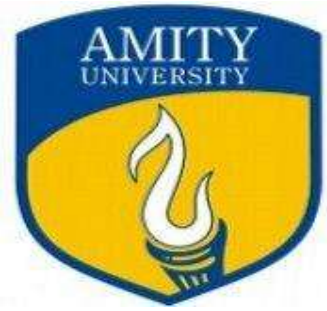
A Project Report

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

Akshay Kumar Raghav

Mohit

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, Akshay Kumar Raghav and Mohit**, student(s) 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

Akshay Kumar Raghav

Mohit

Kuldeep Dwivedi

CERTIFICATE

On the basis of declaration submitted by **Kuldeep Dwivedi, Akshay Kumar Raghav and Mohit**, students 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 nest 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

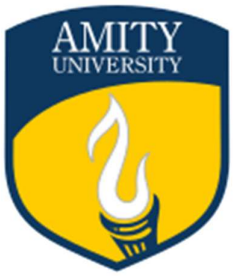
Signature of Guide

(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, **Prof. 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.

Akshay Kumar Raghav

Mohit

Kuldeep Dwivedi

Table of Contents

List of Figures

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

AI research is highly technical and specialized and is deeply divided into subfields that often fail to communicate with each other. Some of the division is due to social and cultural factors: subfields have grown up around particular institutions and the work of individual researchers. AI research is also divided by several technical issues. Some subfields focus on the solution of specific problems. Others focus on one of several possible approaches or on the use of a particular tool or towards the accomplishment of particular applications.

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 also they can communicate with one 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 sense their environment and based on their surrounding these vehicles 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 to reach its destination and then 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 maximize its chances 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[1].

The research work for AI is very technical and specialized, and it is divided into numerous sub fields that use to fail frequently to communicate with one another. Some of the divide is due to social and cultural factors: sub-fields have developed around specific institutions and researchers' activity. Several technological difficulties also split AI research. Some sub-fields concentrate on solving specific challenges. Others concentrate on one of many alternative approaches, the use of a specific instrument, or the completion of specific tasks[1].

Reasoning, knowledge, planning, learning, NLP(communication), perception, and the ability to move and manipulate 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 uses a variety of methods, such as search and mathematical optimization, logic, probability, and economics-based methodologies, and so on. Artificial intelligence is an interdisciplinary field that encompasses a wide

range of sciences and professions, including computer science, mathematics, psychology, linguistics, philosophy, and neuroscience, as well as more specialized fields such as artificial psychology.

2.2 Goals

The difficulty of imitating (or creating) intelligence has been broken down into sub-problems. Researchers are looking for specific features or abilities in an intelligent system. The following features have received 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 in the late 1980s and 1990s had created extremely successful methods for coping with 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; neural net research aims to recreate the brain mechanisms that lead to this ability; statistical AI techniques imitate the probabilistic character of human guesses.

2.2.2 Knowledge representation

An ontology is a set 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 other 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].

2.2.3 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 must be able to set and achieve objectives. They require a method for considering the future and making actions that improve the efficiency of the alternatives accessible to them[3].

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, or the study of self-improving computer systems, has always been at the center 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 developing 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 favorable replies and penalized for poor ones in reinforcement learning. The agent devises a strategy for working in its issue based on this collection of rewards and punishments[3].

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 such as newswire texts may be possible with a sufficiently capable natural language processing system. Feature extraction (or text mining), question answering, and translation are only a few of the applications of natural language processing.

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 ability to derive features of the world using sensor input (such as cameras, microphones, tactile sensors, sonar, and other more exotic devices). The capacity to analyse visual information is known as computer vision. Speech recognition, facial recognition, and object recognition are a few examples of subproblems[5].

2.2.7 Motion and manipulation

Robotics and artificial intelligence are closely connected topics. For robots to perform tasks like object manipulation and navigation, intelligence is required, with sub-problems like localization (recognising where things are), mapping (learning what is around), and motion planning (calculating 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 ability 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 period of 50 years of research. A couple of the more 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 other difficulties. For example, the satplan algorithm employs logic to plan, whereas inductive logic programming is a training aid.

Several distinct forms of logic are employed in AI research. Propositional or sentential logic is the logic of statements that can be true or false. Quantifiers and predicates are also supported by first-order logic, allowing the declaration of facts about things, their attributes, and their connections. The truth of a proposition is assigned a value between 0 and 1, rather than merely True (1) or False (1), in fuzzy logic. For uncertain reasoning, fuzzy systems are frequently employed in today's industrial and consumer product control systems. Subjective logic expresses uncertainty in a different and more clear way than fuzzy logic: inside a Beta distribution, a given binomial opinion meets belief + disbelief + uncertainty = 1. This method differentiates ignorance from very confident probabilistic claims made by an agent.

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

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

Bayesian networks can assist in reasoning (using the Bayesian inference process), learning (using the expectation-maximization technique), planning, and perceiving. By filtering, forecasting, smoothing, and providing explanations for streams of data, probabilistic algorithms can assist perception systems in analyzing processes that occur across time[10].

"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, precise mathematical tools have been constructed to examine how an agent makes decisions and plans. Markov decision processes, dynamic decision networks, game theory, and mechanism design are some of the strategies that can be used[10].

