetric environment contains a hallway with somewhat symmetric places in the right and left sides. Figure 4.1 (c) and (d) represent a factory and maze maps, respectively. The mobile robot can be randomly placed in any possible pose over the state space, and then it is tele-operated within the free spaces. Since we have clean binary maps, we will check for the free and occupied locations over the OGM map based on the cell value as shown in the Formula (4.1) below, where the cell's occupancy status is determined.

$$\text{cell\_value} = \begin{cases} 1, & \text{occupied} \\ 0, & \text{free} \end{cases} \tag{4.1}$$

4.1.2. LiDAR data simulation

LiDAR is a distance sensor that provides the distance information from the target to the obstacle within the maximum sensor range. Simulated LiDAR data employed to detect obstacles in the robot's workspace, and it collects data ranges at given angle intervals. The Aperture angle of the laser sensor is 270° with 1° angle resolution, so the maximum number of rays we have is 271 beams. The Bresenham line-drawing algorithm [47] used to visualize the laser rays, and it provides accurate LiDAR data for the measurement model. This algorithm draws a straight line connecting two grid cells (the one occupied by the LiDAR sensor, while the other cell occupied by a scan endpoint) on an OGM map by determining all 2D coordinates of the discrete cells that should be chosen to build a reasonable approximation to the laser ray.

Other algorithms, such as the Digital Difference Analyzer (DDA) algorithm used to draw a straight line in a picture grid or display [48]. However, DDA algorithm uses float values in its calculations, and to be able to represent the integer number of the pixel, it rounds off the float number which ends with alias line. Besides, the processing time for float values is more than the time taken for integers. Another professional algorithm is Wu's algorithm that addresses the anti-aliasing technique and is used today in computer graphics [49]. However, Wu's algorithm still slower than the Bresenham algorithm. The simplicity and speed of the Bresenham algorithm demonstrate its importance. The error in the Bresenham algorithm is incremental.

The following formulas are used to draw any straight line.

$$y = mx + c \tag{4.2}$$

$$y - y_1 = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) \tag{4.3}$$

$$\frac{x}{a} + \frac{y}{b} = 1 \tag{4.4}$$

In Equation (4.2), c is a constant number that holds the value of y when the line intersects the y-axis, m is the gradient or the slope of the line. In Equation (4.4), a and b represent the values when the line intersects x-axis and y-axis, respectively.

In computer graphics, the vector Equation (4.2) is used in the rasterization process. The conversion process of a line from vector form $y = mx + c$ to pixels (x, y) is called rasterizing [50] as illustrated in Figure 4.2 below.

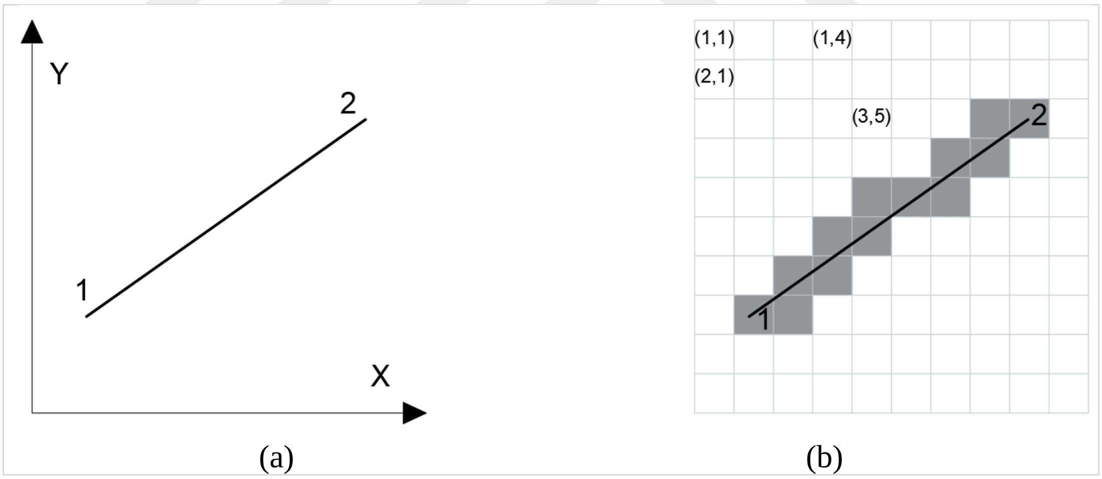


Figure 4.2. Line Representation. (a) vector representation in world frame, (b) raster representation in grid

The goal of the Bresenham algorithm is to find the pixels in the grid that gives a good approximation for the vector line. If the slope is one, then it’s simple to select all diagonal pixels to represent our line, where the value of x equals the value of y. However, sometimes the ray has a positive slope ($m > 1$), and at other times a negative slope ($m < 1$). To imagine this problem, suppose there is a gap between pixels, and the line passes through that gap whether the slope is positive or negative as depicted in Figure 4.3 below. Our suggestion take place in the global frame, after that we can map points into the grid.

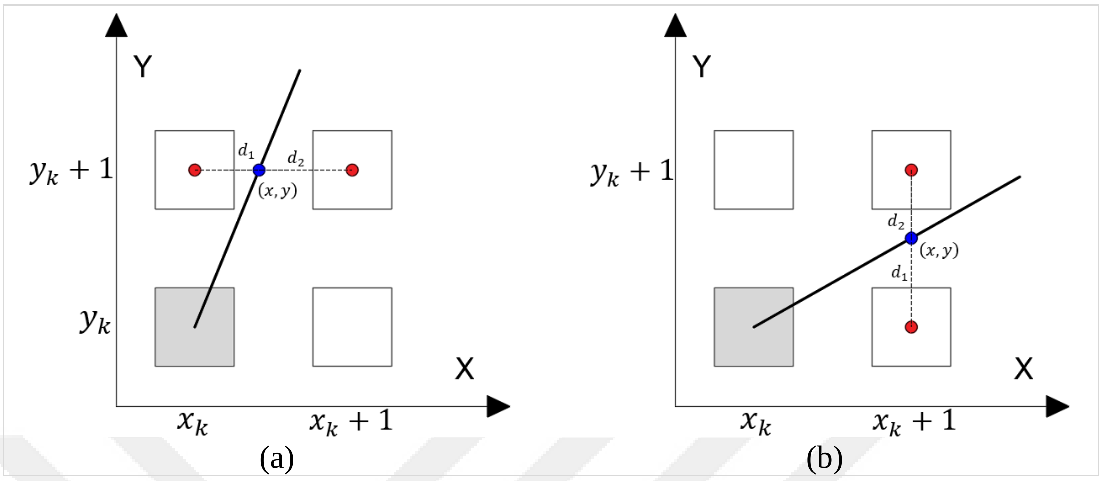


Figure 4.3. (a) Positive slope, (b) negative slope

Figure 4.3 (a) illustrates the problem when the line has a positive slope. In this case, the next point in the y-axis will be incremented, while in the *x*-axis, we do not know whether we should increment the value of *x* or keep the previous value in order to represent a smooth line. In analogy with the positive slope, if the line has a negative slope as shown in Figure 4.3 (b), the next point in the *x*-axis will be incremented. However, in order to get a smooth line should we select the next pixel in the *y*-axis or keep the previous value for *y*.

So, in the case of a positive slope, we should sample *y* value:

$$(m > 1) \Rightarrow \begin{cases} y_{next} = & y_k + 1 \\ x_{next} = & \begin{cases} x_k \\ x_k + 1 \end{cases} \end{cases} \tag{4.5}$$

Contrariwise, in the case of negative slope, we should sample *x* value:

$$(m < 1) \Rightarrow \begin{cases} x_{next} = & x_k + 1 \\ y_{next} = & \begin{cases} y_k \\ y_k + 1 \end{cases} \end{cases} \tag{4.6}$$

The decision in these cases will depend on the distance between the next pixel and the nearest point (x, y) on the line. First, let us take the case of a negative slope described in Formula (4.6). Based on the difference between the distances we can decide which pixel should be selected as shown in the next formula.

$$y_{next} = \begin{cases} y_k, & d_1 - d_2 < 0 \\ y_k + 1, & d_1 - d_2 \geq 0 \end{cases} \tag{4.7}$$

Equation (4.8) and (4.9) find the value of each distance.

$$\begin{aligned} d_1 &= y - y_k \\ \cdots &= m(x_k + 1) + c - y_k \end{aligned} \tag{4.8}$$

$$\begin{aligned} d_2 &= (y_k + 1) - y \\ \cdots &= (y_k + 1) - m(x_k + 1) - c \end{aligned} \tag{4.9}$$

Now, Equation (4.10) calculates for the error ($d_1 - d_2$).

$$\begin{aligned} d_1 - d_2 &= \left[m(x_k + 1) + c - y_k\right] - \left[(y_k + 1) - m(x_k + 1) - c_k\right] \\ \cdots \quad &= 2m(x_k + 1) - 2y_k + 2c - 1 \end{aligned} \tag{4.10}$$

Since the slope ($m = {}^{\Delta y}/_{\Delta x}$), multiply the left and right terms by Δx to avoid float numbers.

$$\begin{aligned} \Delta x\left(d_1 - d_2\right) &= 2\Delta y\,(x_k + 1) - 2y_k + 2c - 1 \\ \cdots \quad &= \left[2\Delta y x_k - 2\Delta x y_k\right] + \left[2\Delta y + 2c\Delta x - \Delta x\right] \end{aligned} \tag{4.11}$$

$\Delta x(d_1 - d_2)$ is called the decision parameter and denoted by the symbol P_k. The second term on the right side is constant and can be normalized as we calculate for all pixels.

$$P_k = \left[2\Delta y x_k - 2\Delta x y_k\right] \tag{4.12}$$

However, every time we increase the *x* value, we should make a decision. The decision for the next pixel is shown in the following Equation (4.13).

$$P_{next} = \left[2\Delta y x_{next} - 2\Delta x y_{next}\right] \tag{4.13}$$

Now, subtract Equation (4.13) from (4.12) to see how much decision variable changes every time.

$$\begin{aligned} P_{next} - P_k &= [2\Delta y x_{next} - 2\Delta x y_{next}] - [2\Delta y x_k - 2\Delta x y_k] \\ \cdots \quad &= 2\Delta y\,(x_{next} - x_k) - 2\Delta x\,(y_{next} - y_k) \end{aligned} \tag{4.14}$$

Here, based on the value of the current decision parameter $P_k = \Delta x(d_1 - d_2)$ there are two scenarios for the next decision:

- if ($P_k < 0$), then no change in decision ($y_{next} = y_k$), and:

$$\begin{aligned} P_{next} &= P_k + 2\Delta y\,(x_k + 1 - x_k) - 2\Delta x\,(y_k - y_k) \\ \cdots \quad &= P_k + 2\Delta y \end{aligned} \tag{4.15}$$

- if ($P_k \geq 0$), then there is a change in the decision ($y_{next} = y_k + 1$), and:

$$\begin{aligned} P_{next} &= P_k + 2\Delta y\,(x_k + 1 - x_k) - 2\Delta x\,(y_k + 1 - y_k) \\ \cdots \quad &= P_k + 2\Delta y - 2\Delta x \end{aligned} \tag{4.16}$$

As we note from the Equation (4.15) and (4.16), the next decision relies on the previous decision. So, in order to compute the initial value for the decision parameter go back to Equation (4.11) and substitute the value of parameter c from Equation (4.2). The initial value for the decision parameter shown in the Equation (4.17) below.

$$\begin{aligned} P_k &= [2\Delta y x_k - 2\Delta x y_k] + [2\Delta y + 2\Delta x y_k - 2\Delta y x_k - \Delta x] \\ \cdots &= 2\Delta y - \Delta x \end{aligned} \tag{4.17}$$

The second case is when the slope is positive. Here, we sample *y* value. If we follow the same procedure as for the negative slope, then we can get the same two scenarios for the next decision parameter based on the value of the current decision parameter $P_k = \Delta x(d_1 - d_2)$:

- if ($P_k < 0$), then no change in decision ($x_{next} = x_k$), and

$$P_{next} = P_k + 2\Delta x \tag{4.18}$$

- if ($P_k \geq 0$), then there is a change in the decision ($x_{next} = x_k + 1$), and

$$P_{next} = P_k + 2\Delta x - 2\Delta y \qquad (4.19)$$

The basic ray tracing algorithm that summarizes all the above equations is mentioned in Appendix A. Bresenham algorithm is effective concerning execution speed and memory use.

