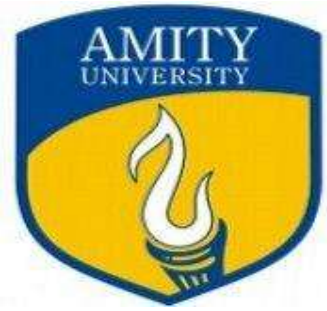
A Project Report

on

Self-Driving Car Simulation Using AI

Submitted to



Amity University Uttar Pradesh

In partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology (CSE)

By

Kuldeep Dwivedi

Under the guidance of

Mrs. Seema Sharma

ASET

Amity School of Engineering and Technology

AMITY UNIVERSITY UTTAR PRADESH

May-June 2022

DECLARATION

We, **Kuldeep Dwivedi** student of B.Tech (CSE) hereby declare that the project titled "**Self-Driving Car Simulation Using AI**" which is submitted by us to **Amity School of Engineering and Technology** Domain of Engineering and Technology, Amity University Uttar Pradesh, in partial fulfilment of requirement for the award of the degree of Bachelor of Technology(CSE), has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition. We hereby declare that I have gone through project guidelines including policy on health and safety, policy on plagiarism etc.

Noida

Date

Kuldeep Dwivedi

CERTIFICATE

On the basis of declaration submitted by **Kuldeep Dwivedi** student of B. Tech (CSE), I hereby certify that the project titled "**Self Driving Car Simulation Using AI**" which is submitted to **Amity School of Engineering and Technology**, Domain of Engineering and Technology, Amity University Uttar Pradesh, in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology (Discipline), is an original contribution with existing knowledge and faithful record of work carried out by them under my guidance and supervision.

To the best of my knowledge this work has not been submitted in part or full for any degree or diploma to this University or elsewhere.

Noida

Date

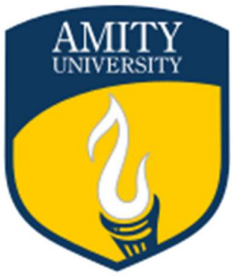
Signature of Guide

(Ms. Seema Sharma)

Amity School of Engineering and Technology

Domain of Engineering and Technology

Amity University Uttar Pradesh



ACKNOWLEDEGEMENT

It is high privilege for us to express our deep sense of gratitude to those entire faculty members who helped us in the completion of the project, specially our internal guide, **Ms. Seema Sharma** who was always there at hour of need.

Our special thanks to all other faculty members, batchmates & seniors of Amity School of Engineering and Technology, Amity University Uttar Pradesh for helping me in the completion of project work and its report submission.

Kuldeep Dwivedi

Table of Contents

Title	Page No.
1. Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Scope	2
1.4 Objective	2
2. Literature Review	3
2.1 Introduction	3
2.2 Goals	3
3. Design and Methodology	11
3.1 Localization	12
3.2 Particle Filter	23
4. Result and Discussion	32
4.1 Localization	32
4.2 Particle Filter	58
5. Conclusion and Future Work	35
References	37

List of Figures

Figure 1 Localization Perception 20
Figure 2 Particle Filter basic idea 21
Figure 3 Phase 1 24
Figure 4 Phase 2 24
Figure 5 Phase 3 24
Figure 6 Phase 4 25
Figure 7 Phase 5 25
Figure 8 Phase 6 26
Figure 9 Phase 7 26
Figure 10 Phase 8 27
Figure 11 Phase 9 27
Figure 12 Phase 10 28
Figure 13 Phase 11 28
Figure 14 Phase 12 29
Figure 15 Phase 13 29
Figure 16 Particle Filter 30
Figure 17 Localization 34

ABSTRACT

"The technology is ahead of the level of governance in many areas," according to the report, which states that "all presume to have a human running the vehicle." "The technology is currently advancing so quickly that it is in danger of outstripping existing standards," according to the report. Such huge concerns have put the research on hold for a long time. With encouraging trials and continued improvement, the driverless automobile, also known as an autonomous car, has become a hotbed of study and expertise.

The goal of this project is to create and optimize a software implementation required for the detection of location i.e. localization, dynamic path finding from a fixed source to fixed destination in a real time scenario and also the necessary controller for the efficient control of an autonomous or the self driving car.

To achieve this goal we will be using several artificial intelligence technique for localization such as Kalman Filters, Histogram Filters and the heuristic approach algorithm A* for dynamic path finding.

Chapter 1

Introduction

1.1 Background

Artificial intelligence research is very technical and specialized, and it is divided into various subfields that frequently fail to communicate with one another. Most of the divide is due to social and cultural factors: subfields have developed around specific institutions and researchers' activity. Several technological difficulties also divide AI research. Some subfields concentrate on solving specific challenges. Others concentrate on one of several alternative approaches, the use of a specific instrument, or the completion of specific tasks.

In this project, we will implement a program for Google's self-driving car by using several artificial intelligence techniques for localization such as Kalman Filters, Histogram Filters and the heuristic approach algorithm A* for dynamic path finding.

The dynamic path finding is very useful as it will give instant path directions based on the current car location. The A* algorithm, which is used to implement path based on the current and final location is very fast in comparison with Dijkstra and BFS. Later we implement PID controller to find the smoothest path to the destination as A* gives only square paths.

Problem Statement

Create and optimize a software implementation for position detection (localization), dynamic path finding from a fixed source to a fixed destination in a real-time scenario, and the appropriate controller for efficient control of an autonomous or self-driving car.

Solution

We will be using several artificial intelligence techniques for localization such as Kalman Filters, Histogram Filters and the heuristic approach algorithm A* for dynamic path finding.

1.2 Motivation

Self-driving cars' very sophisticated technology allows the onboard computer to do hundreds of calculations per second. These include your distance from the objects, your current speed, the conduct of other drivers, and your geographic location. Because the only accidents so far have occurred while human drivers were in control, these ultra-accurate measurements have almost eliminated driving errors for test cars on the road.

Self-driving cars have a tremendous potential for reducing traffic congestion because they are rarely involved in accidents and, these cars can communicate with each another, traffic lights would be obsolete. Better traffic coordination might result in less congestion if people drove slower but made fewer stops.

1.3 Scope

Technological progress is accelerating at an uncontrollable rate. The more a country invests in emerging technology, the better off it will be in the long run. These incentives range from improved infrastructure to increased employment creation. Nowadays, technology's main goal is to reduce human interaction in everyday chores. The big players in this are Machine Learning, Artificial Intelligence, and Computer Vision. We've arrived at a point where we can automate routine chores like driving. Isn't that insane? Autonomous Cars is a frequent term for this type of computer vision. These autonomous vehicles can be aware of their environment and based on their nearby environment these cars can navigate without the assistance of humans.

1.4 Objective

The project's goal is to use path finding algorithms to create a dynamic path for a car so that it can reach its final goal position from starting position. And then we can use a PID controller on the path obtained by the algorithm to find the smoothest path for the car to get there.

Chapter 2

Literature Review

2.1 Introduction

Machine intelligence is known as artificial intelligence (AI). An ideal "intelligent" machine in computer science is a flexible rational agent that senses its surroundings and makes such decisions which increases its probability of achieving an arbitrary goal. When a computer uses cutting-edge technology to competently perform or replicate "cognitive" processes that we intuitively identify with human minds, such as "learning" and "problem solving," the phrase "artificial intelligence" is most probably to be used. Artificial intelligence has a negative connotation, especially among the general public, because it is associated with "cutting-edge" machines (or even "mysterious"). This subjective barrier between "artificial intelligence" and "human intellect" seems to blur with time; for example optical character recognition, is no longer seen as an exemplar of "artificial intelligence," but rather as a banal regular technology. Computers that can beat elite players at chess and go, as well as self-driving automobiles that traverse packed city streets, are modern instances of AI.

Reasoning, knowledge, planning, learning, NLP(communication), perception, and the capability to move and control objects are all fundamental goals in AI research. General intelligence is one of the field's long-term objectives. Statistical methodologies, computational intelligence, and conventional symbolic AI are all currently popular approaches. Artificial intelligence employs a wide range of techniques, including search and mathematical optimization, logic, probability, and economics-based strategies. Artificial intelligence is an interdisciplinary field that includes computer science, mathematics, psychology, linguistics, philosophy, and neuroscience, as well as more specialized fields like artificial psychology..

2.2 Goals

The difficulty of replicating (or creating) intelligence has been dissected into sub-problems. Researchers are looking for specific features or abilities in an intelligent system. The following features have gotten the most attention.