2.3.4 Classifiers and statistical learning methods

The two most fundamental types of AI applications are classifiers and controllers. However, because controllers must first recognize situations before inferring actions, categorization is a critical component of many AI systems. Classifiers are functions that compare patterns to get the closest match. They are particularly intriguing for use in AI since they may be adjusted based on examples. The phrases used to describe these occurrences are observations or patterns. In supervised learning, each pattern belongs to a certain class. A class might be viewed as a decision that must be made. All of the observations and their class labels make form a data set. When a new observation is obtained, it is classified based on previous knowledge.

A classifier can learn using a number of methods, such as statistics and machine learning. The neural network, kernel techniques such as the support vector machine, k-nearest neighbor algorithm, Gaussian mixture model, naive Bayes classifier, and decision tree are the most commonly used classifiers. These classifiers' performance has been compared across a number of tasks. The qualities of the data to be classified have a considerable influence on a classifier's performance. The "no free lunch" theorem states that no one classification technique is optimal in all situations. It seems more like an art than a science to pick the correct classifier for the job[11].

2.3.5 Neural networks

A neural network is a collection of nodes that resembles the human brain's massive network of neurons. Walter Pitts and Warren McCullough pioneered the concept of non-learning neural nets a decade before the field of AI research was established. The perceptron is similar to linear regression, was invented by Frank Rosenblatt.

The two most common forms of neural networks are acyclic or feedforward neural networks (in which the signal only flows one way) and recurrent neural networks. The

most common feedforward networks are perceptrons, multi-layer perceptrons, and radial basis networks.

The backpropagation method, initially developed as the reverse mode of automated differentiation and later extended to neural networks, is now widely used to train neural networks. Hierarchical temporal memory is a technique for replicating some of the structural and algorithmic properties of the neocortex.

2.3.6 Control Theory

Control theory is an engineering and mathematics interdisciplinary discipline that analyses the behavior of dynamical systems with inputs and how feedback impacts their behavior.

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 only forward and backward movements are possible and 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 a new programming language called Pyro as a result of their recent efforts. 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 a tool 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 turn 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."

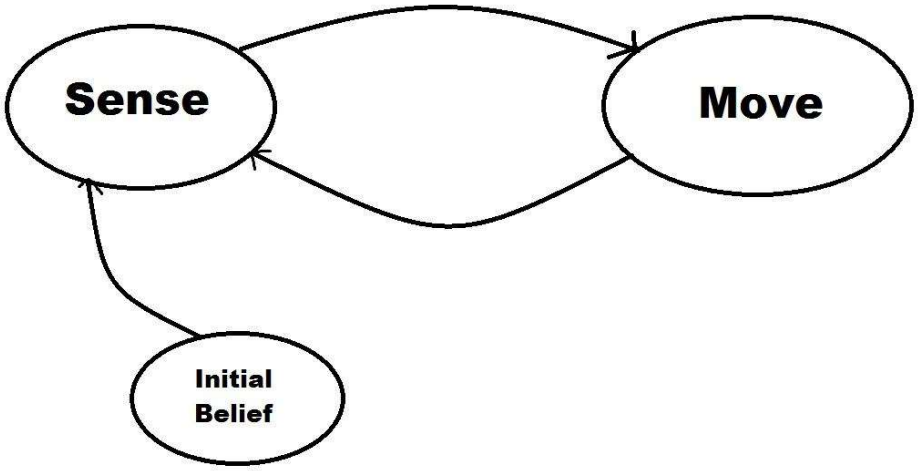


Figure 1 Localization Perception

3.2 Kalman Filter

Kalman filter is an estimator that infers parameters parameters of interest from indirect, inaccurate and uncertain observations. We can say a Kalman Filter is an optimal estimator. Kalman filter has a recursive nature so that new measurements can be processed as they arrive.

We need to represent unpredictable disturbances with a discrete time linear dynamic system described by a vector difference equation with additive white noise.

A deterministic dynamic system's state is the smallest vector that sums up the system's whole history. In the absence of noise, knowing the state allows for theoretical prediction of the deterministic system's future (and prior) dynamics and outputs.

Optimization- If all noise is Gaussian, the Kalman filter minimizes the mean square error of the estimated parameters.

Junior, Stanford's most current self-driving car, is equipped with laser range finders that take distance scans ten times per second, resulting in approximately one million data points. The main purpose of gathering all of this information is to locate other vehicles. The Kalman filters use data from the range finders as input.

Our self-driving automobile will avoid colliding with other vehicles if we can figure out how to detect them. To avoid collisions, we need to know not only where other automobiles are - as in the case of localization - but also how fast they are traveling.

3.2.1 Why Kalman Filter is better?

- Due to optimality and structure, good results in practice.
- Convenient form for online real time processing.
- Gives a fundamental understanding, it is simple to formulate and implement.
- There is no need to invert the measurement equations

3.2.2 Why Filter?

The process of determining the "best estimate" from noisy data is referred to as "noise filtering." A Kalman filter, on the other hand, not only cleans up the data measurements, but also projects them onto the state estimate.

3.2.3 Tracking

The Kalman Filter is a well-known method for estimating the state of a system as it produces a uni-modal distribution by estimating a continuous state.

Example- A car (the large rectangle) senses an object in space (*) at times *t=0, t=1, t=2, t=3*.Where would you assume the object would be at *t=4*?

The velocity is changing according to a vector. Assuming there is no drastic change in velocity, we can expect t=4 to be along the same vector line.