The following Table 4.1 below describes the LiDAR model which utilized to simulate the rays. Map, actual current robot's pose, sensor's pose relative to the mobile robot, maximum sensor range, and all ray angles provided to the algorithm as inputs. The algorithm returns the 2D coordinates of the endpoints of the rays that hit obstacles or return the 2D coordinates of the endpoints of the maximum range in the absence of obstacles.

Table 4.1. LiDAR data simulation algorithm

```
1:   **LiDAR_Model_Algorithm** $(m, x_{t,actual}, x_{sensor}, z_{max}, \theta_{ep})$
2:      $X_{ep} = Y_{ep} = \emptyset$
3:      $x_s = x_t + x_{sens}\cos\theta - y_{sens}\sin\theta$
4:      $y_s = y_t + x_{sens}\sin\theta + y_{sens}\cos\theta$
5:      for all $b$ do
6:         $x_{ep}^b = x_s + z_{max}\cos(\theta + \theta_{sens} + \theta_{ep})$
7:         $y_{ep}^b = y_s + z_{max}\sin(\theta + \theta_{sens} + \theta_{ep})$
8:         $(X, Y)$ = **Ray_Tracing_Algorithm** $(x_s, y_s, x_{ep}^b, y_{ep}^b)$
9:         for all $X$
10:           If $m(y, x) = 1$
11:              add endpoint $(x, y)$ to $(X_{ep}, Y_{ep})$
12:              break
13:           endif
14:        endfor
15:     return $[X_{ep}, Y_{ep}]$
```

4.1.3. **Robot trajectory**

Our differential mobile robot can be randomly placed in any possible pose over the working space. Then the user can tele-operate the robot within the free space over the map. Continuously the augmented MCL algorithm tries to localize the mobile robot in its environment map. However, to make a fair comparison between the conventional methods of augmented MCL and our proposed schemes, we enabled the robot to follow the same pre-defined path under the same environment features. First, over the same map, we created a standard robot trajectory by controlling the robot remotely, and then we performed our tests using these trajectories, such as one shown in the following Figure 4.4 below.

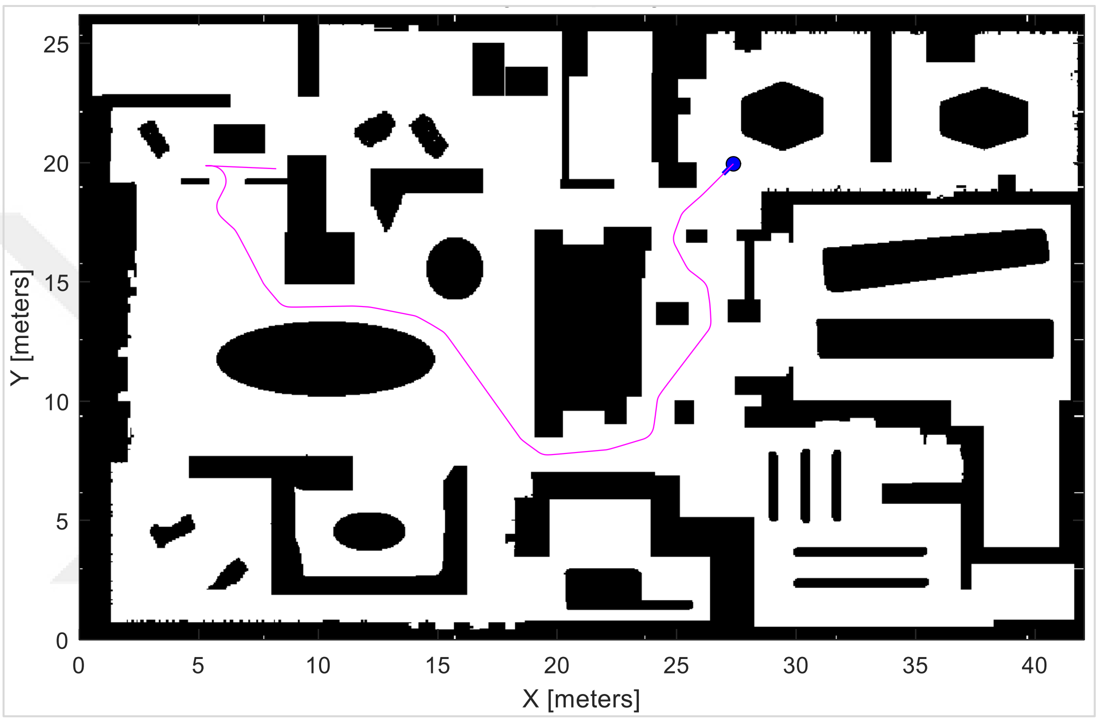


Figure 4.4. Predefined trajectory for the comparison purpose over the factory map.

According to the Robot forward kinematics model discussed in section 2.1.2 the robot is driven by independent rotational and translational velocities controlled by the user. The output of the forward kinematics is the 2D coordinates of the robot’s position and heading direction. In the first time, the user will drive the robot by controlling its rotational and translational velocities, our algorithm will save these inputs continuously. Later in the comparison step, the algorithm will be allowed to use the memorized values of velocities to control the movement of the robot.

4.2. **MCL phases**

This section provides the results and discussions of the Odometry motion model and the Likelihood field range finder model that explained above in section 2.1.1 and 2.2.1, respectively. Besides, we discuss the term convergence to judge the localization performance of the mobile robot. Finally, we will provide results for the resampling technique, and these results will be discussed.

4.2.1. Odometry motion model

Map-free odometry motion model illustrated in Table 2.2 is utilized to calculate the motion of the mobile robot over time. Odometry motion model generates random samples from the state transition probability $p(x_t \mid x_{t-1}, u_t)$. We represented odometry information by a sequence of three free noise parameters, initial rotation δ_{rot1} followed by translation δ_{trans}, and finally, the second rotation δ_{rot2}. Formula (2.3) demonstrates the noise modeling for the odometry information, where the measured motion is given by the actual motion corrupted with noise. However, noise samples generated from Gaussian normal distribution with zero mean and variance σ^2 by using the closed-form principle given in Equation (1.13).

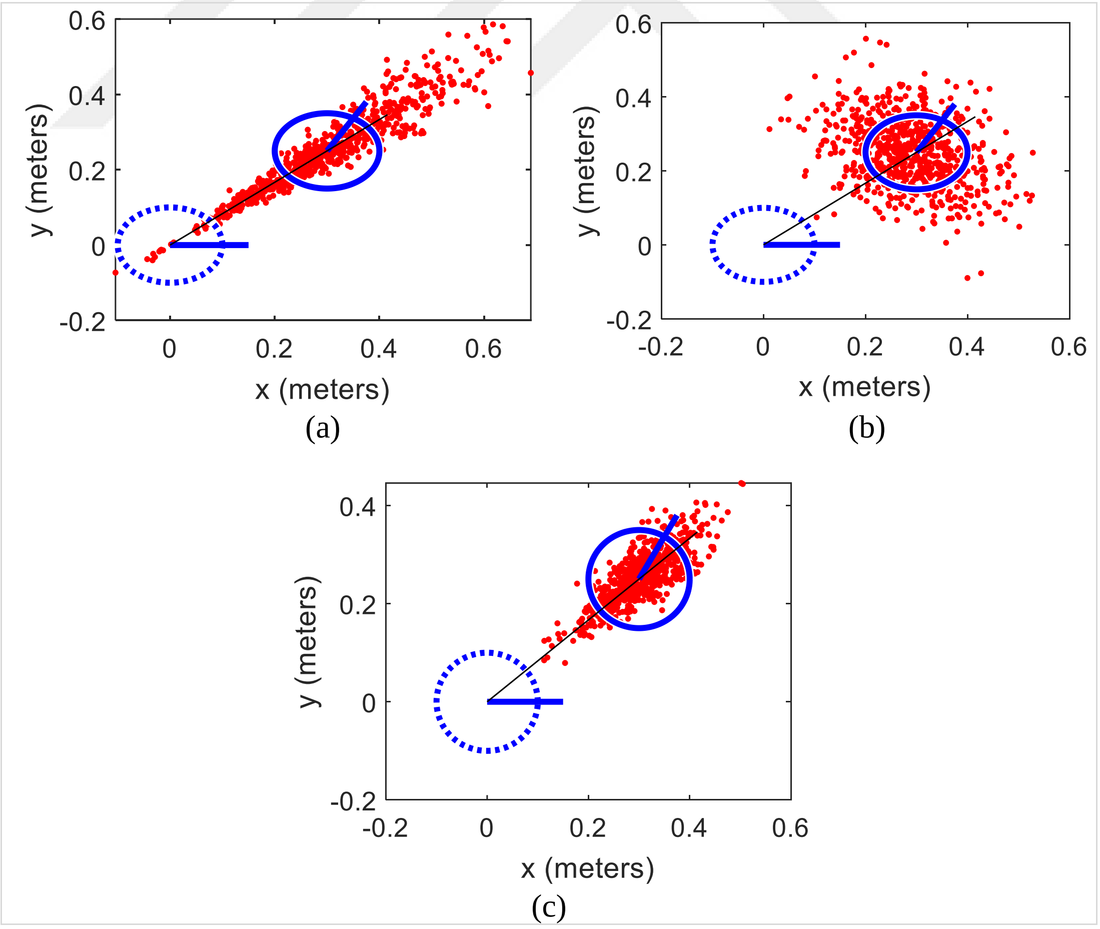


Figure 4.5. Generating 700 samples from the odometry motion model at different noise motion parameters

Figure 4.5 above, depicts a sampling of 700 samples based on the designed odometry motion model when the robot changes its pose from $[0\text{m}, 0\text{m}, 0\text{rad}]$ to $[0.3\text{m}, 0.25\text{m}, \frac{\text{pi}}{3}\text{rad}]$. These samples were generated by using different values of robot specific error parameters α. Figure 4.5 (a) denotes a high noise in translation, Figure 4.5 (b) shows a high noise in rotation, while Figure 4.5 (c) demonstrates the typical noise parameters. Motion noise parameters model the accuracy of the robot motion. These parameters are larger in cases when the robot is less accurate. In our thesis, the typical values for motion noise parameters were 0.2 for each.

4.2.2. Likelihood field range finder model

Our observation likelihood model uses the simulated LiDAR data discussed in the previous section to perceive the robot's environment. First and foremost, to increase the real-time performance of the observation model we pre-computed the likelihood in the form of a two-dimensional table. The results of the precomputation process discussed in Equation (2.17) are shown in Figure 4.6 below where we transformed the OGM map of the robot's environment to the likelihood field. As we see in the figure below, the likelihood value is one at the obstacles where the distance transform has a zero value, while going to the darker locations means that an obstacle is unlikely to be perceived there and the distance transform increases. The following Figure 4.6 from (b) to (c) depicts the likelihood field of the simple map at different values of measurement noise. The typical value of the measurement noise we have considered is 0.2 meters as illustrated in Figure 4.6 (b). Figure 4.6 (c) describes the likelihood field at a very small value for the measurement noise and it appears as if we neglect the noise in measurement data. However, Figure 4.6 (d) represents the likelihood field with high measurement noise that may affect adversely the results of the measurement model.

The likelihood model computes a likelihood for each scan endpoint independently, then multiply these values to obtain the measurement probability for the current scan as described in Table 2.3 (line 8). Therefore, we have to handle the large distance transform of the individual beam that leads to a likelihood value close to zero which in turn makes the measurement probability zero. We defined the maximum distance to find nearest obstacles on the map to be two meters, this distance prevents the likelihood of one ray to be zero at very long distance transform, then avoid the overall weight of the sample to be zero through multiplication process described in Equation (2.18). On the other side, we can improve the performance speed of the likelihood filed model by decreasing the number of rays included in the likelihood calculations. By skipping some rays, we reduced memory use, increased the real-time performance in terms of decreasing the execution time, also we prevented the total weight to be zero when multiplying very small fractional numbers of likelihood values.