2.2.1 Problem solving, Reasoning and Deduction

Previous Artificial Intelligence research produced algorithms that mimicked humans' step-by-step reasoning while solving puzzles or making logical conclusions. AI research from 1980s to 1990s had created extremely effective methods for managing uncertain or partial information based on probability and economics ideas.

Most of these systems involve a "combinatorial explosion" when the problem size surpasses a certain threshold, in which the memory usage used grows drastically. Researchers in Artificial Intelligence are focusing on developing more efficient problem-solving systems.

Humans solve the majority of issues by drawing rapid, instinctive conclusions rather than the laborious, sequential calculations that early AI research could imitate. Embodied agent methods highlight the role of sensorimotor skills in higher reasoning; Research based on neural net aims to recreate the brain mechanisms that contribute9 to this ability; statistical AI techniques imitate the probabilistic character of human guesses.

2.2.2 Knowledge representation

An ontology is a collection of ideas within a field and their interconnections that represents knowledge. AI research revolves around knowledge representation and engineering. Many of the issues that robots are supposed to address will need a detailed comprehension of the world. Among the things AI must represent are objects, qualities, categories, and interactions between objects; situations, events, states, and time; causes and consequences; knowledge about knowledge; and many other, less well-studied areas. The collection of things, connections, theories, and additional things that the machine is conscious of is represented by an ontology. The most broad ontologies try to offer a base for all other information.

2.2.3 Planning

A hierarchical control system is one that uses a hierarchy of devices and software to make decisions. A hierarchical control system makes decisions using a combination of devices. Intelligent agents should be able to set and achieve objectives. They require a method for considering the future and making actions that enhance the efficiency of the

alternatives accessible to them. In traditional planning issues, the agent might assume that is the only object in the environment and that the outcomes of its actions are easy to predict.

2.2.4 Learning

Machine learning, which can be defined as study of self-improving computer systems, has always been at the centre of AI research. The capacity to find patterns in a stream of data is known as unsupervised learning. Both classification and numerical regression are part of supervised learning. Classification is used to establish where something belongs after looking at multiple instances of items from various categories. The process of building a model that describes the relation between inputs and outputs and estimates how the outcomes should vary as the inputs change is known as regression. The agent is rewarded for favourable replies and penalised for poor ones in reinforcement learning. The agent devises a strategy for working in its issue based on this collection of rewards and punishments.

2.2.5 Natural language processing (communication)

Natural language processing allows machines to comprehend and interpret human speech. Natural language user interfaces and direct knowledge acquisition from human-written sources for e.g. newswire texts or editorials may be possible with a sufficiently capable natural language processing system. Some of the most common application of NLP are Feature extraction (or text mining), question answering, and translation.

Semantic indexing is the study of natural language and the extraction of meaning from it. Indexing vast numbers of representations of the user's input is becoming significantly more efficient as processor speeds have grown and data storage costs have decreased.

2.2.6 Perception

Machine perception is the capability to derive features of the world using sensor input (cameras, microphones, tactile sensors, sonar, and other more exotic devices). The capacity to analyze visual information is known as computer vision. Speech recognition, facial recognition, and object recognition are a few examples of subproblems.

2.2.7 Motion and manipulation

Robotics and AI are closely connected topics. For robots to perform tasks like object manipulation and navigation, intelligence is required, with sub-problems like localization (finding where things are), mapping (knowing the environment and what is around), and motion planning (finding out what to do next) or path planning (shifting from one location in space to another, which may require complex moves).

2.2.9 Social Intelligence

Affective computing is the research and development of systems and devices that can recognise, understand, process, and reproduce human sentiments. This interdisciplinary field includes computer science, psychology, and cognitive science. While the field's origins can be traced back to early philosophical examinations of emotion, Rosalind Picard's 1995 paper on emotional computing helped to establish the current branch of computer science. One of the research's motivations is the ability to reproduce empathy. The system should be able to detect people's emotions and adjust its behaviour accordingly, providing appropriate responses.

The emotional and social abilities of an intelligent agent serve two functions. To begin, AI must understand people's motivations and emotional states in order to forecast their actions. (Game theory, decision theory, the capability to model human emotions, and perceptual skills for identifying emotions are all examples.) Furthermore, an intelligent machine may wish to be able to exhibit emotions—even if it does not experience them—in order to appear responsive to the emotional dynamics of human contact in order to enhance human-computer connection.

2.3 Tools used to implement AI

AI has generated a significant number of tools to handle the most difficult issues in computer science over the course of 50 years of research. Some of the most common techniques are listed here.

2.3.1 Search and Optimization

Many AI issues can technically be resolved by considering a vast set of possibilities: A search could be used to express logic. For example, logical proof may be thought of as

going down the path from arguments to results, each stage needing the application of an inference rule. To discover a path to a target goal, planning algorithms employ means-ends analysis to search across branches of objectives and sub goals. Local searches in coordinate space are used in robotics algorithms to move limbs and grab items. Many learning algorithms employ optimization-based search methodologies.

For most real-world issues, simple thorough searches are rarely sufficient: the search space (the number of places to seek) soon increases to astronomical proportions. As an outcome, the process either takes too long or never ends. "Heuristics" or "rules of thumb" are frequently employed to rule out choices that really are unlikely to produce to the desired outcome (called "pruning the search tree"). Heuristics provide the software with a "best guess" of the solution's location. The total sample of the solution search is reduced using heuristics.

In the 1990s, a new type of search emerged, one based on the mathematical theory of optimization. In many cases, you may start with an estimate and steadily refine it until there are no more refinements available. We start our search at a random location on the terrain and then move our guess uphill utilising leaps until we reach the peak, equivalent to blind hill climbing. Simulated annealing, beam search, and random optimization are some of the other optimization techniques.

In evolutionary computation, optimization search is used. They may start with a population of organisms (guesses) and let them evolve and integrate over time, with only the fittest surviving each iteration. Evolutionary computation includes swarm intelligence approaches and evolutionary programming.

2.3.2 Logic

Logic is commonly used to describe and solve problems, but it may also be used to solve other difficulties. The satplan method, for example, uses logic to plan, whereas inductive logic programming is used to train.

Various kinds of logic are employed in AI research. The logic of propositions that can be true or untrue is known as propositional or sentential logic. Quantifiers and predicates are also supported by first-order logic, allowing the declaration of facts about things, their attributes, and their connections. In fuzzy logic, rather than assigning True

values as 1 or False values as 1, the truth is assigned a number between 0 and 1. Fuzzy systems are commonly used in today's industrial and consumer product control systems for uncertain reasoning. Subjective logic explains uncertainty in a different and more straightforward way than fuzzy logic: a given binomial opinion meets belief + disbelief + uncertainty = 1 inside a Beta distribution. This technique distinguishes ignorance from an agent's very confident probabilistic claims.

Default logics, non-monotonic logics, and circumscription are examples of logics that aid with default reasoning and the qualifying problem. Description logics, situation calculus, event calculus, and fluent calculus (for capturing events and time), causal calculus, belief calculus, and modal logics are some of the extensions of logic that have been suggested to handle particular types of knowledge.

2.3.3 Probabilistic methods for uncertain reasoning

Various AI challenges need the agent's ability to deal with ambiguous or incomplete data. Using concepts of probability theory and economics, AI researchers have built a number of sophisticated tools to handle these problems.

Bayesian networks can assist in reasoning, learning, planning, and perceiving. By filtering, forecasting, smoothing, and providing explanations for streams of data, probabilistic algorithms can assist perception systems in analysing methods that occur across time.

"Utility" is a term used in economics to describe how useful something is to a rational agent. Using decision theory, decision analysis, and information value theory, specific mathematical tools have been constructed to examine how an agent creates decisions and plans. Some of the methods that can be used are Markov decision processes, dynamic decision networks, game theory, and mechanism design.

2.3.4 Classifiers and statistical learning methods

Two of the most fundamental types of Artificial Intelligence applications are classifiers and controllers. Many AI systems, however, rely on classification since controllers must first detect conditions before inferring actions. Classifiers are algorithms that compare patterns to find the most similar ones. They're especially attractive for AI because they can be tweaked depending on instances. Observations or patterns are the

terms used to characterise these occurrences. Each pattern in supervised learning corresponds to a specific class. A class could be thought of as a decision that needs to be made. A data set is made up of all of the observations and their class designations. A new observation is categorised based on past information when it is obtained.