The Kalman filter is a piece of software that allows you to take points, even if they are unclear, and anticipate where future locations might be, like in this example. You'll also be able to determine the speed of the moving object. Based on laser and radar data, the Google self-driving car employs tactics like these to figure out where the traffic is.

3.2.4 Gaussian

A histogram is a visual representation of continuous space divided into discrete areas. It's a technique for simulating a continuous distribution. The distribution is presented in a Gaussian, which is a continuous function over the space of locations, in Kalman filters. The total area beneath the Gaussian add upto one. If we call the space (x-axis), x, then the Gaussian is characterized by two parameters - the mean (greek letter mu, μ), and the width of the Gaussian, called the variance (often written as a quadratic variable σ2.

Any Gaussian in a one-dimensional parameter space, is characterized by (μ,σ2). Our task when using Kalman filter is to maintain a μ and a σ2, that serves as our best estimate of the location of the objects we are trying to find.

The exact formula is an exponential of a quadratic function:

- $f(x) = \frac{1}{\sqrt{2\pi\sigma 2}} exp\left[\frac{-1}{2}\frac{(x-\mu)2}{\sigma 2}\right]$

The Kalman Filter represents all distributions of the Gaussians and iterates two things:

1. Measurement updates
2. Motion updates

The measurement cycle is commonly referred to as a measurement update when utilizing Kalman filters, whereas the motion cycle is referred to as prediction. The Bayes Rule - a product, multiplication - will be used in the measurement update. Total probability will be used in the prediction update - a convolution and an addition. Gaussians can be used to discuss these two cycles. Assume you're trying to locate another vehicle and you have the following prior distribution with a mu mean: Then, in blue, with a mean nu, we have a measurement that informs us something about the

vehicle's localization. Predicting the peak can be done by the resulting Gaussian that is more certain than the two component Gaussians. That is, the covariance is smaller than either of the two covariances in isolation. Intuitively speaking, this is the case because we can actually gain information from the two Gaussians.

This outcome may appear counter intuitive at first. It's helpful to imagine these two measures being taken at the same time (perhaps one is from a laser and the other from a radar sensor). Each of these sensors may make a mistake, but we MUST obtain greater assurance about our position by utilizing both sensors than we would if we only used one (otherwise, why would we use a second sensor?). As a result, the resulting peak will have to be higher.

3.2.5 Co-Variance

Co variance is determined as:

The covariance of two random variables x_1 and x_2 is

$$\begin{aligned} \operatorname{cov}(x_1, x_2) &\equiv E[(x_1 - \bar{x}_1)(x_2 - \bar{x}_2)] \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x_1 - \bar{x}_1)(x_2 - \bar{x}_2)p(x_1, x_1)dx_1 dx_2 \\ &\equiv \sigma^2_{x_1 x_2} \end{aligned}$$

where p is the joint probability density function of x_1 and x_2.

The ***correlation coefficient*** is the normalised quantity

$$\rho_{12} \equiv \frac{\sigma^2_{x_1 x_2}}{\sigma_{x_1}\sigma_{x_2}}, \quad -1 \le \rho_{12} \le +1$$

The covariance of a *column vector* $\mathbf{x}=[x_1 .. x_n]'$ is defined as

$$\begin{aligned} \text{cov}(\mathbf{x}) &\equiv E[(\mathbf{x}-\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})'] \\ &= \int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty}(\mathbf{x}-\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})'\,p(\mathbf{x})\,dx_1..dx_n \\ &\equiv \mathbf{P}_{\mathbf{xx}} \end{aligned}$$

and is a *symmetric n* by *n* matrix and is *positive definite* unless there is a linear dependence among the components of $\mathbf{x}$.

The $(i,j)^{th}$ element of $\mathbf{P}_{\mathbf{xx}}$ is $\sigma^2_{x_i x_j}$

Interpreting a covariance matrix:

diagonal elements are the variances, off-diagonal encode correlations.

3.2.6 Diagonalizing a Covariance Matrix

cov(x) is symmetric => can be diagonalized using a basis that is orthonormal We achieve separation of error contributions by shifting coordinates (pure rotation) to these unity orthogonal vectors. The eigenvectors create the axes of error ellipses, and the basis vectors are the eigenvectors. The axes' lengths correspond to the standard deviations of the independent noise contribution in the direction of the eigenvalues and are the square root of the eigenvalues.

3.2.7 Equal Variance

Suppose we have two Gaussians, as in Baye's Rule, a prior μ and a probability σ^2, and the measurement has a mean of ν (nu) and a covariance of r^2, so that the new mean, μ' is the weighted sum of the old means, where μ is weighted by r^2, ν is weighted by σ^2 and normalized by the sum of the weighting factors.

The new variance term after the update is given by the following equation:

$$\frac{1}{\sigma^{2\prime}} = \frac{1}{\sigma^2} + \frac{1}{r^2}$$

We can determine the location of the new mean on the graph because we know that:

1. The prior Gaussian has a much higher uncertainty, a longer, lower slope, therefore σ2 is larger

2. This means the ν is weighted much larger than the μ

3. So the mean will be closer to the ν than the μ

3.2.8 Multivariate Gaussian

We've just talked about one-dimensional motion so far, and while this covers all of the elements of a Kalman filter, we should at least touch on what happens when we go from one to higher dimensions.