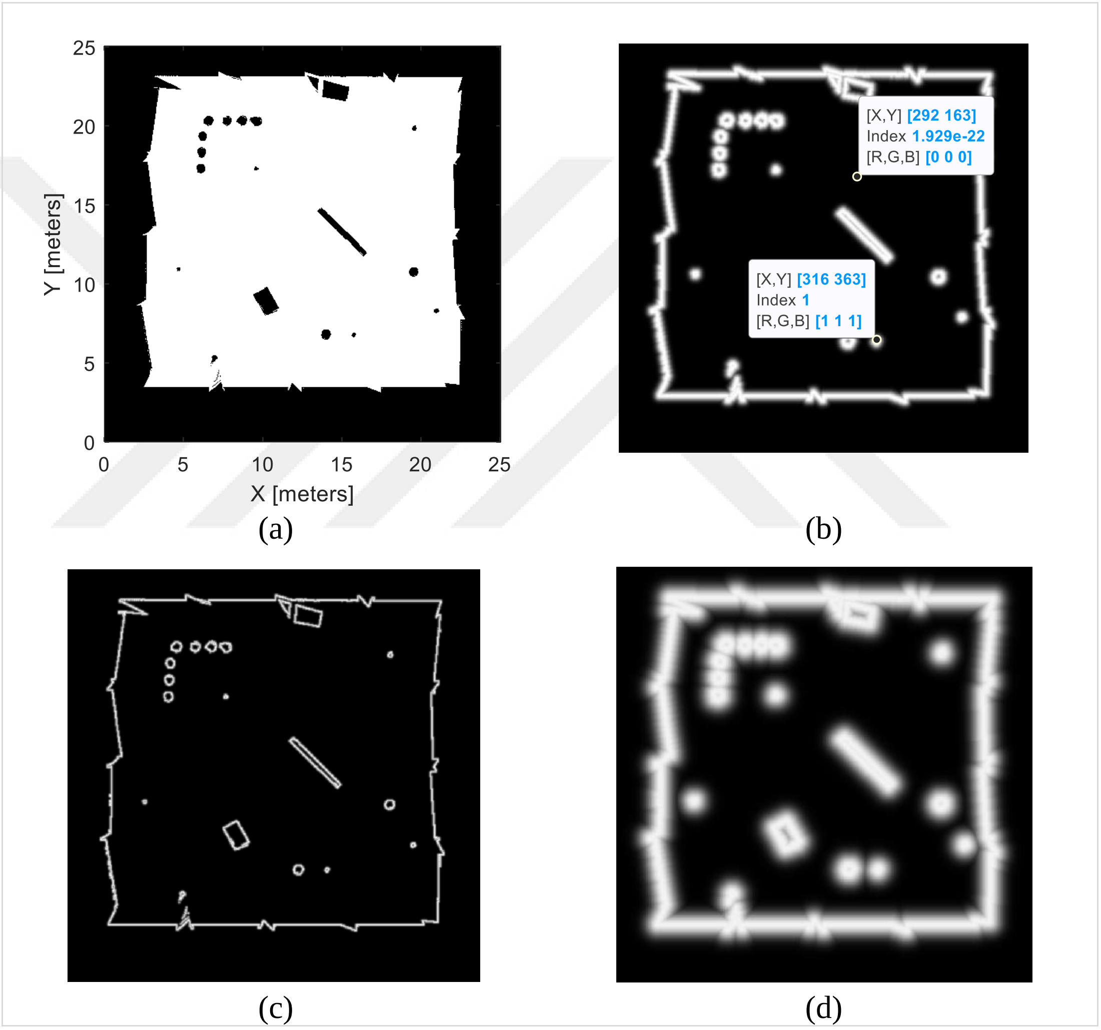


Figure 4.6. (a) OGM simple map, (b)-(d) likelihood field of the map with a standard deviation: (b) $\sigma = 0.2\text{m}$, (c) $\sigma = 0.05\text{m}$, (d) $\sigma = 0.5\text{m}$

In global localization, the samples are randomly distributed over the robot's working space. The weight of all samples before the correction step is equal. Figure 4.7 below, depicts the scan rays after skipping some of them and shows five samples generated normally across the predicted robot's space. The likelihood field model just uses the scan endpoints to check for the obstacles. The key idea is to match the most recent laser scan with the pre-mapped environment via the likelihood field. The current perception is applied to each hypothesis state and then we read the likelihood value of each endpoint from the likelihood table that computed beforehand.

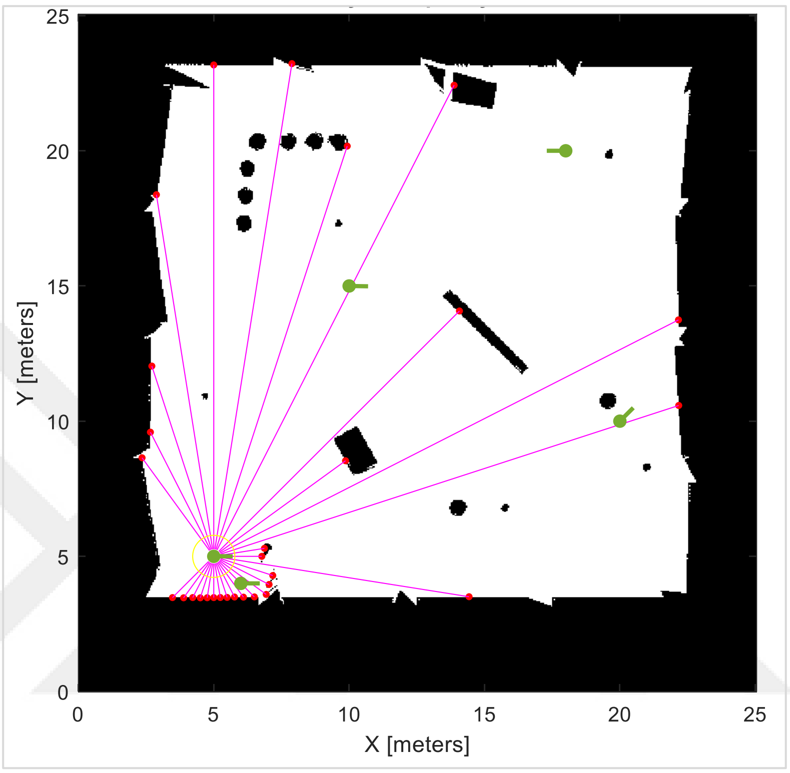


Figure 4.7. Scan rays emitted from the actual sensor's pose. The randomly generated samples appear in green color

Figure 4.8 below, demonstrates the likelihood approach for calculating the weight of the samples. Figure 4.8 (a) shows the actual robot's pose in blue and scan endpoints in red. Figures (b) to (f) represent the colored likelihood filed of the simple map and describe how the algorithm applies the current perception on each hypothesis state. Figure 4.8 (b) shows the perception data when applied to one sample that has the same state as the actual robot's state, we can note that the LiDAR data completely match the map. Thus, the likelihood of each endpoint will be high, and the weight of the corresponding sample will also be high. However, the scan endpoints shown in Figures (c) to (f) do not match the map, and the weight of each sample here will be different according to the number of rays hitting the obstacle. Special cases in Figure (c) and (d) where there are some endpoints off the map, such cases, we assumed the minimum likelihood value to handle the scan endpoints that occur outside the map.

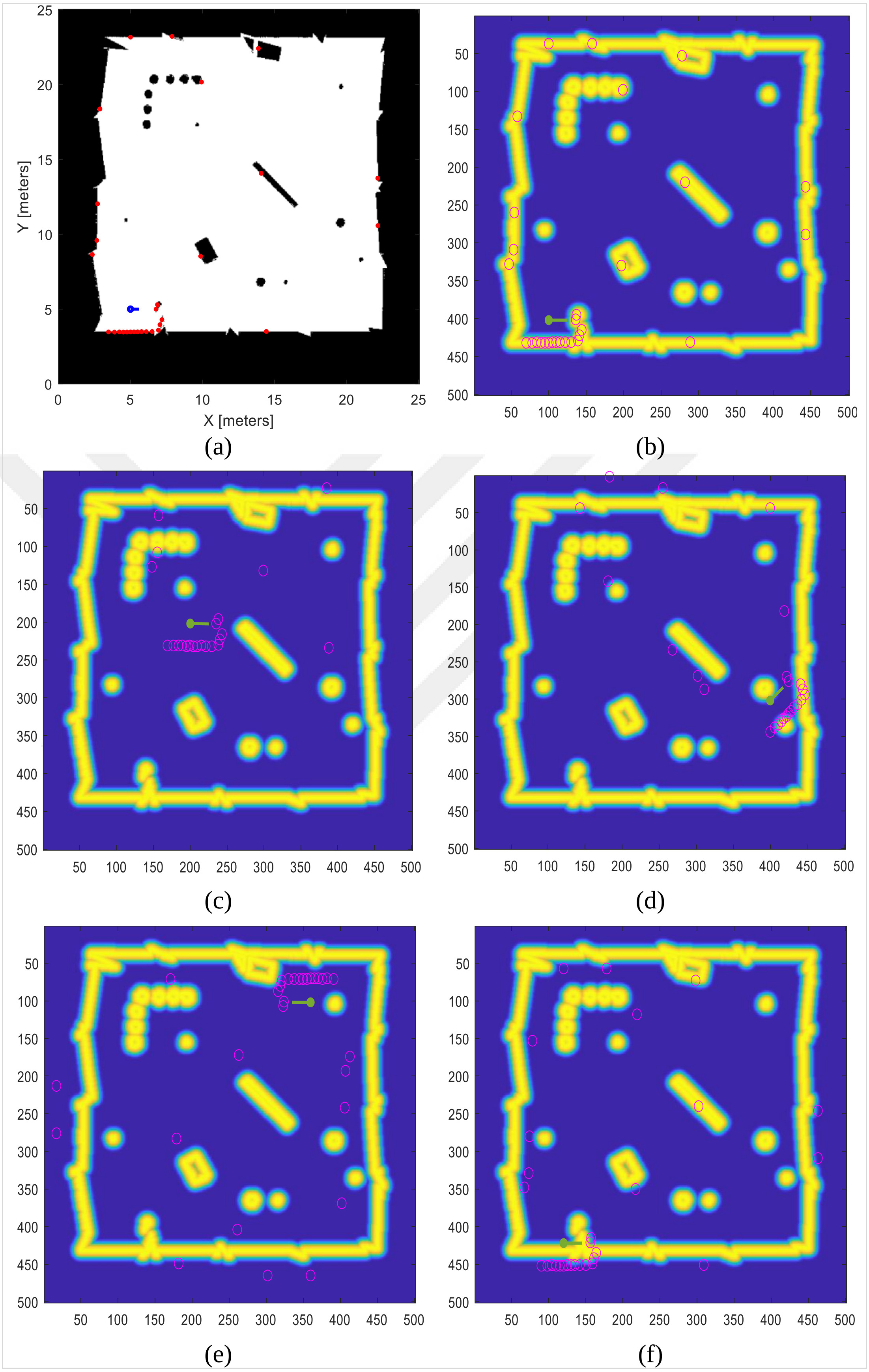


Figure 4.8. (a) actual robot's pose with the scan endpoints. figures (b)-(f) likelihood field of the simple map, also show how the scan applied on hypothesis states in green