Statistics and machine learning are two approaches that a classifier can use to learn. The most often used classifiers are the neural network, kernel approaches such as the support vector machine, k-nearest neighbour algorithm, Gaussian mixture model, naive Bayes classifier, and decision tree. The performance of these classifiers has been compared across a variety of tasks. The characteristics of the data to be categorised have a significant impact on the performance of a classifier. According to the "no free lunch" theorem, no single categorization strategy is optimal in all circumstances. Choosing the right classifier for the job appears to be more of an art than a science.

2.3.5 Neural networks

A neural network is a set of nodes that resembles the enormous network of neurons that makes up the human brain. A decade before the area of AI research was created, Walter Pitts and Warren McCullough invented the concept of non-learning neural networks. Frank Rosenblatt created the perceptron, which works similarly to linear regression.

Acyclic or feedforward neural networks (in which the signal travels only one direction) and recurrent neural networks are the two most prevalent types of neural networks. Perceptrons, multi-layer perceptrons, and radial basis networks are the three types of feedforward networks most commonly used.

Backpropagation is a method for training neural networks that was first developed as a reverse mode of automated differentiation and later applied to neural networks. Hierarchical temporal memory is a method for simulating some of the neocortex's structural and algorithmic qualities.

Chapter 3

Design and Methodology

3.1 Localization

In order to drive the car must first we have to locate it. However it is not quite possible to measure a car location using an instrument. Even the GPS have margin of error of around 5 m. Now, think about this error on a standard driving lane whose width is 3.5 m. So, to solve this problem a method called Localization is used.

Localization is the ability for a machine to locate itself in space or we can also say which is a program that can measure location indirectly. Rather than install a GPS device in our robot, we are going to write a program to implement localization which will locate position of our car and also reduce the margin of error caused by GPS earlier and will make it between 2cm to 10cm. This high level of accuracy enables a self-driving car to understand its surroundings and establish a clear image of road and its lanes. With this a car can even detect diverging and merging of a lane, plan lane changes and determine lane paths even when markings aren't clear.

Imagine a car is lost in a 1D world where it can only move in forward and backward direction, sideways motion is not possible. Since our car is unaware of its position, it believes that every point in this one dimensional world is equally likely to be its current position. It can be described mathematically by saying that car's probability function is uniform over the sample space which is 1D.

Currently, the majority of self-driving cars being developed by automakers and tech companies do not include probability in their AI systems. We recognize that this may come out as a surprising statement. Given that human driving entails constantly assessing and modifying probability as well as coping with uncertainty, you'd think that self-driving car AI would do the same.

Then we will draw a graph of this probability function with probability on the y-axis and location on the x-axis, we would draw a straight, level line. This line describes a uniform probability function, and it represents the state of maximum confusion.

Now, assume there are three landmarks on the road which is 1D, and assume these landmarks as bumps on the road. Since the robot has just sensed that it is near a door,

it assigns these locations greater probability (indicated by the bumps in the graph) whereas, all of the other locations have decreased belief. This function represents another probability distribution which is the best representation of car's current position relative to the bump on the road.

3.1.1 Robot Movement

Suppose car moves to the right a certain distance, then we can shift the location accordingly and all the bumps also shifts to the right as expected but not perfectly as some of the bumps are also flattened. This flattening occurs due to the uncertainty in car motion because the car doesn't know exactly how far it has moved, its knowledge became less accurate and so the bumps have become less sharp.

Here, we perform convolution by shifting and flattening these bumps. Convolution is a mathematical operation that takes two functions and measures their overlap or basically we can say which is the measure of overlap of two functions over one another. For example, if two functions have zero overlap, the value of their convolution will be equal to zero. If they overlap completely, their convolution will be equal to one. As we slide between these two extreme cases, the convolution will take on values between zero and one. In our convolution, the first function represents the location function, and the distance moved is represented by the second function.

Now, again car reaches to another bump and senses itself right next to it so, again the measurement is same as before. Just like after our first measurement, the sensing of a bump will increase our probability function by a certain factor everywhere where it found a bump. So, should we get the same location that we had after our first measurement? No! Because unlike with our first measurement, when we were in a state of maximum uncertainty, this time we have some idea of our location prior to sensing. This prior information, together with the second sensing of a bump, combine to give us a new probability distribution.

Localization, specifically the Monte Carlo localization method that we are using here, is nothing but the repetition of *sensing* and *moving*. There is an initial belief that is tossed into the loop. If you sense first it comes to the left side. Localization cycles through move and sense. Every time it moves it loses information and every time it

senses it gains information. Measure of information in a distribution is called as Entropy.

3.1.2 Bayes' Rule

Suppose we are driving a car, and a situation came where we have to apply brakes. Do we considering the possibility that the car ahead of you will slam on their brakes next? Are we evaluating the chances that the automobile in front of we will be able to make a timely stop? Did the possibility of colliding with the automobile ahead of us and the car behind us cross our mind? Most folks aren't making those kinds of probability calculations in their heads. Instead, they've learnt to make probabilistic judgments over time. On other days, we may choose to disregard some of those possibilities and increase your risks. On other days, you become more aware of the chances and drive with extreme caution.

As we are used to the idea of probability, most self-driving cars now being developed by automakers and tech companies do not yet have probability embedded in their AI systems. We recognise that this may come out as a surprising statement. Given that human driving entails constantly assessing and modifying probability as well as coping with uncertainty, you'd think that self-driving car AI would do the same.

Uber has created Pyro (programming language). The Uber AI Labs is attempting to establish Pyro as the programming language of choice for anyone working with AI who also needs to deal with probability.

We have the frequentists, who look at the universe in a rather generic way based on long-term frequencies of recurring events, and the conditional probability camp, which believes that the nature of your probabilities is determined by the scenario. We'll look at best well-known parts of the conditional probability camp, the Bayesian view of probability, in a moment.

Bayes' Rule is used for updating a probability distribution after making a measurement.

- x = grid cell
- Z = measurement

The equation for Bayes' Rule looks like this:

$$P(x \vee Z) = \frac{P(Z \vee x)P(x)}{P(Z)}$$

The left hand side of this equation should be read as "The probability of X after observing Z." In probability, we often want to update our probability distribution after making a measurement. Sometimes, this isn't easy to do directly.

Bayes' Rule tells us that this probability (the left-hand side of the equation) is equal to another probability (the right-hand side of the equation), which is often easier to calculate. In those situations, we use Bayes' Rule to rephrase the problem in a solvable way.

To use Bayes' Rule, we first calculate the non-normalized probability distribution, which is given by $P(Z|X)P(X)$, and then divide that by the total probability of making measurement Z, P(Z), which is called the normalizer.

3.1.3 Theorem of total probability

Now let's look at motion, which will emerge out to be something we will call total probability. Hopefully you remember caring about a grid cell "xi" and we asked what is the chance of being in xi after robot motion. Now, we will use a time index to indicate after and before motion:

Compute this by looking at all the grid cells the robot could have come from one time step earlier (at time t-1), and index those cells with the letter j, which in our example ranges from 1 to 5. For each of those cells, look at the prior probability $P(X_j^{t-1})$, and multiply it with the probability of moving from cell X_j to cell X_i, $P(X_i|X_j)$. This gives us the probability that we will be in cell X_i at time t:

- $P(X_i^t) = \Sigma_j P(X_j^{t-1})P(X_i|X_j)$

You can see the correspondence of *A* as a place *i* of time *t* and all the different *Bs* as the possible prior locations. This is often called the Theorem of Total Probability.

3.1.4 Perception

Once a car knows where it is, its next job is to perceive and track objects it might collide with (for example other cars or pedastrians). This problem has many similarities to

localization. The sensors of other objects are noisy and we want to predict where they are. Again, the underlying model is a modification of a hidden markov model to allow for continuous variables. The picture below shows a variable representation. At each time point there are two variables.

However, though a robot represents localizing itself and perceiving others with very similar underlying models, the two problems are solved using different algorithms. Instead of using a particle filter to estimate position, perceiving other objects is solved with kalman filters. Kalman filters make an additional assumption about the variables that they are tracking. The algorithm assumes that the location variables are gaussian. This assumption simplifies the problem into one where the solution to where the other cars are can be computed exactly (and thus much faster).

Localization can be represented as the following iteration of sense, move and initial belief.

"It first sets an initial belief senses with the measurements and then moves."