1. Instead of being a number, the mean becomes a vector with one element for each of the dimensions.

2. The variance is replaced by the covariance, which is a matrix with d rows and d columns when the space has d dimensions.

3. This gives us a complicated formula, but it is not something you need to remember.

Create a two-dimensional estimate with the x-axis representing location and the y-axis representing velocity, which we'll refer to as v, but Sebastian uses an x with a dot over it. (The first derivative of x (position) with respect to t (time), which is another way to indicate velocity, is represented by an x with a single dot above it.)

If you know where you are but not how fast you are going, you can express it with an extended Gaussian about the proper position but quite broad in the velocity space. We can't forecast the location you'll assume in the prediction step because we don't know the velocity. There are, nevertheless, some intriguing connections. If the velocity is 0 and the prediction is correct, where would the posterior be after the prediction? If you know you start at position 1, a velocity of zero will put you exactly where you started (1, 0).

3.2.9 Motion Update

Let's say we move to the right, which has its own set of uncertainties. The motion instruction is then added to the mean, resulting in a prediction with a higher uncertainty than the initial uncertainty.

This makes sense intuitively; as you walk to the right, your uncertainty grows as you lose information about your location (as manifested by the uncertainty). The math: New mean is the old mean plus the motion, u. The new $\sigma^2$is the old σ^2plus the variance of

the motion Gaussian,r^2. This is *very* similar to Unit One, where motion decreased our knowledge and updates increased our knowledge. This is a fundamental aspect of localization.

3.2.9 Kalman Filter Designing

A state transition function and a measurement function, both of which are represented as matrices, are required when creating a Kalman Filter. The state transition matrix F and the measurement matrix H are referred, respectively. The update step and the prediction step are separated in our Kalman filter. The names of the matrices and variables, as well as the equations for these steps, are listed below.

This is essentially a more complex version of the one-dimensional situation we've been discussing. When you use the given measurements to run the filter, you can estimate the velocity and make better forecasts. When you run the filter, you want the measurement update to come back first, followed by the motion update.

measurements=[1,2,3]
x=matrix([[0.],[0.]])

p=matrix([[1000.,0.],[0.,1000.]])

u=matrix([[0.],[0.]])

F=matrix([[1.,1],[0,1.]])

H=matrix([[1.,0]])

R=matrix([[1.]])

I=matrix([[1.,0.],[0.,1.]])
filter(x, P)

Once you fill it in you get an output. This output iteratively displays *x* and *P* as they update. Notice that at the first output of *x,* we know nothing about the velocity. This is because we need multiple location measurements to estimate velocity. As we continue making measurements and motions, our estimate converges to the correct position and velocity.

This is the model that Google's self-driving car uses to create predictions about where other cars are and how fast they're moving. The car has radar on the front bumper that

measures distance from other vehicles and provides a noisy estimate of speed. Additionally, the automobile utilises its roof-mounted lasers to measure the distance between other cars and calculates a noisy estimate of velocity.

So, when the Google car is on the road it uses radars and lasers to estimate the distance and the velocity of other vehicles on the road using a Kalman filter, where it feeds in range data from the laser and uses state spaces of the relative distance, *x* and *y*, and the relative velocity of *x* and *y*.

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.

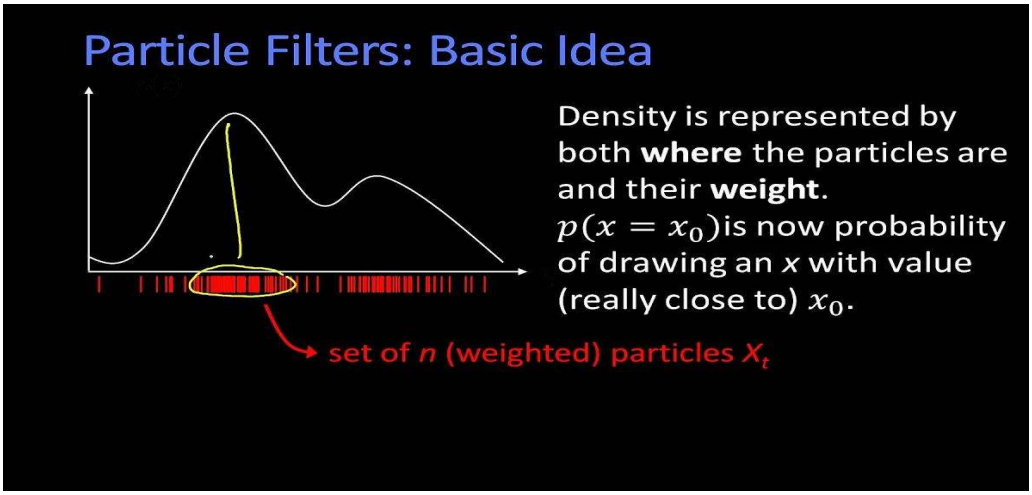


Figure 2 Particle Filter Basic Idea

When it comes to 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 when it comes to 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.

The 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 | 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.