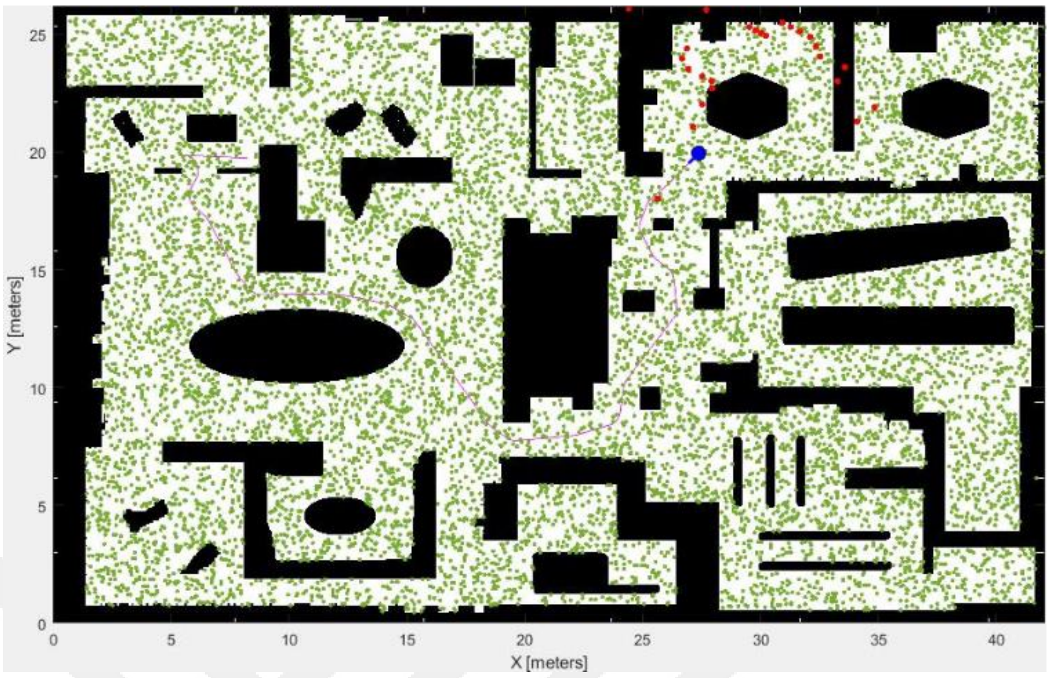
In the updating phase, the weight of the samples will be updated according to the sensor measurements (to what degree the scan matches the map). Therefore, the sum of the weights after the correction will not be equal to one. PDF of the multivariate normal distribution is evaluated based on the updated importance factors for the samples. Uncertainty was explicitly modeled in the observation model.

4.2.3. **Pose estimation**

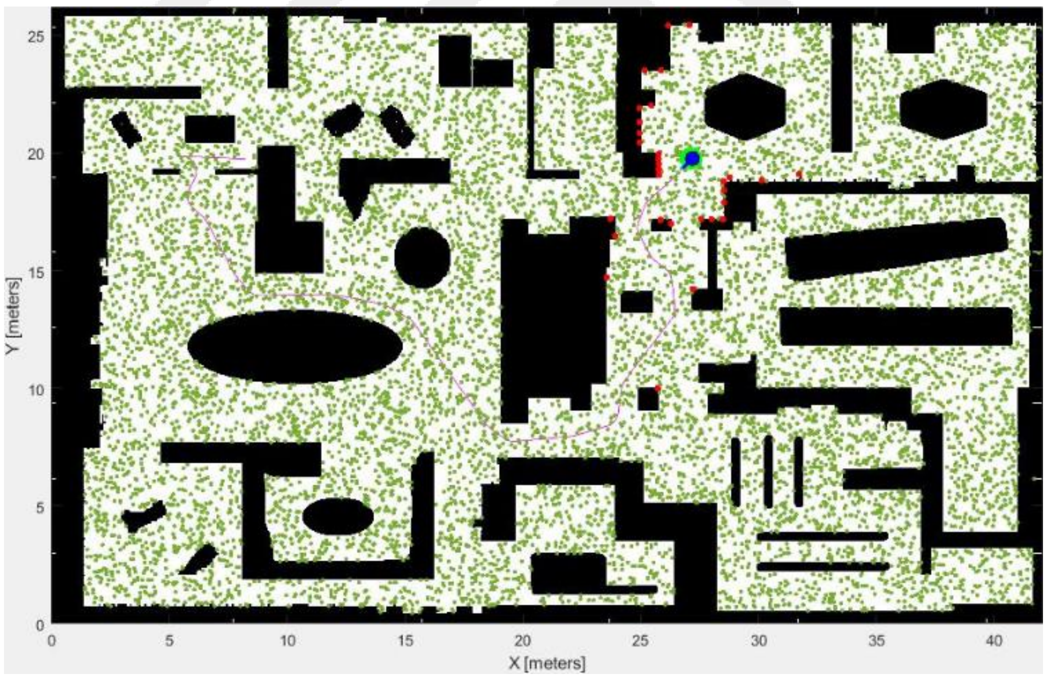
After assigning the weight of the samples based on the observation model, the MCL filter tries to estimate the robot's pose. The updated weights describe the probability of a sample to match the actual robot's pose. One way to infer the robot's pose is to take the sample with the maximum weight (best sample) as the estimated pose. However, by taking into account the error in each hypothesis state we consider all the samples around 0.50 meters of the best sample then we apply Equations (2.27) to (2.31) on these hypothesis states to compute the estimated pose.

The convergence term means that all hypothesis states are sufficiently close to each other. MCL filter convergence is achieved if 90% of samples are one meter close to the best sample. In the simulation environment, convergence quality is measured by checking the distance between the best sample and the actual robot's pose. However, in real life, we judge the quality of convergence based on the match ratio between the actual and emitted rays by the imaginative sensor installed on the best sample. If the filter is converged and the quality of convergence is greater than 90% then we postulate that the mobile robot has successfully localized itself in its indoor environment.

Figure 4.9 (a) below, illustrates the normal distribution of 15,000 samples of equal weight over the robot's state space within the factory map. The convergence quality is approximately zero where the scan measurement does not match the map (the best sample's state does not match the actual robot's pose). However, after two update iterations, the weight of samples change under the likelihood model and the best sample is inferred near the robot's pose as shown in Figure 4.9 (b). The match ratio was 90%.



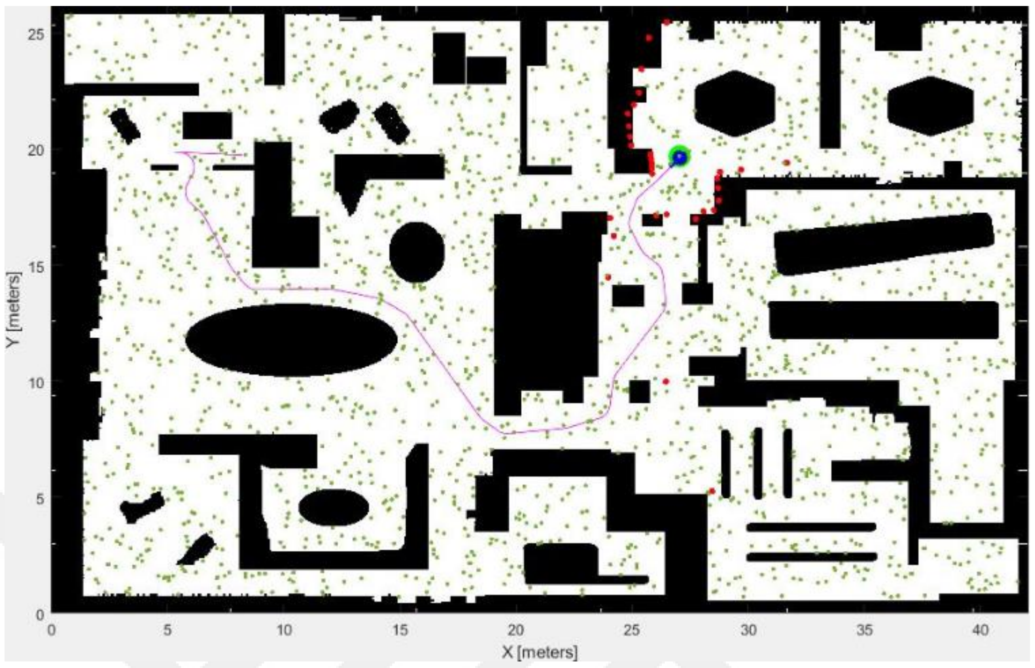
(a)



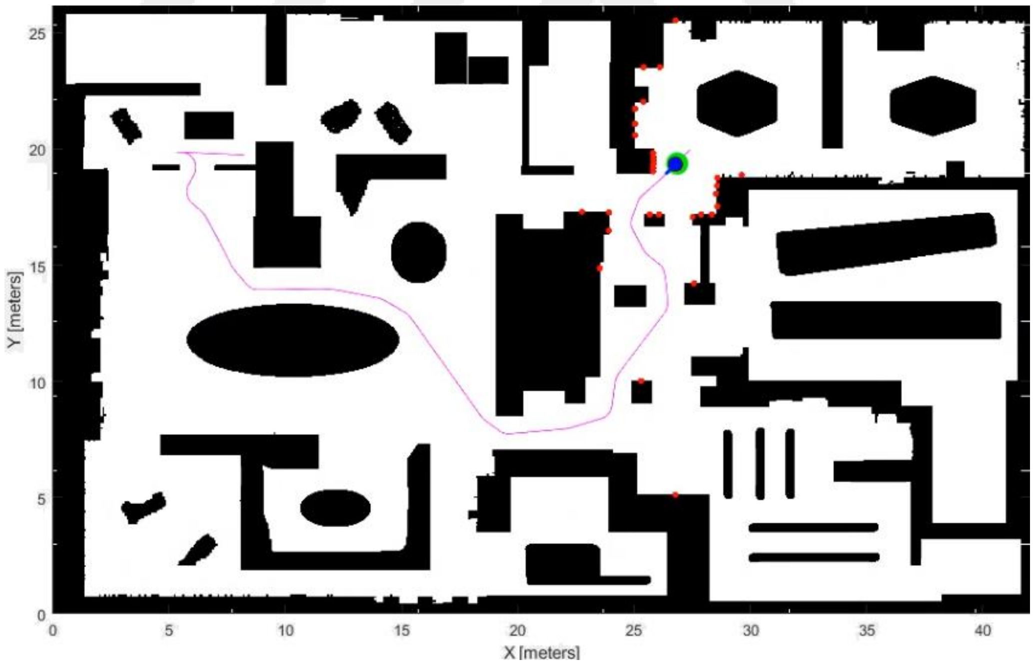
(b)

Figure 4.9. Global Localization problem. The mobile robot appears in blue, while the scan in red. (a) 15,000 particles (dark green) were distributed uniformly over the working space of the factory map. (b) Robot’s pose is estimated (light green) after two corrections

4.2.4. **Resampling phase**



(a)



(b)

Figure 4.10. Resampling phase. (a) after the first iteration of resampling, (b) after the third iteration of resampling

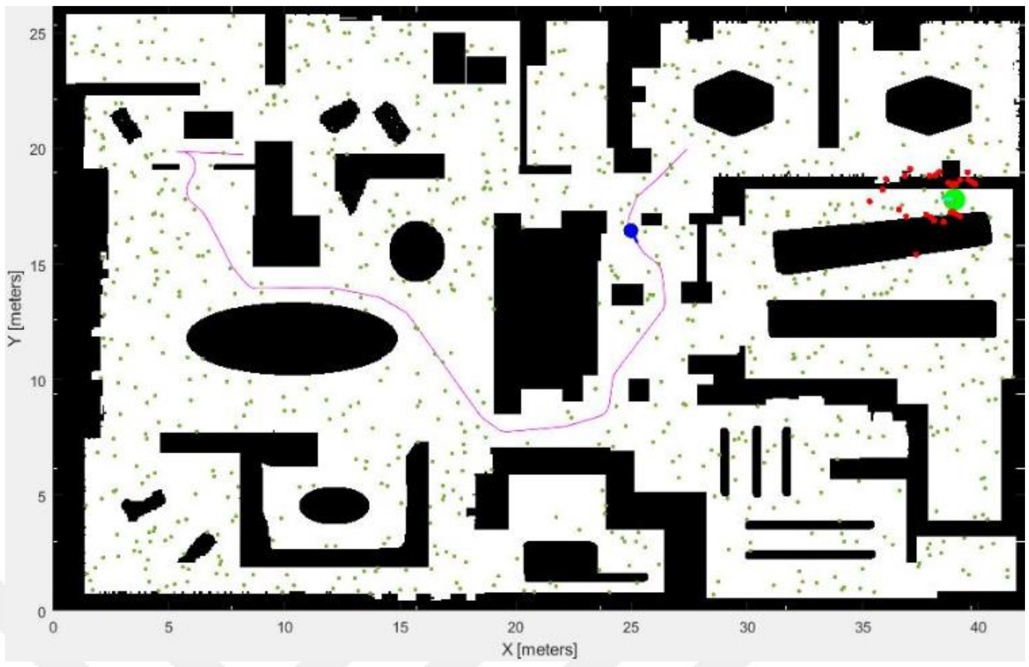
The weight of the samples was initialized by one and then corrected multiplicatively under the likelihood observation model. Since the number of samples is limited, each sample will survive based on its importance weight. After two update iterations, the resampling phase takes place to concentrate the computational burden of the MCL filter on the high-likelihood regions in the state space.