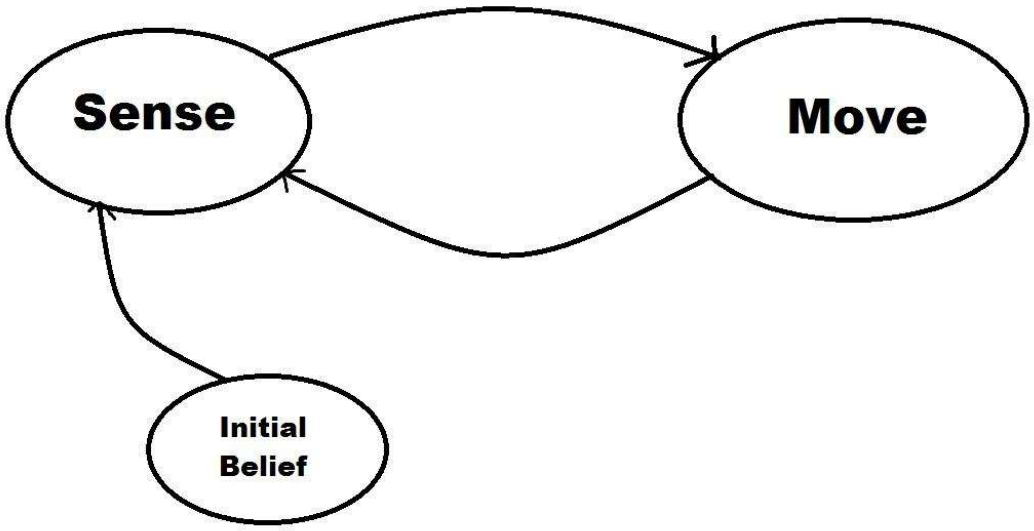


Figure 1 Localization Perception

3.3 Particle Filter

Particle filters have recently been used to overcome a number of difficult perceptual challenges in robotics. Particle filters' early accomplishments were limited to low-dimensional estimation problems, such as robot localisation in known-map situations.

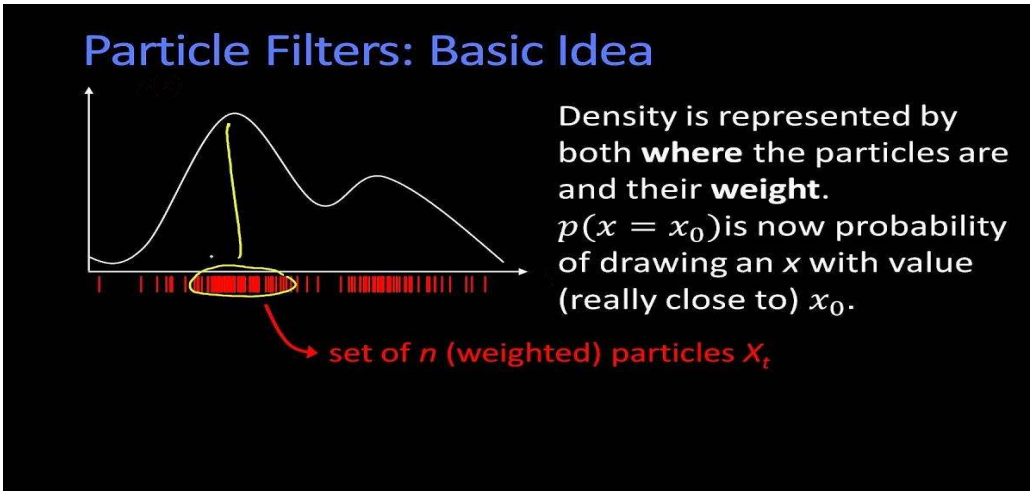


Figure 2 Particle Filter basic idea

When dealing with efficiency, the decision is still out. In some cases, particle filters scale exponentially, thus representing particle filters in more than four dimensions would be a mistake. However, they scale far better in case of tracking domains.

Range sensors are represented by the blue stripes on the robot, which is positioned in the upper right hand corner of the environment. The sensors gauge the distance of nearby barriers using sonar sensors, which are sound sensors. These sensors aid the robot in calculating an appropriate posterior distribution for its location. What the robot doesn't realise is that it's starting in the middle of a corridor and has no idea where it's going.

Goal of particle filters is to have the particles guess where the robot is moving while still allowing them to live, akin to "survival of the fittest," with particles that are more consistent with the measurements having a better chance of surviving. As a result, high-probability locations will accumulate more particles, making them more indicative of the robot's posterior beliefs. The robot's approximate belief as it localises itself is made up of the particles.

The versatility of particle filter technology is due to its supremacy in nonlinear and non-Gaussian systems. Furthermore, the particle filter's multi-modal processing capabilities is one of the reasons for its widespread use. Particle filtering has been used in a variety of fields around the world. Particle filters are currently commonly used in the estimate of financial market models, especially stochastic volatility models.

When the model is nonlinear and the noises are not Gaussian, particle filters methods are recursive Bayesian filters that give a practical and appealing solution to approximate the posterior distributions. These methods provide broad solutions to a wide range of issues where linearization and Gaussian approximations are either intractable or produce unacceptable results. Non-Gaussian noise assumptions and the integration of state variable limitations can also be done naturally. Furthermore, particle filter methods are extremely adaptable, simple to construct, parallelizable, and useful in a wide range of situations.

Researchers have lately begun to take advantage of structural features of robotic domains, resulting in practical particle filter applications in areas with up to 100,000 dimensions. Because no model, no matter how comprehensive, can capture the complete complexity of even the most basic robotic settings, specific methods and procedures are required for particle filter performance in robotic domains. This section discusses some of these recent developments and gives links to more detailed publications on particle filters in robotics.

The more broader case of (almost) unrestrained Markov chains is addressed by particle filters. The primary concept is to approximate the posterior of a set of sample states x that correspond to Xi, or particles. Each of these is a concrete state sample of index I where i's range represents the particle filter's size.

The most basic version of particle filters is given by the following algorithm.

- Initialization: At time t=0, draw M particles according to p(x0).

Call this set of particles X0

- Recursion: At time t>0 , generate a new particle for each already existing particle by drawing from the actuation model . Call the resulting set XT

Subsequently, draw M particles from Xi , so that each particle in X is drawn (with replacement) with a probability proportional to $p(zi \mid xi)$.

Call the resulting set of particles Xt.

Particle filters appeal to robots for a variety of reasons. To begin with, they may be used to simulate practically any probabilistic robot model that can be expressed as a Markov chain. Furthermore, particle filters can be used at any time and do not require a predetermined computing period; instead, their accuracy rises as computational resources become available. This makes them appealing to robotics, which frequently deals with stringent real-time limitations that must be met with difficult-to-control computer technology.

Finally, they are relatively simple to put into practise. The implementer does not need to linearize non-linear models or care about closed-form solutions of conditional statements of multiple forms, as in Kalman filters. The fundamental objection levelled by particle filters is that, in general, populating a d-dimensional space necessitates an increasingly large number of particles in d.

As a result, the majority of effective applications have been l3imited to low-dimensional state spaces. Structure (for example, conditional independences), which is used in many robotics challenges, has only lately led to applications in higher dimensional environments.

3.3.1 Working of Particle Filter Visualization

Our car only drives in one direction for the purposes of debate. The GPS is used to locate the car's original location. We sample 10 locations (particles) based on the measurement noise parameter provided by the GPS manufacturer because GPS is not reliable. The amount of particles utilised is adjustable, and we chose a low number only to make things easier to see. In practise, we can start with hundreds to thousands of particles and experiment to find the right amount that strikes a reasonable balance between precision and processing cost.

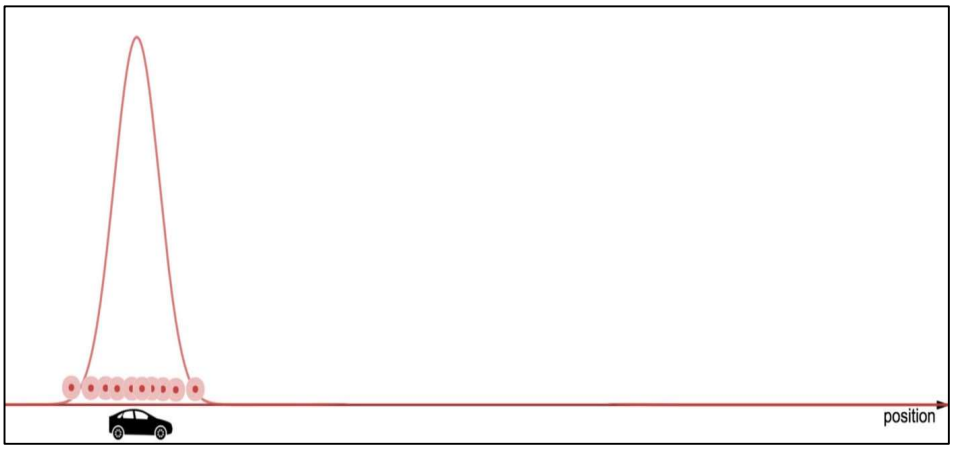


Figure 3 Phase 1

Now suppose that there is no GPS then we can distribute the particle uniformly along the track. This is the same as we did in Localization i.e we don't know the position of the car.