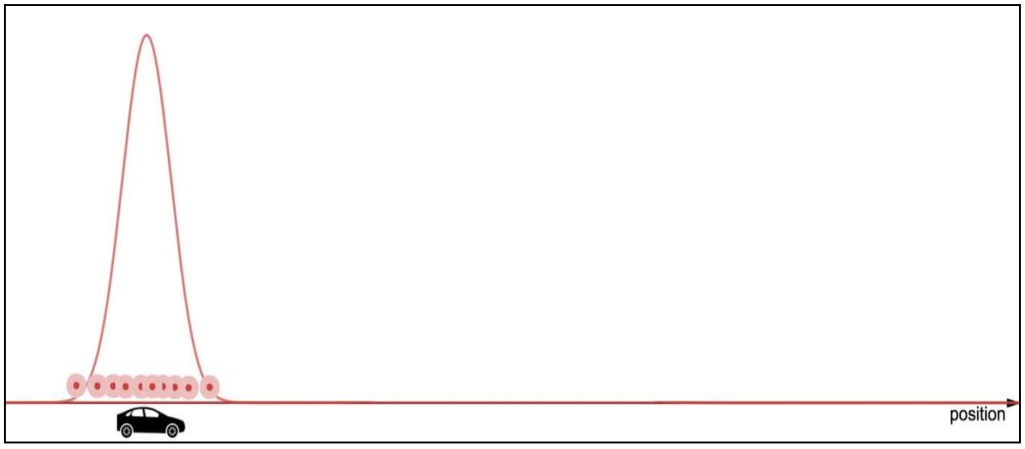


Figure 3 Phase 1

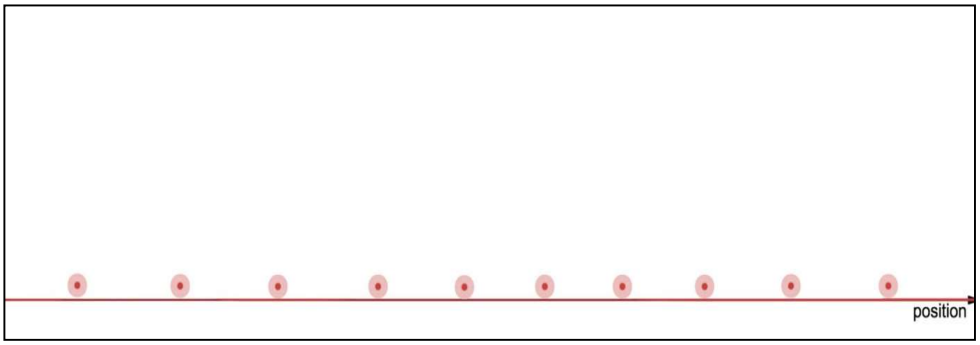


Figure 4 Phase 2

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

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

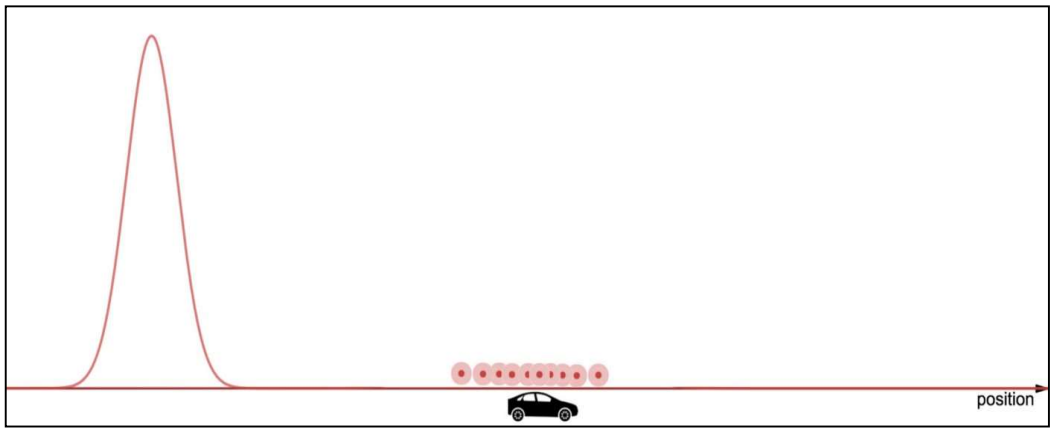


Figure 5 Phase 3

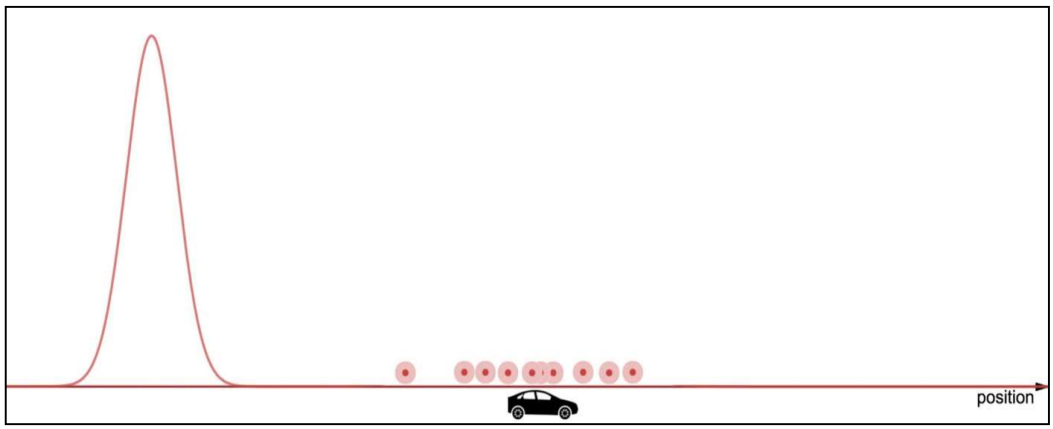


Figure 6 Phase 4

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 3.17 Phase 4