As discussed before in section 2.4.2 we have two different directions to do resampling based on the ratio between fast and slow average weight of the likelihood observation model. On the first side when the short-term average is better than the long-term average, low-weight samples will inherit the pose of high-weight samples (samples are inferred by replacement). Figure 4.10 above shows the continuation of the pose estimation process that started in Figure 4.9, where Figure 4.10 (a) shows how some of the low weight samples inherit the pose of high weight samples after one resampling iteration. However, some low weight samples survived by the stochastic resampling method, and this will help overcome disturbances. The localization performance in this step was 93%. Figure 4.10 (b) depicts how all samples converged to the robot's pose (high likelihood region) after three resampling iterations. The convergence quality was 97%.

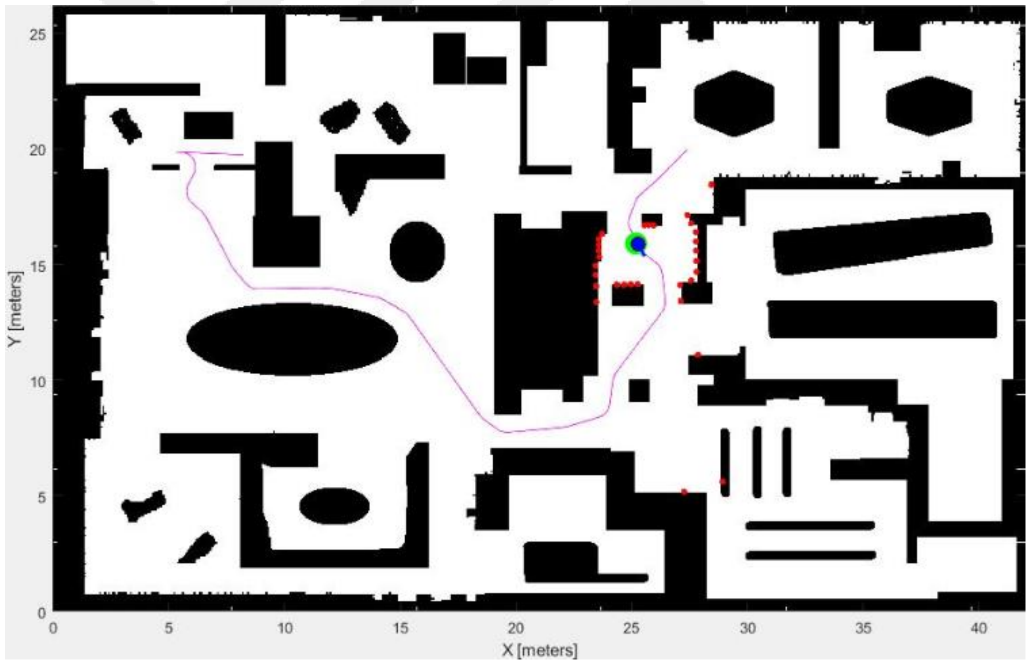
However, a sudden descent in the short-term average value compared to the long term average signposts a decrease in the localization efficiency, and the resampling algorithm discussed in Table 2.5 will try to add random samples that are proportional to the amount of decrease in the performance of localization to envisage other poses.

During the mission, the mobile robot may be kidnapped or the localization may fail due to a drop in probabilities. The following Figure 4.11 illustrates the reaction of the filter when the robot encounters dynamic obstacles for a couple of seconds, where the probabilities dropped down and the localization performance was 32%. MCL filter adds random samples according to the uniform distribution all over the state space as illustrated in Figure 4.11 (a). The number of generated samples was based on the divergence between long-term and short-term averages of the observation model. After 5 resampling iterations, the mobile robot successfully localizes itself on the map as shown in Figure 4.11 (b), the quality of convergence was 93%.

Again, after each resampling iteration, the weights of all samples sum up to one. Resampling transforms a set of samples into another collection of the same sized samples. The resampling of samples should be stopped when the mobile robot stops moving.



(a)



(b)

Figure 4.11. Resampling based on augmented MCL. (a) scattering random samples as a result of localization failure, (b) robot successfully localize itself again after 5 iterations of resampling

However, determining the appropriate frequency to sample is a matter of experience. High frequency is likely to result in loss of diversity, while low frequency results in loss of samples in a low likelihood states. The estimator variance or loss of diversity is demonstrated in the sample distribution as an approximation error. Our MCL algorithm sample when the mobile robot moves at least linear distance 20 cm or rotates 30 degrees since the robot can localize itself based on the odometry reading when it moves small movements, and resample the particles every two update iterations.

4.3. Proposed Schemes

This section outlines the simulation results and analyses for evaluating the efficacy of our proposed schemes compared to the conventional approaches of augmented MCL. A fair comparison is conducted on the basis of the number of samples needed to solve the global localization problem, to recover the robot in localization failures, and to recover the kidnapped robot. Besides, this comparison is done in terms of traveled distance, and the time required for the robot to localize itself in its world. First, we presented the results and analysis of the optimized global localization technique that detects mapped regions with high probabilities only based on the initial scan data. Second, we outlined the results of our effective resampling strategy that deals with the kidnapped robot problem and enables the robot to recover quickly when the sample weights drop-down due to unmapped dynamic obstacles within the sensor's field of view. Both of these novel methodologies have the potential to significantly reduce uncertainty in the global indoor localization problem.

Our differential mobile robot has a maximum linear velocity of 0.5 meters per second. For each sample set (10k, 5k, 1k, 500 samples), the localization algorithm was executed 30 times over each prepared OGM map. After that, we took the average results for these executions. As we discussed above, each sample drawn with the odometry motion model will be updated twice before resampling it.

4.3.1. Improved global localization

Our optimized technique relies only on the initial LiDAR scan z_1 gathered in the immobile state at the initialization phase. In this case, the mobile robot gets knowledge about the features of its current region while it has no prior knowledge of its current initial state. The proposed algorithm begins estimating the best sample distribution based on the initial perceptual data using the observation model. When getting this estimation, the available samples will spread over the high likely areas with high

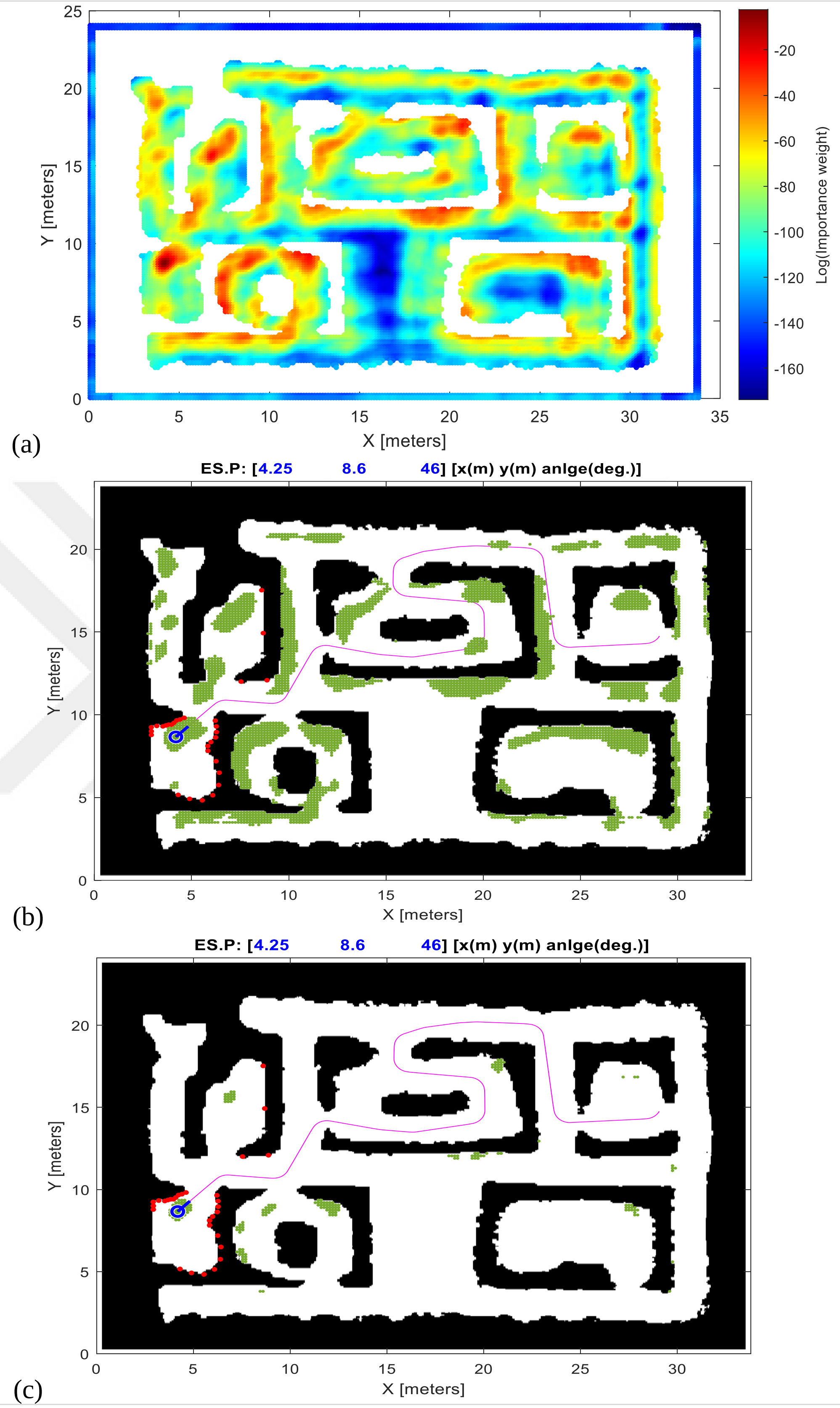


Figure 4.12. Maze map. (a) The logarithmic scale of grid cells probability. (b) and (c) represent distributing 10k and 500 samples over the areas with the highest probabilities while the mobile robot still in the immobile state

measurement probabilities, starting from the highest probability until all samples are distributed.

By applying the likelihood model on each grid cell with different orientations as discussed in section 3.2, the robot gets the information about how the measurement probabilities carried on the pose configuration space with respect to its current pose as illustrated in Figure 4.12 (a) above. While the mobile robot randomly placed on the maze map, the algorithm shows that the region around the mobile robot has the highest probabilities. However, it keeps also some far-away regions with high probabilities to prevent any localization failures due to unnatural sensor noise.

Figure 4.12 (b) and (c) above represent the scattering of 10k and 500 samples according to the uniform normal distribution over the pose space, respectively. As we see, even with a few numbers of samples, the proposed technique achieved a precise initial distribution that represents the initial belief. We utilized the uniform distribution to describe the belief in the expected areas while the density is zero anywhere else. However, we reduced the computational burden by searching the robot's pose every 3 cells, resulting in a 15cm search resolution in *x* and *y* directions. Equivalently, 3 degrees resolution is adapted for the angle search space in each grid cell. Thus, for a map of size 15m x 15m, 101x101x120 different pose values are evaluated.