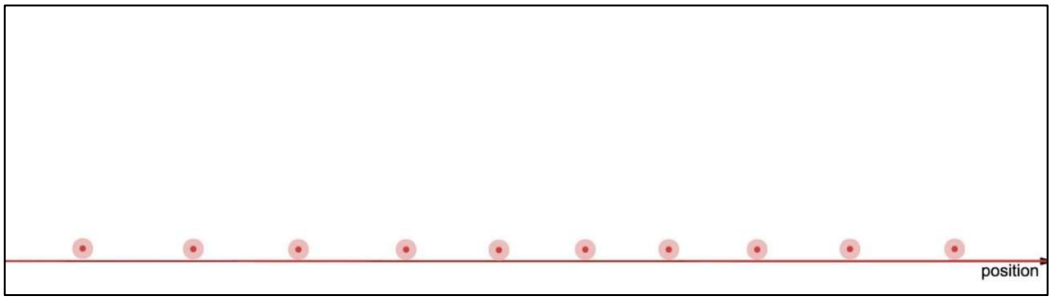


Figure 4 Phase 2

For each particle, we apply a dynamic model (like x' = x + v δt) to calculate the next position at time t + δt.

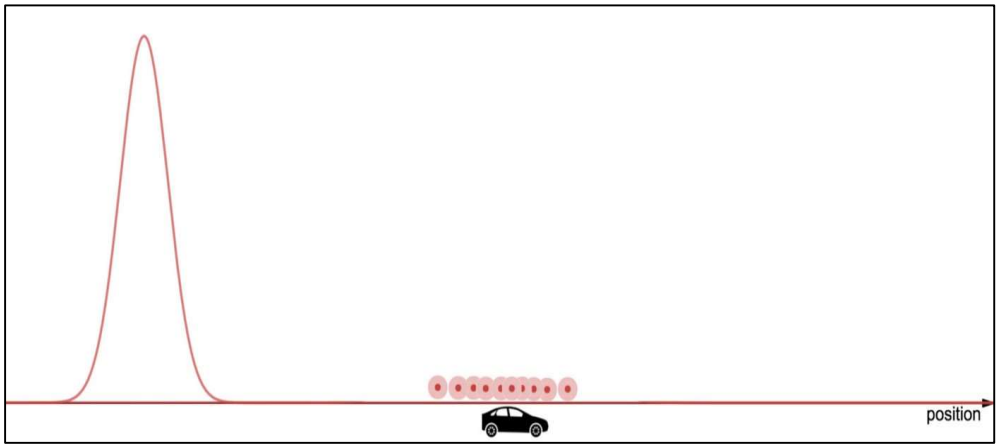


Figure 5 Phase 3

We add separate random noise to each particle's location to represent process noise produced by factors such as weather and road conditions. Experimental data or an analytical model are used to determine the level of the noise. Because of the uncertainty effects, the particles should spread wider apart.

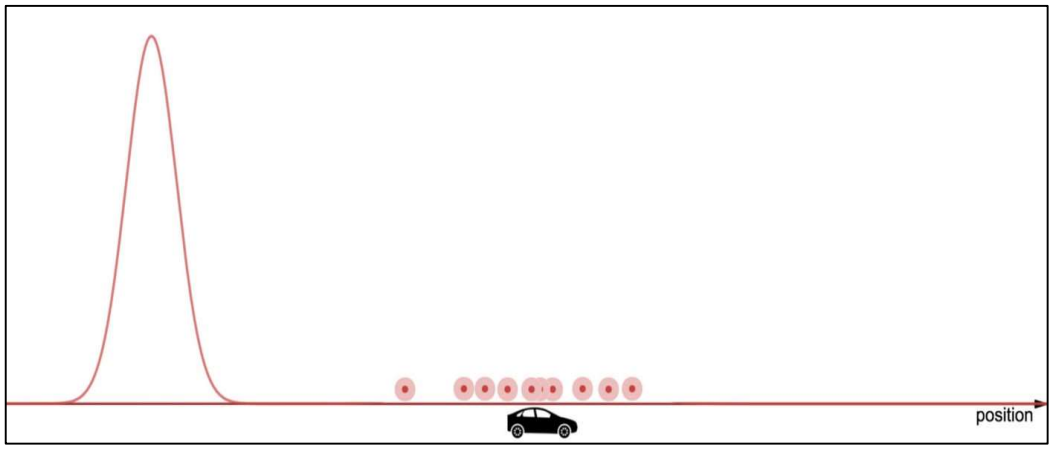


Figure 6 Phase 4

Now we will find all of the surrounding landmarks using the per-defined map

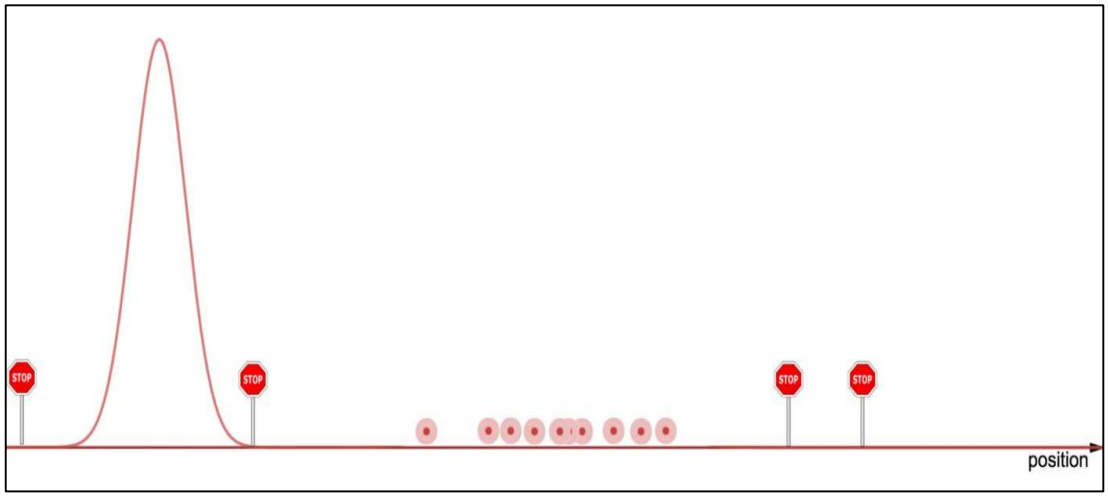


Figure 7 Phase 5

Now, we remove the landmarks that are out of sight (left most land mark in this case)

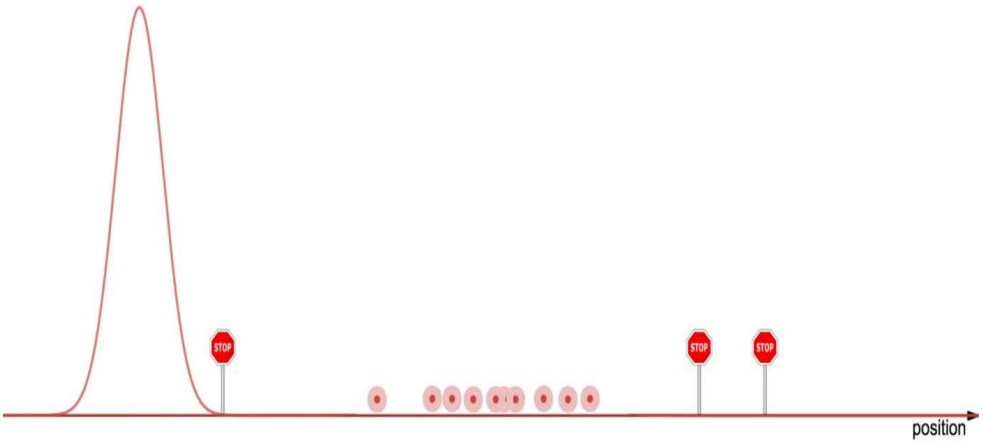


Figure 8 Phase 6

Each particle indicates the car's possible location. We want to compute the probability utilising data from our sensors. A distance and an orientation from a landmark are included in each reading. However, as previously stated, the measurement does not identify the landmark. Instead, We utilize the distance and orientation information to locate the closest landmark on a map. In P2, for example, we use our first sensor reading to select the right-most stop sign.