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

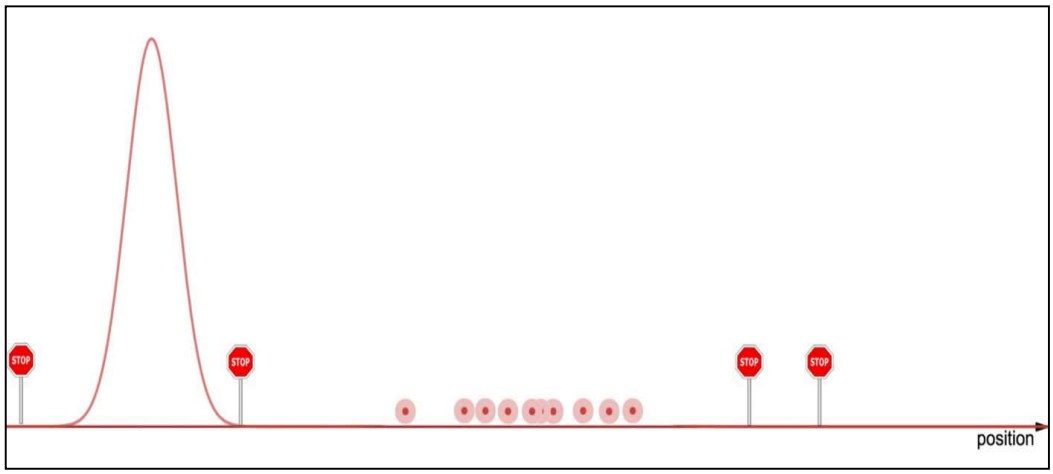


Figure 7 Phase 5

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

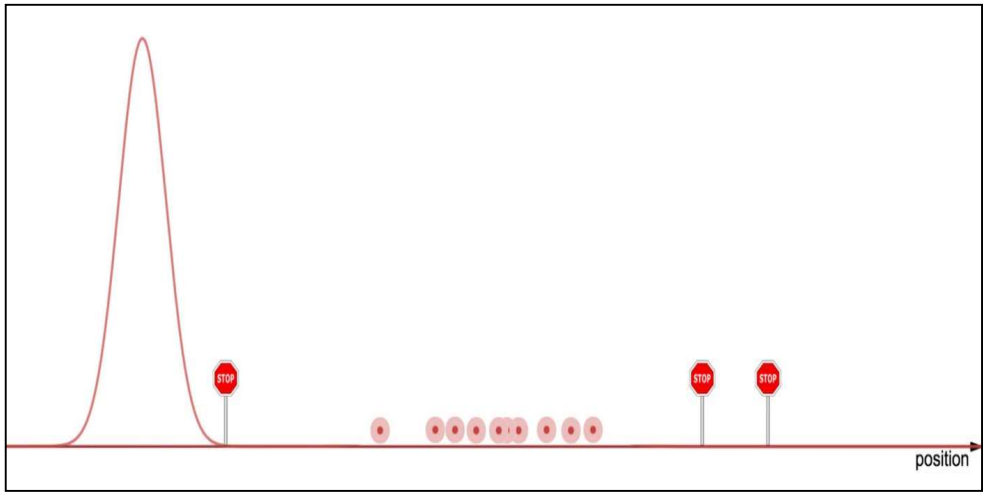


Figure 8 Phase 6

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 use the distance and orientation information to use a map to find the nearest landmark. In P2, for example, we use our first sensor reading to select the right-most stop sign.