Since the pose estimation is done after the update phase, we tried in Figure 4.12 (b) and (c) to estimate the mobile robot’s pose based on the sample with the maximum weight. The actual pose for the robot was [4.20m 8.66m 45degrees], we noticed that the initial robot’s state that matches the map achieved precisely by our proposed technique.

Once the robot makes a simple movement, the filter that uses our proposed method converges to the actual robot’s state faster than the traditional approach, as shown in Figure 4.13 below. In the proposed algorithm, there are many high probability samples near the true robot’s pose, so, after one iteration of the resampling process, all samples with low probabilities converge to the true pose. By employing the proposed method, the mobile robot successfully localized itself in the global environment for the two sample sets (10k & 500 samples) in record time, while the traveled distance was (0.41m & 0.26m), respectively. Since the computational time for small sample sets is faster than the larger sets, they survive faster and maintains high performance during missions. But this is not applicable at all for the conventional approach, since at the global localization, samples are scattered randomly over the pose space, and they need time to catch the high likelihood region. The two sample sets (10k & 1k samples) that used the conventional method were able to locate the robot in its environment after traveling a distance of 4.34m and 15.15m, respectively.

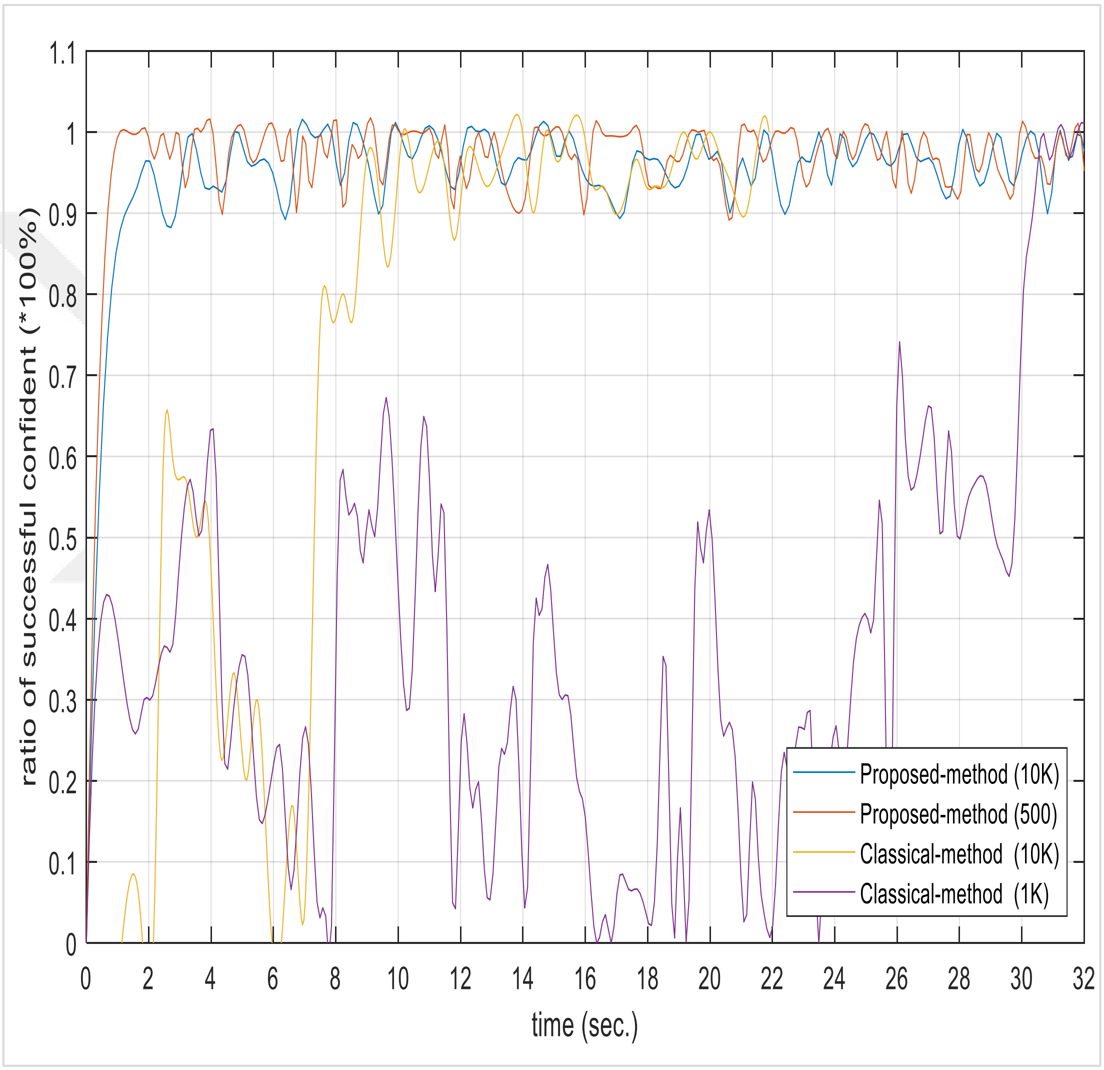


Figure 4.13. The ratio of successful confident during the global localization problem of the proposed and traditional method on the maze map

However, in the traditional approach that illustrated in Figure 4.13, the sample set (500 samples) failed to locate the robot within 60 seconds in the maze map, since this type of maps includes a lot of similar areas. Contrariwise, the same sample set (500 samples) was able to locate the mobile robot in the simple map after the robot traveled 7.34 meters as shown in Figure 4.14 below, since this map is small and simple.

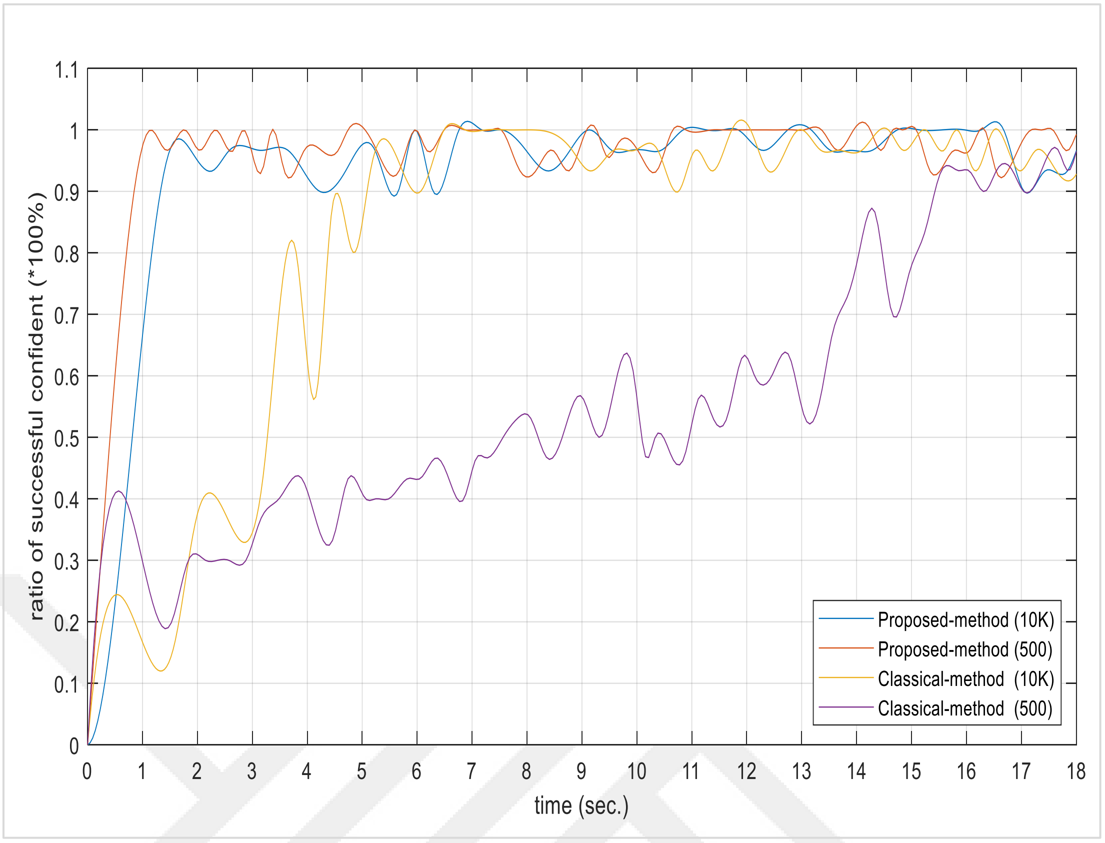


Figure 4.14. The ratio of successful confident during the global localization problem of the proposed and traditional method on the simple map

4.3.2. Improved resampling scheme

The crux of our proposed algorithm is how the robot behaves when losing its convergence during tasks. Since the mobile robot locates itself reliably in the environment using our proposed global localization algorithm illustrated above, the filter may lose the convergence when the robot encounters unmapped dynamic obstacles like humans or due to sensors noise or whatever that causes dropping in probabilities. In this case, the augmented MCL will add random samples to the sample set based on the divergence between fast and slow average weights. However, the proposed resampling technique infers the poses of random samples under the uniform normal distribution over a specified area rather than drawing them over all possible poses on the map as the traditional MCL does.

Figure 4.15 below, illustrates how our proposed resampling method compared to the traditional resampling approach behaves when the mobile robot runs across unmapped dynamic objects.

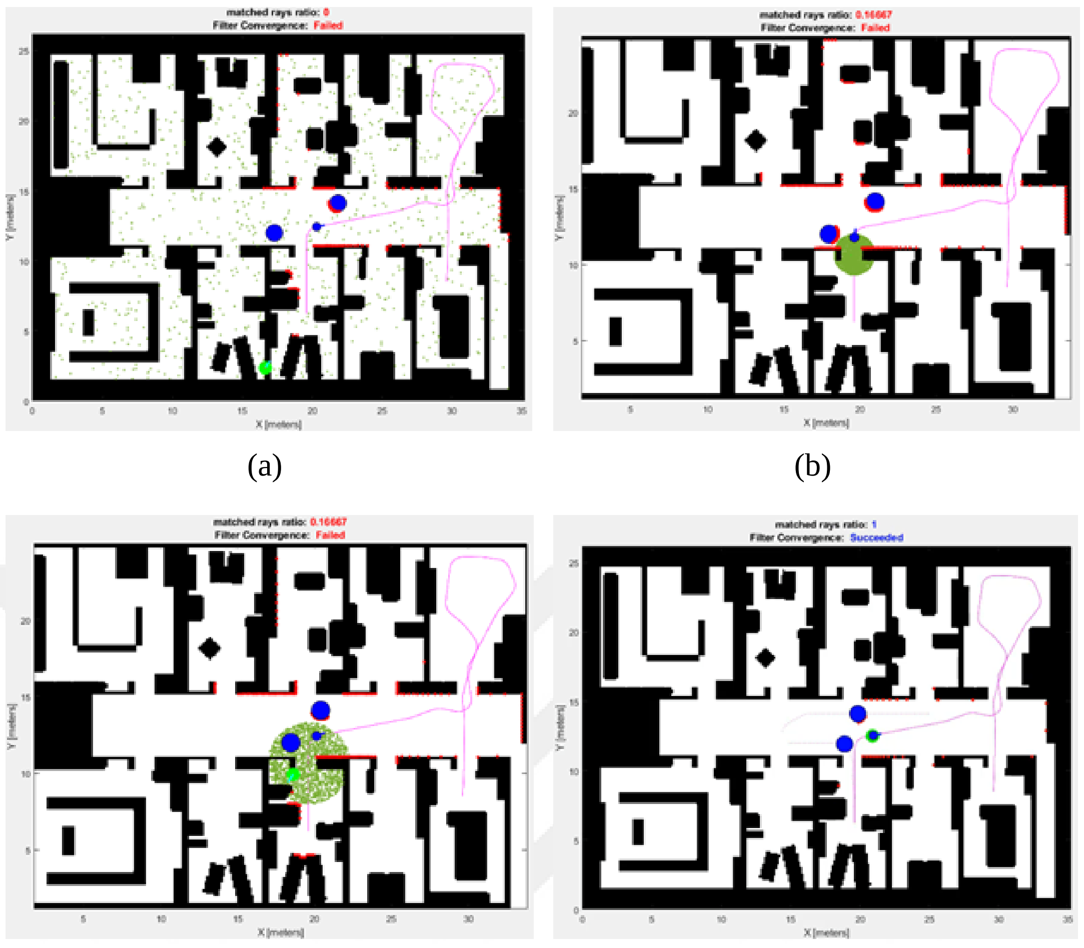


Figure 4.15. The robot is trying to recover itself on the symmetric map after losing its convergence due to two dynamic objects. (a) the response of the resampling algorithm of the conventional stochastic MCL. (b)-(d) the response of our improved resampling strategy

As the measurement probabilities drop-down, the robot will think that it is getting lost (although it is not) and random samples will be added to the sample set based on the divergence between fast and slow average weights to estimate the correct distribution accurately. Figure 4.15 (a) illustrates how the classical MCL distributes the samples over the robot's state space without taking into account any prior knowledge about the last confident robot's state. However, Figure 4.15 (b), (c), and (d) depict the response of our resampling method at the same criteria. Figure 4.15 (b) demonstrates how the improved resampling strategy restricts the region to add samples to a circular region around the last successful estimation based on the maximum velocity of the mobile robot. The area of the selected region where the robot has a chance to locate itself there increases over time if the robot still cannot recover itself as shown in Figure 4.15 (c).