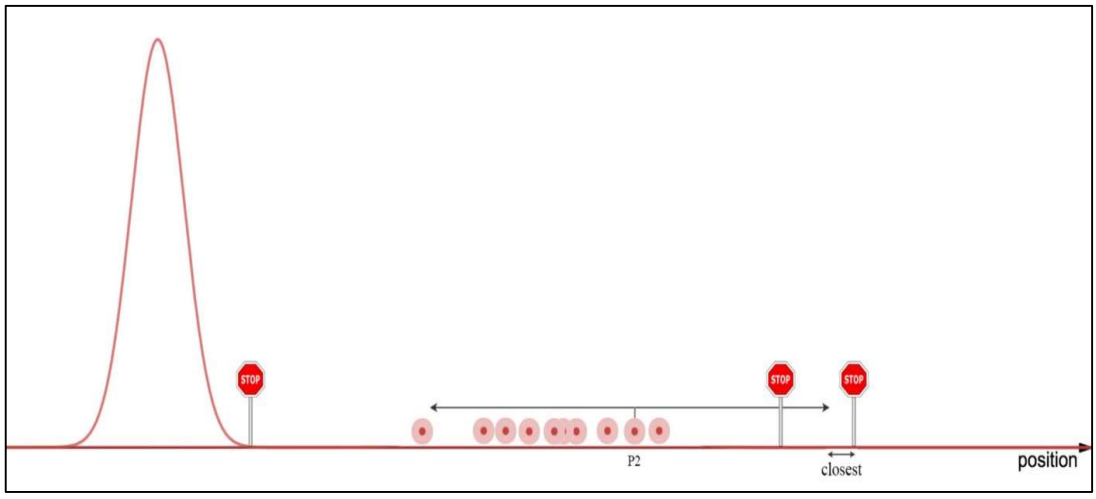


Figure 9 Phase 7

Now, we assume that the measurement noise is Gaussian Distributed, so we find out the probability of P2 representing our current location

$$PDF = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

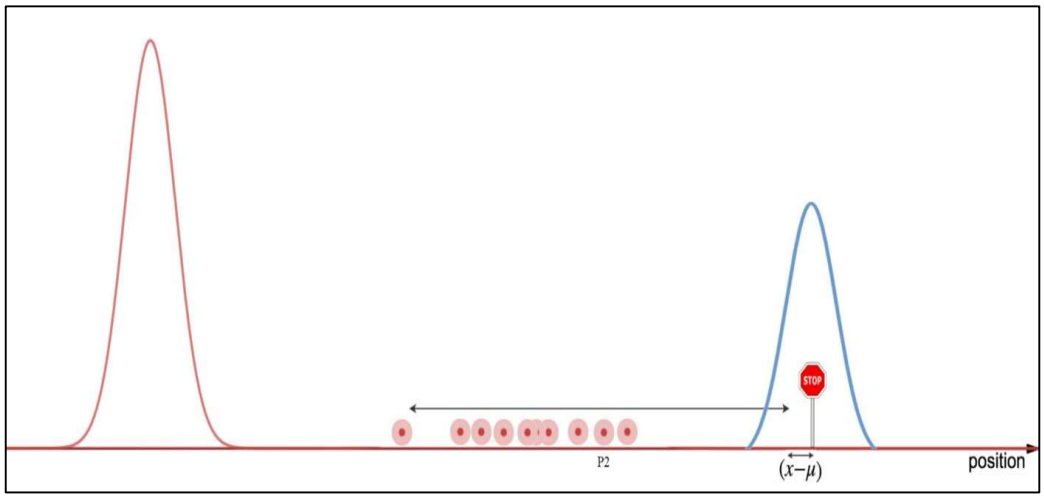


Figure 10 Phase 8

Now, we have to once again perform the calculations for every received measurement. Then by multiplication of all the PDFs, we get the final most probable position of the particle/car .

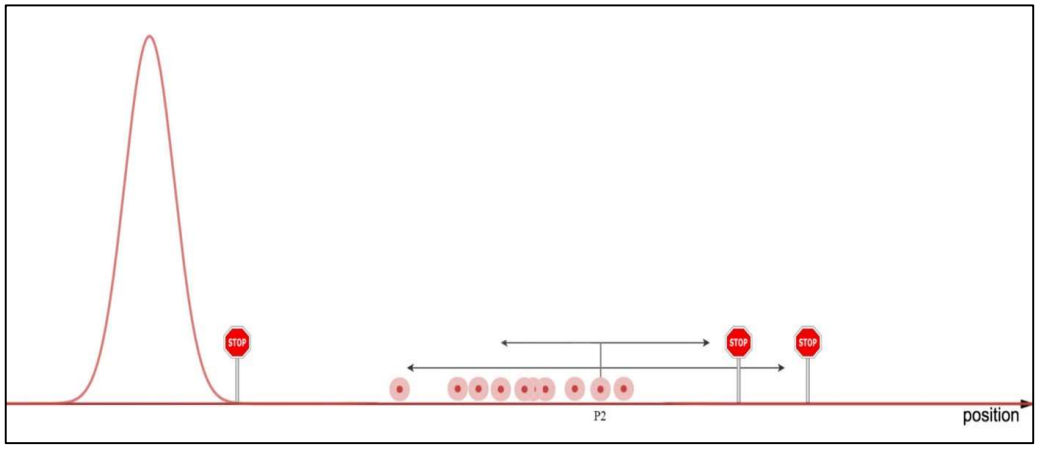


Figure 11 Phase 9

For each particle, we repeat the procedure. After that, we normalise their probabilities till the entire probability equals one. Our particles' weight is determined by this. Our automobile is estimated to be in the particle with the highest weight.

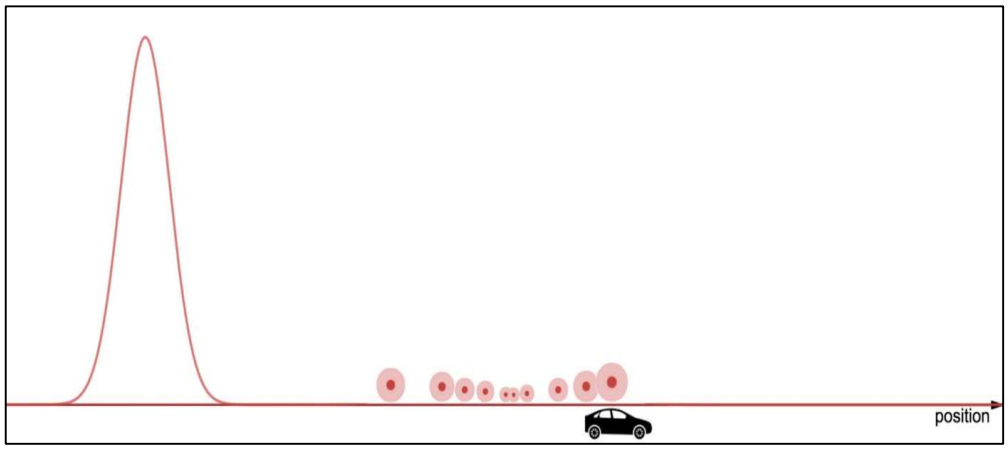


Figure 12 Phase 10

Based on the weights, we resample another 10 particles. If P2 has double the weight of P1, it will have twice the chance of being chosen. We have two clusters of places in our situation. This means that our measurements indicate that our car might be parked in two different places. This kind of ambiguity can be reduced by increasing the number of landmarks in the map or reduce the time (δt) between predictions.