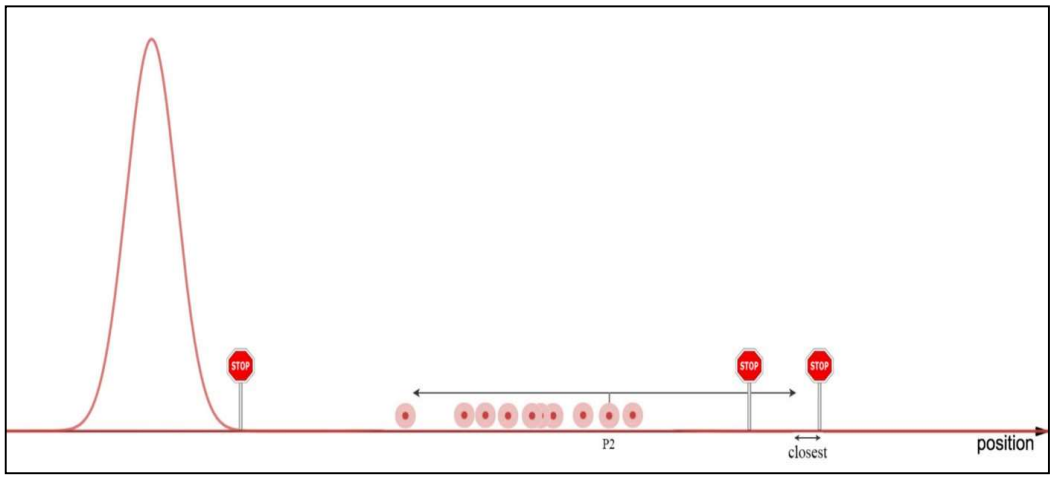


Figure 9 Phase 7

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

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

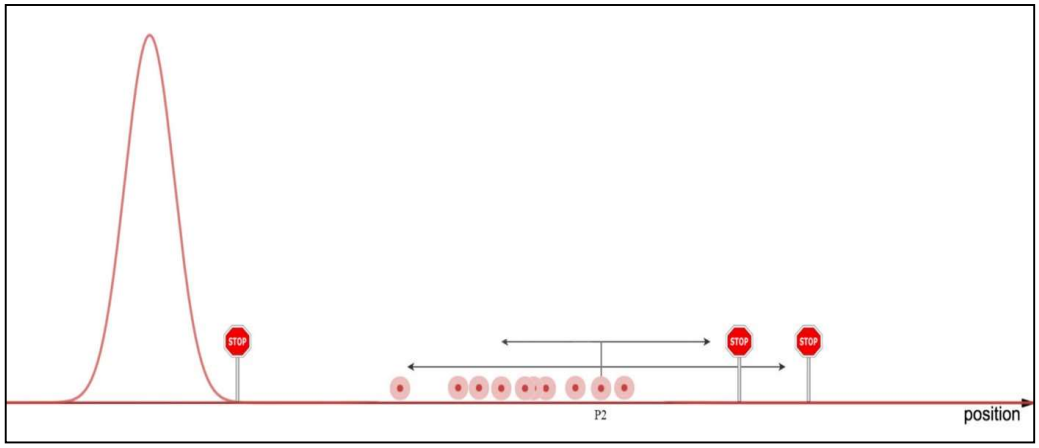


Figure 10 Phase 8

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.

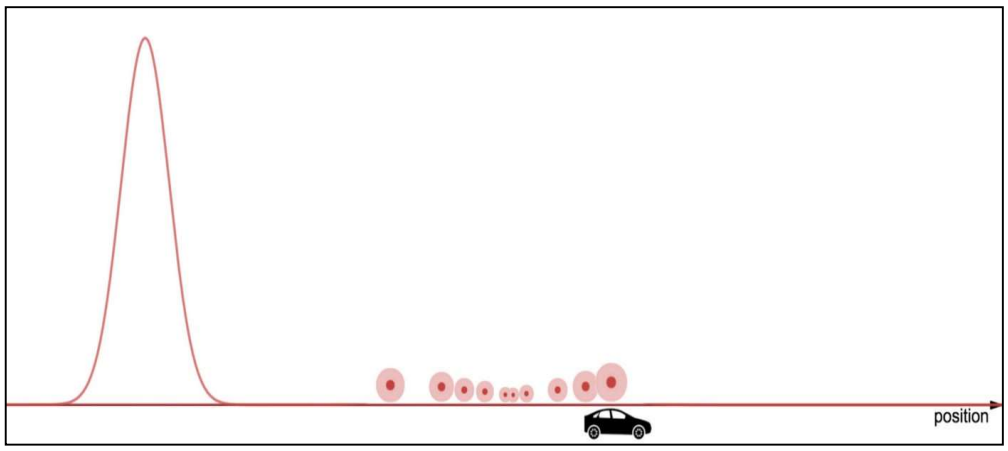


Figure 11 Phase 9

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.

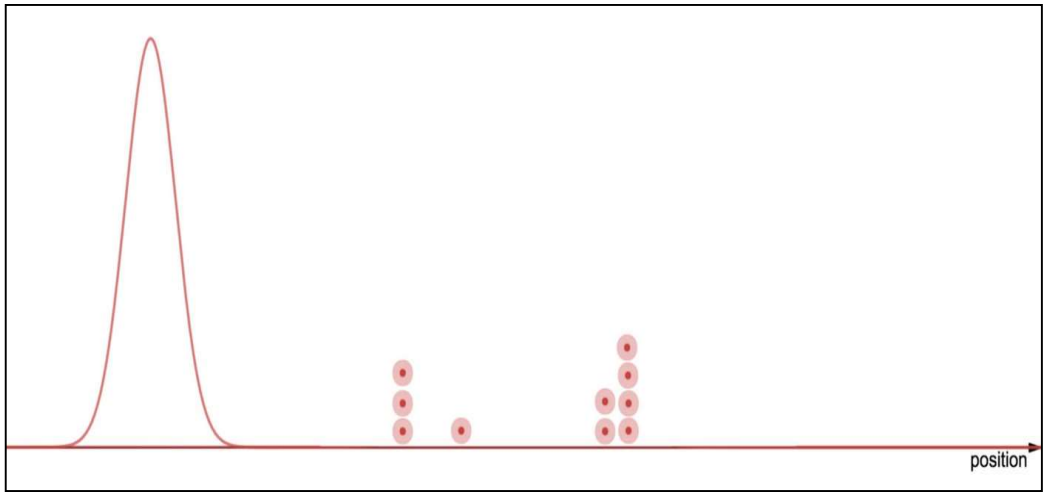


Figure 12 Phase 10

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.