As a result of this technique, our robot has a great opportunity to estimate its pose promptly once the impact of the dynamic obstacles is gone as depicted in Figure 4.15 (d).

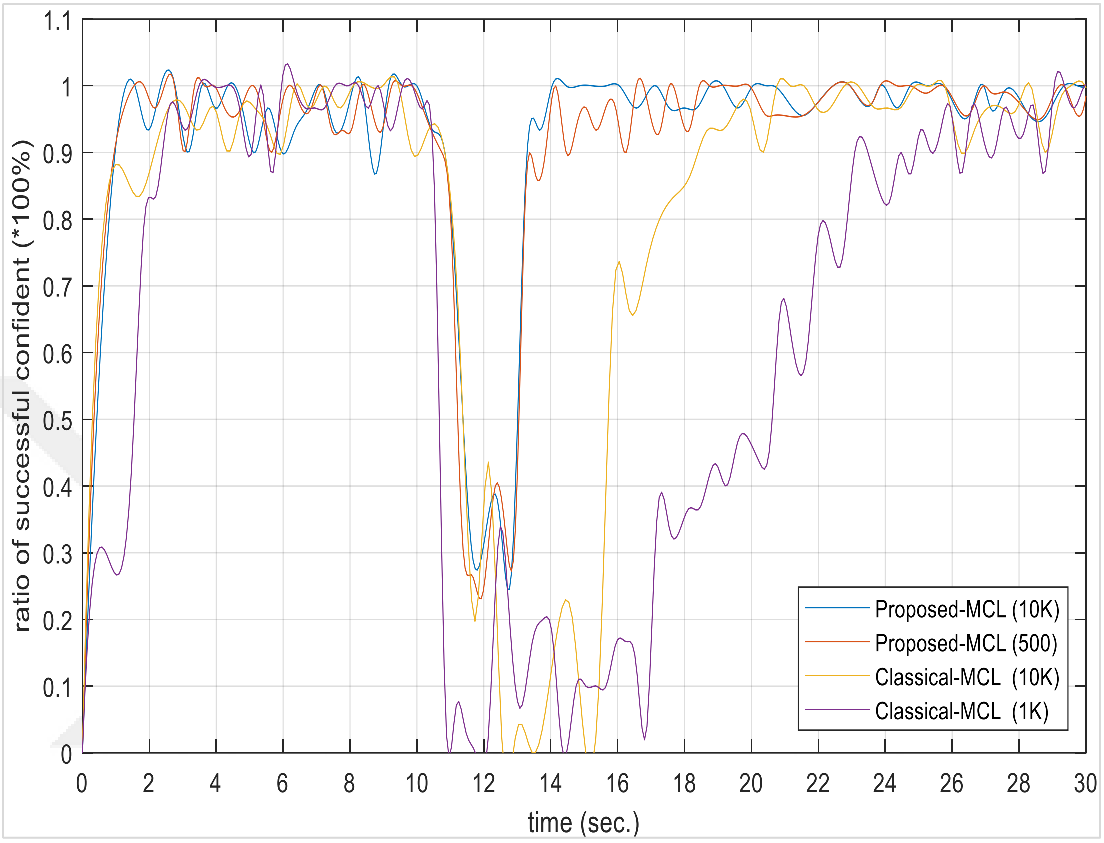


Figure 4.16. The proposed and conventional resampling scheme responses when the probabilities drop down due to disturbances on the symmetric map

The new approach guarantees to resample any high probability samples to a certain area around the robot's pose rather than publishing them randomly over the robot's working space. In this way, the probability of the robot recovering itself in record time is very high and the distance traveled by the robot when it loses itself will be very short compared to the classical approach as shown in Figure 4.16 above. Figure 4.16 compares the responses of the proposed and conventional resampling schemes discussed in Figure 4.15; the robot performs its mission at maximum velocity. After 10 seconds, the robot encounters two unmapped dynamic objects that cause a drop in probabilities for three seconds. The proposed approach recovered the robot directly when the effect of the dynamic objects disappeared. In this case, the distance traveled by the robot around one meter for two tested sample sets (10k & 500 samples). As we see, even a small sample set (500 samples) recovered the robot very efficiently.

Under the same criteria, the conventional method recovered the robot after 4.37 meters in 9.77 seconds for 10k samples and around 7.47 meters in 14.09 seconds for 1k samples as illustrated in Figure 4.16. Also, we noticed that when the size of the sample set decreases the filter needs more time for recovering as the samples spread over the pose space, where the small sample set needs time to catch the high probability area. Again, the sample set (500 samples) failed to locate the robot within 60 seconds in this map, since it contains symmetric regions. Note, regarding fast converging for the classical approach at initialization that shown in the figure above, instead of using global localization we used the local distribution method to publish all samples around the robot pose in order to speed up the convergence process at a start-up where we were not interested in this test.

By utilizing the proposed approach, fewer samples will be enough to re-infer the true belief of the mobile robot's state. Therefore, the real-time performance of the algorithm will increase in terms of decreasing the computational burden. Besides, since the samples are distributed over a specified region with high probabilities, the samples will take less time to converge to the actual robot's state.

Table 4.2 through Table 4.5 below summarizes the results for our proposed approaches compared to the conventional methods regarding the global localization and resampling schemes over different environments by employing different sample sets (10k, 5k, and 500 samples). We noticed that our proposed approaches showed significant improvement in terms of time and distance traveled by the robot until successfully localize itself, either at global localization or even when it loses itself during missions. The small sample set (500 samples) showed very efficient results where the percentage of improvement exceeded 90% over the large maps and played a big role in reducing computational resources. On the other hand, the percentage of improvement over the simple map was less than 75% and in other cases less than 50%, this is a result of the map's simplicity and its small size. However, the results of the proposed resampling strategy show an increase in time and distance as the sample set size decreases, this is because when the specified area increases the large sample set is able to cover that area better than small sample set, which means there are many samples will be around robot's pose. But this is solved by AMCL, where it can adapt the number of samples when the robot loses its pose.

Table 4.2. Symmetric map results [35m X 26m]

		Conventional approach		Proposed approach			
		Global Localization	Resampling	Global Localization	%	Resampling	%
10k	[sec.]	5.529	9.771	1.152	79.16	1.886	80.70
	[m]	2.16	4.37	0.37	82.87	0.92	78.95
5k	[sec.]	8.590	15.010	0.993	88.44	2.069	86.22
	[m]	3.39	6.64	0.33	90.27	0.94	85.84
500	[sec.]	>60	>60	0.833	98.61	3.035	94.94
	[m]	>25	>25	0.24	99.04	1.34	94.64

Table 4.3. Factory map results [42m X 26m]

		Conventional approach		Proposed approach			
		Global Localization	Resampling	Global Localization	%	Resampling	%
10k	[sec.]	4.503	9.712	1.539	65.82	3.510	63.86
	[m]	1.71	4.08	0.52	69.59	1.46	64.22
5k	[sec.]	6.992	15.310	1.120	83.98	3.597	76.51
	[m]	2.62	6.36	0.38	85.50	1.51	76.26
500	[sec.]	>60	>60	0.855	98.58	5.110	91.48
	[m]	>25	>25	0.25	99.00	2.12	91.52

Table 4.4. Maze map results [33m X 24m]

		Conventional approach		Proposed approach			
		Global Localization	Resampling	Global Localization	%	Resampling	%
10k	[sec.]	10.263	9.027	0.997	90.29	2.318	74.32
	[m]	4.07	5.23	0.33	91.89	1.68	67.88
5k	[sec.]	13.386	10.806	0.860	93.58	2.903	73.14
	[m]	5.31	6.52	0.29	94.54	1.96	69.94
500	[sec.]	>60	>60	0.753	98.75	3.999	93.34
	[m]	>25	>25	0.23	99.08	2.10	91.60

Table 4.5. Simple map results [25m X 25m]

		Conventional approach		Proposed approach			
		Global Localization	Resampling	Global Localization	%	Resampling	%
10k	[sec.]	3.526	7.013	1.312	62.79	3.389	51.68
	[m]	1.58	3.90	0.52	67.09	2.03	47.95
5k	[sec.]	4.234	7.854	1.129	73.33	3.808	51.52
	[m]	1.95	4.07	0.44	77.44	2.63	35.38
500	[sec.]	13.437	13.637	0.738	94.51	5.304	61.11
	[m]	6.58	6.61	0.25	96.20	3.50	47.05

5. CONCLUSIONS AND FUTURE WORK

This thesis demonstrates two novel approaches that improve mobile robot global positioning and resampling process in indoor environments. The proposed techniques allow the mobile robot to infer its initial belief precisely and recover itself promptly when the robot kidnapped or when the localization fails.

First, the MCL algorithm is implemented based on the PF to overcome the uncertainty in the mobile robot's localization problems. The odometry motion model and the likelihood field range finder model were carried out to originate and correct samples over the robot's configuration space, respectively. MCL filter estimates the mobile robot's state by utilizing noisy sensors data, such as LiDAR sensor. Besides, the augmented MCL is presented to accomplish the resampling phase.

The problem of global localization involves thousands of samples, especially when processing large maps, which reduces the real-time performance of the localization due to the increase in the computational effort. However, we presented two techniques that improve the global localization and resampling phase in indoor environments. First, by taking the initial scan measurements into account, our improved global localization approach can detect the high likelihood regions based on the observation model, leading to the best sample distribution in high-probability areas instead of randomly scattering samples. Second, we devised a novel scheme that urged the localization algorithm to add the random samples around the robot's position rather than over the robot's state space when the probabilities drop-down by taking into account the prior knowledge about the most recent confident estimate pose.

Several simulations were carried out to evaluate our localization approaches. The results demonstrate great superiority of the proposed schemes over the conventional localization methods. Where it showed reliable and fast global localization, even with small sample sets. Moreover, it showed a smart resampling technique that enables the robot to recover quickly when probabilities drop-down. This resulted in a noticeable increase in real-time performance in terms of decreasing the computational cost. Both of these novel methodologies demonstrate their potential to significantly reduce uncertainty in the global indoor localization problem. Furthermore, small sample sets have demonstrated their ability and high efficiency to localize mobile robots regardless of the size of indoor environments.

However, there are limitations to the proposed approaches. The environment of the mobile robot should not be highly symmetric nor should the dynamic map differ significantly.

To further improve the computational time at global localization, we will combine our improved global localization algorithm with the Hough Scan Matching (HSM) [51] method to perform a global angle search. Based on the HSM the algorithm will filter a very specific number of heading directions to provide them for the observation model to start the process of searching the high probability areas.