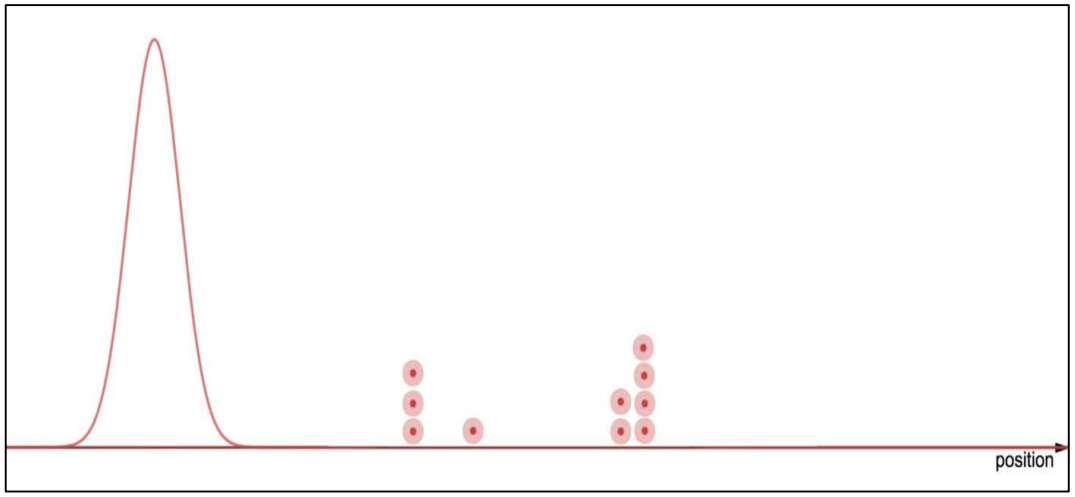


Figure 13 Phase 11

Now, for each particle we have to apply our car's dynamic model again for getting the next position and then we have to add random process noise.

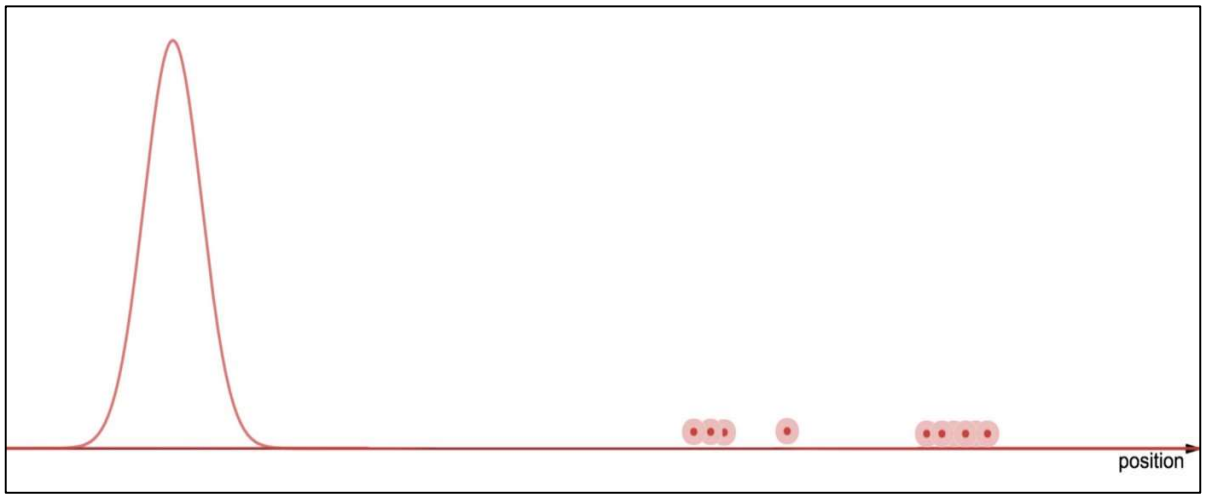


Figure 14 Phase 12

The process is then repeated with a new set of measurements. Our sensor readings are less unclear this time. The particles from the original cluster have been removed, and the fresh ten particles have formed a tight cluster.

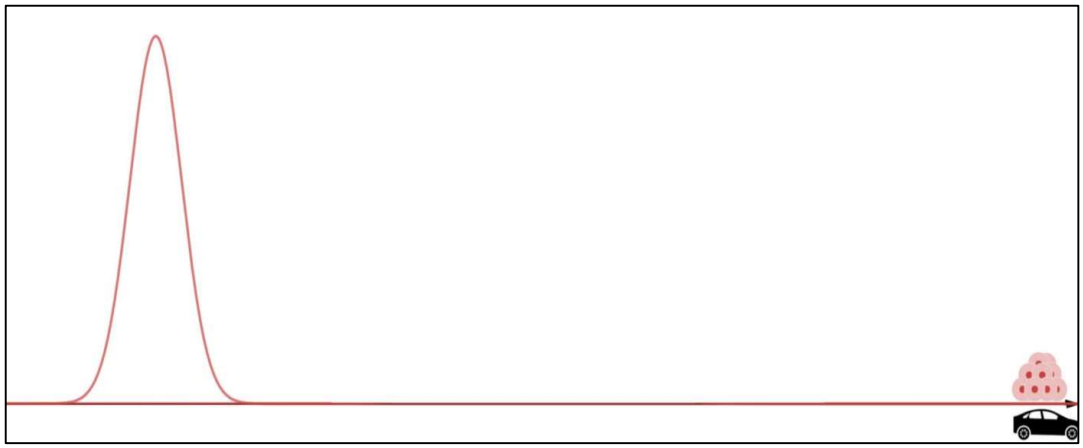


Figure 15 Phase 13

3.3.2 Particle Filter in low dimension space

The 'classical' successful application of particle filters in robotics is mobile robot localization. The problem of estimating a mobile robot's pose relative to a given map using sensor measurements and commands is addressed by mobile robot localization. To define the pose a two-dimensional Cartesian coordinate and the robot's rotational heading direction are used generally.

If the inaccuracy can be assured to be modest at all times, the problem is known as position tracking. The global localization problem, which is the difficulty of locating a

robot in the face of global uncertainty, is more usual. The kidnapped robot problem, in which a well-localized robot is teleported to another place without being told, is the most challenging variant of the localization problem. This issue was raised, for example, during the Robocup soccer competition, in which judges randomly selected up robots and placed them somewhere else. Multiple robots that can observe each other are used in various localization difficulties.

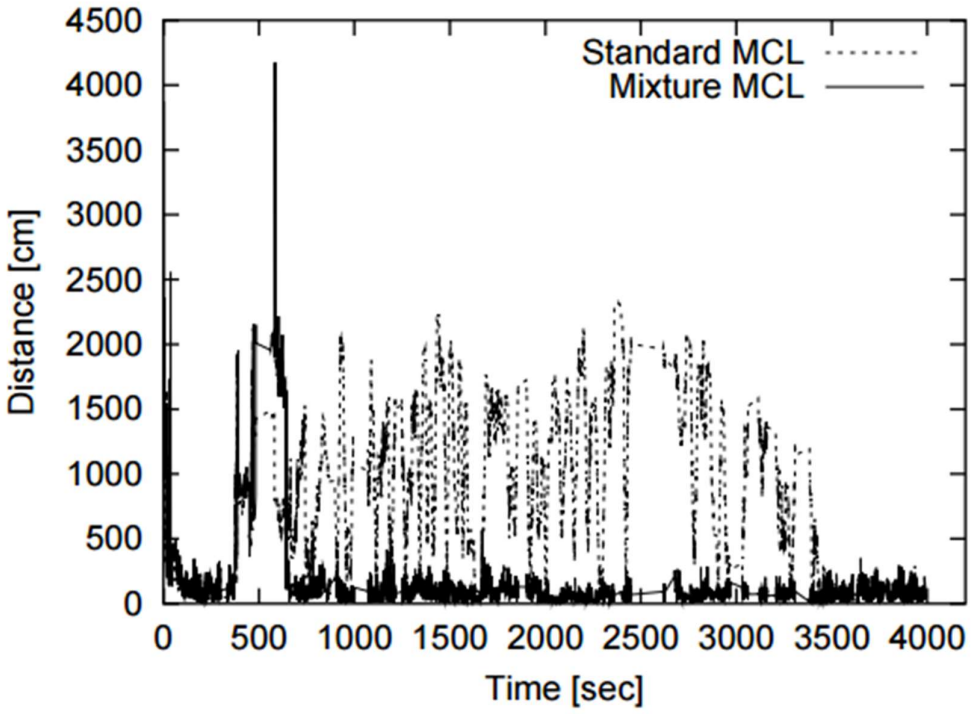


Figure 16 Particle Filter

Figure illustrates particle filters in the context of global localization of a robot in a known environment. At various stages of robot activity, a number of particles approximate the posterior (1), as seen in a development of three instances. Each particle is a representation of a three-dimensional pose variable that includes the robot's Cartesian coordinates as well as its orientation in relation to the map. The pictures in Figure 1 show how the particle filter approximation has evolved over time, from worldwide uncertainty to a well-localized robot.

In the domain of localization, particle filters are also known as Monte Carlo localization (MCL). MCL was inspired by the condensation method, a particle filter that was widely used in computer vision applications at the time. Particle filters routinely outperform alternative strategies in most variations of the mobile localization issue, including

parametric probabilistic techniques like the Kalman filter and more classical techniques.