Moreover, since the step of the skipping beams that have done before providing laser rays to the observation model helped in reducing the effect of the dynamic obstacles in dropping the probabilities, we will propose a method to actually detect dynamic obstacles based on comparing consecutive sensor scans. Sensor rays that hit the detected dynamic obstacles will be excluded from measurement probability estimation. Consequently, measurement probabilities will not drop due to dynamic obstacles.

REFERENCES

[1] Wongphati M., Matsuda Y., Osawa H., Imai M., Where Do You Want to Use a Robotic Arm? And What Do You Want from the Robot?, *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication,* 2012.

[2] Diolaiti N., Melchiorri C., Teleoperation of a Mobile Robot through Haptic Feedback, *Proceedings: HAVE 2002 - IEEE International Workshop on Haptic Virtual Environments and their Applications*, 2002.

[3] Kim K., Lee H., Park J., Yang M., Robotic Contamination Cleaning System, *IEEE International Conference on Intelligent Robots and Systems*, 2002.

[4] Smith F. M., Backman D. K., Jacobsen S. C., *Telerobotic Manipulator for Hazardous Environments, Journal of Robotic Systems*, 1992, **9**(2), 251–260.

[5] Lin Q., Kuo C., Virtual Tele-Operation of Underwater Robots, *Proceedings - IEEE International Conference on Robotics and Automation*, 1997.

[6] Siegwart R., Nourbakhsh I. R., Scaramuzza D., *Introduction to Autonomous Mobile Robots,* 2nd ed., MIT Press, Massachusetts, 2011.

[7] Williams S. B., Pizarro O., Webster J. M., Beaman R. J., et al., *Autonomous Underwater Vehicle-Assisted Surveying of Drowned Reefs on the Shelf Edge of the Great Barrier Reef, Australia, Journal of Field Robotics*, 2010, **27**(5), 675–697.

[8] Ishikawa M., Trident Snake Robot: Locomotion Analysis and Control, *IFAC Proceedings Volumes (IFAC-PapersOnline),* 1 September 2004.

[9] Everett H. R., *Sensors for Mobile Robots*, Crc Press, New York, 1995.

[10] Dellaert F., Fox D., Burgard W., Thrun S., Monte Carlo Localization for Mobile Robots, *Proceedings - IEEE International Conference on Robotics and Automation,* Detroit, USA, 1999.

[11] Fox D., Burgard W., Dellaert F., Thrun S., Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, *Proceedings of the National Conference on Artificial Intelligence*, 1999.

[12] Rohani M., Gingras D., Gruyer D., Dynamic Base Station DGPS for Cooperative Vehicle Localization, *2014 International Conference on Connected Vehicles and Expo, ICCVE 2014 - Proceedings,* 2014.

[13] Milstein A., Sánchez J. N., Williamson E. T., Robust Global Localization Using

Clustered Particle Filtering, *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[14] Weiss G., Wetzler C., von Puttkamer E., Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans, *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, 1994.

[15] Burgard W., Derr A., Fox D., Cremers A. B., Integrating Global Position Estimation and Position Tracking for Mobile Robots: The Dynamic Markov Localization Approach, *IEEE International Conference on Intelligent Robots and Systems*, 1998.

[16] Engelson S. P., Passive Navigation and Visual Place Recognition, *Doctoral dissertation, Yale University*, 1994.

[17] Engelson S. P., McDermott D. V., Error Correction in Mobile Robot Map Learning, *Proceedings - IEEE International Conference on Robotics and Automation*, 1992.

[18] Burgard W., Fox D., Hennig D., Schmidt T., Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids, *Proceedings of the National Conference on Artificial Intelligence*, 1996.

[19] Schultz A. C., Adams W., Continuous Localization Using Evidence Grids, *Proceedings - IEEE International Conference on Robotics and Automation*, 1998.

[20] Fantian K., Youping C., Jingming X., Gang Z., Zude Z., Mobile Robot Localization Based on Extended Kalman Filter, *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, 2006.

[21] Kwon S. J., Yang K. W., Park S., An Effective Kalman Filter Localization Method for Mobile Robots, *IEEE International Conference on Intelligent Robots and Systems*, 2006.

[22] Liu J., Yuan K., Zou W., Yang Q., Monte Carlo Multi-Robot Localization Based on Grid Cells and Characteristic Particles, *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, 2005.

[23] Gasparri A., Panzieri S., Pascucci F., Ulivi G., A Hybrid Active Global Localisation Algorithm for Mobile Robots, *Proceedings - IEEE International Conference on Robotics and Automation*, 2007.

[24] Zhang X., Chen X., Li J., Li X., Vision-Based Monte Carlo - Kalman Localization in a Known Dynamic Environment, *9th International Conference on Control, Automation, Robotics and Vision, 2006, ICARCV '06*, 2006.

[25] Thrun S., Burgard W., Fox D., *Probabilistic Robotics*, 3rd ed., MIT Press, United States, 2005.

[26] Grisetti G., Stachniss C., Burgard W., *Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters*, *IEEE Transactions on Robotics*, 2007, **23**(1), 34–46.

[27] Csorba M., Simultaneous Localisation and Map Building, PhD.Thesis, University of Oxford, Department of Engineering Science, Oxford, 1997, 8547.

[28] Thrun S., Fox D., Burgard W., Dellaert F., *Robust Monte Carlo Localization for Mobile Robots*, *Artificial Intelligence,* 2001, **128**(1–2), 99–141.

[29] Thrun S., *Bayesian Landmark Learning for Mobile Robot Localization*, *Machine Learning*, 1998, **33**(1), 41–76.

[30] Cassandra A. R., Kaelbling L. P., Kurien J. A., Acting under Uncertainty: Discrete Bayesian Models for Mobile-Robot Navigation, *IEEE International Conference on Intelligent Robots and Systems*, 1996.

[31] Fox D., Burgard W., Thrun S., *Markov Localization for Mobile Robots in Dynamic Environments, Journal of Artificial Intelligence Research*, 1999, **11**, 391–427.

[32] Leonard J. J., Durrant-Whyte H. F., *Mobile Robot Localization by Tracking Geometric Beacons*, *IEEE Transactions on Robotics and Automation*, 1991, **7**(3), 376–382.

[33] Jensfelt P., Kristensen S., *Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking, IEEE Transactions on Robotics and Automation*, 2001, **17**(5), 748–760.

[34] Thrun S., Particle Filters in Robotics, *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, 12 August 2002.

[35] Winkler G., *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods*, 2nd ed., Springer Science & Business Media, Neuherberg, 2012.

[36] Pyo Y., Cho H., Jung L., Lim D., *ROS Robot Programming (English),* 1st ed., ROBOTIS, South Korea, 2017.

[37] Anvari I., Non-Holonomic Differential Drive Mobile Robot Control & Design : Critical Dynamics and Coupling Constraints, Master Thesis, Arizona State University, Electrical Engineering, Arizona, 2013.

[38] Jia Q., Wang M., Liu S., Ge J., Gu C., Research and Development of Mecanum-Wheeled Omnidirectional Mobile Robot Implemented by Multiple Control Methods, *M2VIP 2016 - Proceedings of 23rd International Conference on Mechatronics and Machine Vision in Practice,* 19 January 2017.

[39] Konolige K., *Improved Occupancy Grids for Map Building*, *Autonomous Robots*, 1997, **4**(4), 351–367.

[40] Catapang A. N., Ramos M., Obstacle Detection Using a 2D LIDAR System for

an Autonomous Vehicle, *Proceedings - 6th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2016*, 5 April 2017.

[41] Craig J. J., *Introduction to Robotics: Mechanics and Control*, 3rd ed., Prentice Hall, New Jersey, 2005.

[42] Pawlak Z., Decision Rules, Bayes' Rule and Rough Sets, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1999.

[43] Gordon N. J., Salmond D. J., Smith A. F. M., Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation, *IEE Proceedings, Part F: Radar and Signal Processing*, 1993.

[44] Kong A., Liu J. S., *Sequential Imputations and Bayesian Missing Data Problems*, *Journal of the American Statistical Association*, 1994, **89**(425), 278–288.

[45] Choe G. M., Wang T. jiang, Liu F., Choe C. H., et al., *Anadvanced Integrated Framework for Moving Object Tracking*, *Journal of Zhejiang University: Science C*, 2014, **15**(10), 861–877.

[46] Hoang Vo T., BinarySearch(A, n, Num), MATLAB Central File Exchange, https://www.mathworks.com/matlabcentral/fileexchange/56271-binarysearch-a-n-num, (Access date: 1 June 2020).

[47] Bresenham J. E., *Algorithm for Computer Control of a Digital Plotter*, *IBM Systems Journal*, 1965, **4**(1), 25–30.

[48] Zhou X., Li X., Improved DDA Line Drawing Anti-Aliasing Algorithm Based on Embedded Graphics System, *2010 3rd International Conference on Advanced Computer Theory and Engineering (2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, 20 August 2010.

[49] Wu X., *An Efficient Antialiasing Technique*, *ACM SIGGRAPH Computer Graphics*, 1991, **25**(4), 143–152.

[50] Zingl A., A Rasterizing Algorithm for Drawing Curves, *Na*, 2012.

[51] Censi A., Iocchi L., Grisetti G., Scan Matching in the Hough Domain, *Proceedings - IEEE International Conference on Robotics and Automation*, 2005.

APPENDICES

APPENDIX-A

Table A.1. Bresenham’s line drawing algorithm for ray tracing

```
1:  **Ray_Tracing_Algorithm** $(x_1, y_1, x_2, y_2)$

2:      $X = Y = \emptyset$

2:      $d_x = x_2 - x_1$
3:      $d_y = y_2 - y_1$
4:      $P = 2d_y - d_x$

5:      If  $d_y > d_x$
6:         for $x = x_1$ to $x_2$ do
7:             add pixel $(x, y)$  to  $(X, Y)$
8:             $x = x + 1$
9:             If  $P < 0$
10:                $P = P + 2d_y$
11:            else
12:                $P = P + 2d_y - 2d_x$
13:                $y = y + 1$
14:            endif
15:        endfor
16:     else
17:        for $y = y_1$ to $y_2$ do
18:            add pixel $(x, y)$  to  $(X, Y)$
19:            $y = y + 1$
20:            If  $P < 0$
21:                $P = P + 2d_x$
22:            else
23:                $P = P + 2d_x - 2d_y$
24:                $x = x + 1$
25:            endif
26:        endfor
27:     endif

28:     return $[X, Y]$
```

PUBLICATIONS

[1] **Abualkebash H.**, Ocak H., Improved Global Localization and Resampling Techniques for Monte Carlo Localization Algorithm, *9th International Conference on Advanced Technologies (ICAT'20)*, Istanbul, Turkey, 10-12 August 2020.

[2] **Abu-Alkebash H.**, Bader A., Hashlamon I., Three Degree of Freedom Delta Robot, Design, Control, and Implementation for Educational Purposes, *Australian Journal of Basic and Applied Sciences*, 2017, **11**(5), 126–132.

RESUME

Humam AbuAlkebash was born in 1992 in Jebel Akhdar, Libya. After finishing his high school from Hisham Ben Abdalmalik Secondary School Jericho, Palestine, he started Mechatronics Engineering from Palestine Polytechnic University (PPU) Hebron, Palestine, and graduated in 2015. Amid his BS studies in Mechatronics, he trained in several factories, such as Palestinian Joint Venture for the Manufacture of Iron and Steel, and Nieroukh Scales & Metallic Furniture Company. Moreover, he has participated as a trainee in several training courses such as programming of microcontrollers, principles & foundations of solar energy, and energy & electronics principles in Jerusalem District Electricity Company (JDECO). In 2015, he started working in the Royal Industrial Trading Company in various maintenance sections. In 2016, he joined the PPU university as a teaching assistant in the Department of Mechanical Eng. / Mechatronics Engineering. During his stay at PPU University, he prepared himself for higher studies. In the year 2017, he was awarded the Turkish government scholarship (YTB) for his master's degree in Mechatronics Engineering from Kocaeli University, Turkey.