3.3.3 Particle Filter in High Dimensional Spaces

The poor performance of plain particle filters in higher dimensional spaces is an often cited drawback. This is due to the fact that the number of particles required to populate a state space rises exponentially with its dimension, similar to the scaling constraints of traditional HMMs. Many robotics challenges, on another side, have structure that can be used to design more efficient particle filters. The simultaneous localization and mapping problem is an eg of such a problem. The difficulty of creating a map of the environment using a moving robot is addressed by SLAM. The SLAM problem is difficult to solve because faults in the robot's localization produce systematic problems in the map's environmental feature localization. The lack of an initial map in the SLAM problem makes employing techniques like MCL to locate the robot during mapping difficult. Furthermore, the robot must determine if two environment features detected at separate times relate to the same physical feature in the environment, which is a difficult data association problem. To make problems worse, a map's space can have hundreds of thousands of dimensions. Because the size of the state space is frequently unknown at the start of mapping, SLAM methods must also estimate the problem's dimensionality. Furthermore, most SLAM applications necessitate real-time processing.

3.3.4 Creating a particle

The particle filter you are going to program maintains a set of 1,000 ($N = 1000$) random guesses as to where the robot might be, represented by a dot. Each dot is a vector that contains an x-coordinate (38.2), a y-coordinate (12.4), and heading direction (0.18). The heading direction is the angle (in radians) the robot points relative to the x-axis; so as this robot moves forward it will move slightly upwards.

In your code, every time you call the function robot() and assign it to a particle p[i], the elements p[i].x, p[i].y, p[i].orientation (which is the same as heading) are initialized at random.

In order to make a particle set of 1,000 particles, you have to program a separate piece of code that assigns 1,000 of those to a list.

```
N = 1000

p = []

for i in range(N):

        x = robot()

        p.append(x)

print len(p)
```

3.3.5 Second half of Particle Filter

The second half of particle filters works like this: imagine a robot that sits between four locations and can measure their exact distances. The right-hand figure shows the robot's location and the distances it measures, as well as "measurement noise." Which is modelled as a Gaussian with a mean of zero. This means that there's a chance the measurement will be too short or too long, and that likelihood is determined by a Gaussian distribution.

We now have a measurement vector made up of the four values of the four distances between L1 and L4. The situation described below occurs when a particle hypothesises that its coordinates are somewhere other than where the robot actually is (the red robot represents the particle hypothesis).

A different going direction is also hypothesised by the particle. You can use our robot's measurement vector and apply it to the particle.

However, this ends up being a very poor particle measuring vector. If the red particle was a good match for the robot's real location, the green represents the measurement vector we would have anticipated.

The more likely the set of measurements given that position will be, the closer your particle is to the correct place. The trick to particle filters is that the difference between the actual and projected measurements results in an importance weight, which informs you how essential that individual particle is. The more significant the weight, the more significant it is.

When you have a group of particles, each one has its own weight; some appear to be highly believable, while others appear to be quite implausible, as indicated by the particle's size.

The particles will then be allowed to survive at random, but their chances of surviving will be proportional to their weights. That is, a particle with a higher mass will survive in greater numbers than a particle with a lower mass. This means that after re-sampling, which involves drawing new particles at random from old ones and replacing them in proportion to their importance weight, the particles with a greater importance weight will survive while the smaller ones would perish. This selection method's "with replacement" feature is critical because it allows us to choose the high probability particles several times. As a result, the particles tend to congregate in areas with a high posterior probability.

From here, you'll want to develop a mechanism for calculating importance weights based on the likelihood of a measurement, as well as a resampling approach that collects particles in proportion to those weights.

Chapter 4

Results and Discussion

4.1 Localization

Localization involves a robot continuously updating its belief about its location over all possible locations. Stated mathematically, we could say the robot is constantly updating its *probability distribution* over the *sample space*. For a real self-driving car, this means that all locations on the road are assigned a probability. If the AI is doing its job properly, this *probability distribution* should have two qualities:

1. It should have one sharp peak. This indicates that the car has a very specific belief about its current location.
2. The peak should be correct! If this weren't true, the car would believe—very strongly—that it was in the wrong location. This would be bad for Stanley.

Our Monte Carlo localization procedure can be written as a series of steps:

1. Start with a belief set about our current location.
2. Multiply this distribution by the results of our sense measurement.
3. Normalize the resulting distribution.
4. Move by performing a convolution. This sounds more complicated than it really is: this step really involved multiplication and addition to obtain the new probabilities at each location.

Figure 17 Localization

4.2 Particle Filter

Here is a floor plan of an environment where a robot is located and the robot has to perform global localization. Global localization is when an object has no idea where it is in space and has to find out where it is just based on sensory measurements.

The robot, which is located in the upper right hand corner of the environment, has range sensors that are represented by the blue stripes. The sensors use sonar sensors, which means sound, to range the distance of nearby obstacles. These sensors help the robot determine a good posterior distribution as to where it is. What the robot doesn't know is that it is starting in the middle of a corridor and completely uncertain as to where it is.

In this environment the red dots are particles. They are a discrete guess as to where the robot might be. These particles are structured as an x coordinate, a y coordinate and also a heading direction — three values to comprise a single guess. However, a single guess is not a filter, but rather it is the set of several thousands of guesses that together generate an approximate representation for the posterior of the robot.

The essence of particle filters is to have the particles guess where the robot might be moving, but also have them survive, a kind of "survival of the fittest," so that particles that are more consistent with the measurements, are more likely to survive. As a result, places of high probability will collect more particles, and therefore be more representative of the robot's posterior belief. The particle together, make up the approximate belief of the robot as it localizes itself.

Chapter 5

Conclusion and Future Work

The software implemented will be a key part in designing the self-driving car. The techniques used for particle tracking , motion planning and smoothing minimizes error rates to maximum extent and helps in designing a reliable Working self – driving car . Except for the software implementation, a lot goes into implementing the actual car itself .

Efforts can be made to further optimize the error rates in future so that such cars can be used in future in general traffic.

In the last decade, automobile manufacturers have made great progress toward making self-driving cars a reality; nevertheless, there are still a number of technological hurdles to surmount before self-driving vehicles are safe enough for road use. GPS can be unreliable, computer vision systems have limits when it comes to recognising road scenes, and changing weather conditions can impair on-board processors' capacity to accurately detect and track moving objects. Self-driving cars have yet to show that they can recognize and navigate unstructured surroundings like construction zones and accident zones in the same way that human drivers can.

These obstacles, however, are not insurmountable. The amount of road and traffic data available to these cars is growing, newer range sensors are recording more data, and road scene analysis algorithms are improving. The move from human-driven vehicles to completely autonomous vehicles will be gradual, with vehicles doing just a subset of driving functions automatically at first, such as parking and driving in stop-and-go traffic. More driving activities can be reliably outsourced to the car as technology develops.

Chapter 6
References

[1] https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373

[2]http://en.wikipedia.org/wiki/Autonomous_car

[3]http://robohub.org/how-do-self-driving-cars-work/

[4]http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf

[5]http://robots.stanford.edu/papers/thrun.pf-in-robotics-uai02.pdf

[6]http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[7]http://en.wikipedia.org/wiki/PID_controller

[8] Sensor Fusion of Laser and Stereo Vision Camera , By Mrs.Daya Gupta and Sakshi Yadav

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.5692&rep=rep1&type=pdf

[9]https://developer.nvidia.com/blog/drive-labs-how-localization-helps-vehicles-find-their-way/

[10]https://towardsdatascience.com/particle-filter-a-hero-in-the-world-of-non-linearity-and-non-gaussian-6d8947f4a3dc

[11]https://jonathan-hui.medium.com/tracking-a-self-driving-car-with-particle-filter-ef61f622a3e9http://robots.stanford.edu/papers/thrun.ijrr-minerva.pdf

[12] D. Avots, E. Lim, R. Thibaux, and S. Thrun. A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. In IROS-2002

[13]S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hahnel, ¨ C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. International Journal of Robotics Research, 19(11), 2000.

[14] W. Burgard, A.B. Cremers, D. Fox, D. Hahnel, ¨ G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tourguide robot. Artificial Intelligence, 114(1-2), 1999.

[15]G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. An experimental and theoretical investigation into simultaneous localisation and map building (SLAM). In Lecture Notes in Control and Information Sciences: Experimental Robotics VI, 2000. Springer.