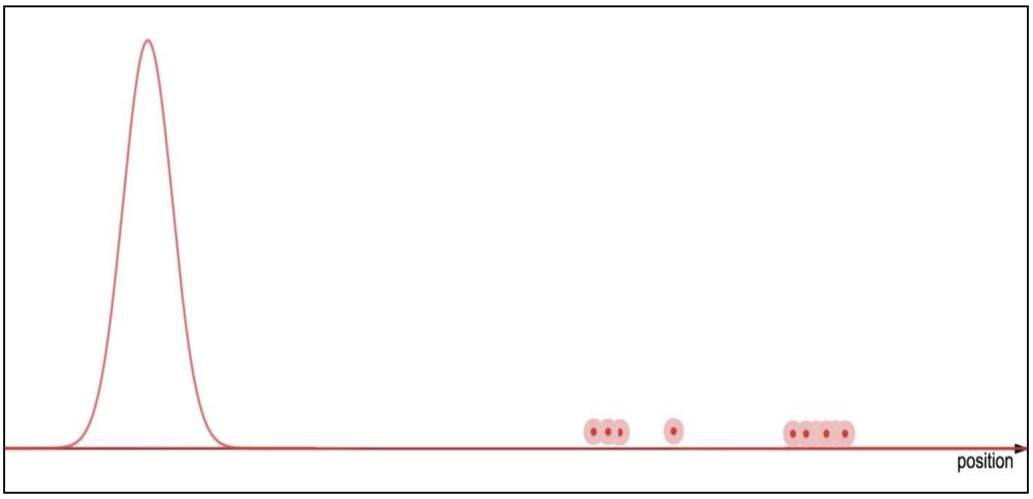


Figure 13 Phase 11

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.

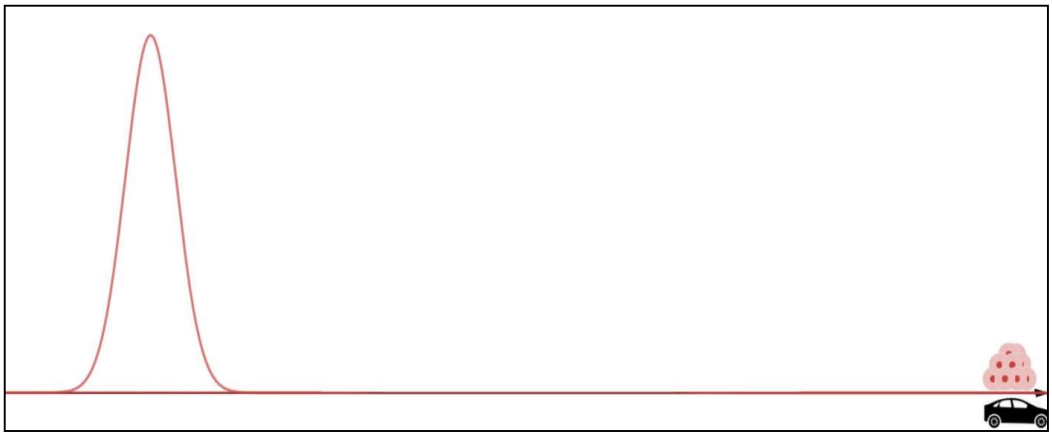


Figure 14 Phase 12

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.

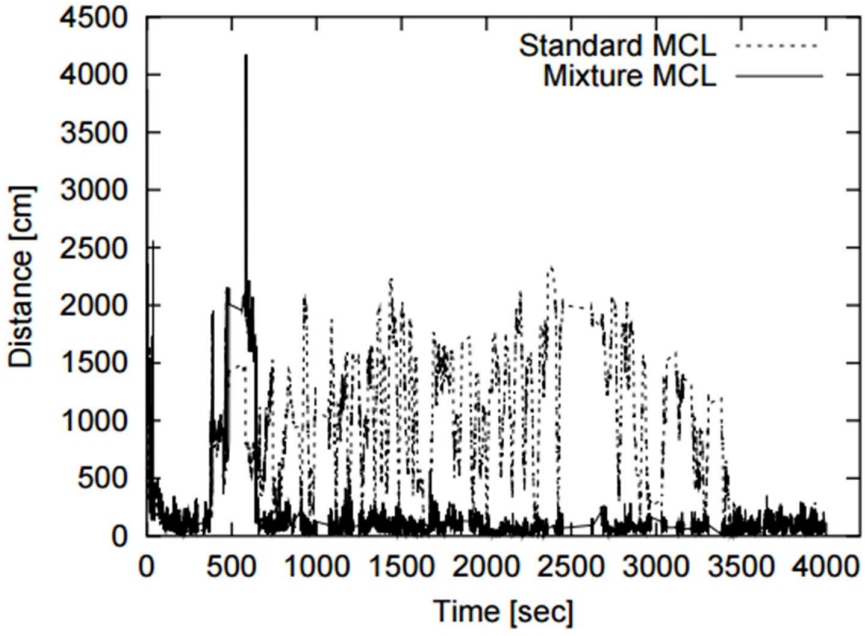


Figure 15 Particle Filter

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.

Particle filters are also known as Monte Carlo localization in the context of 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 figure on the right displays the location of the robot 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.

3.4 Motion Planning

3.4.1 Introduction

Once we have located the car or the robot and its surrounding environment for various obstacle using the concepts such as localization and Kalman filter. Next step is to find the path so that the robot or can perform motion on that path.

The fundamental problem in motion planning is that a robot might live in a world that looks like the one pictured below and will want to find a goal like $. The robot has to have a plan to get to its goal, as indicated by the red line route.

Given:

- Map Starting Location
- Goal Location Cost (time it takes to drive a certain route)
- **Goal**: Find the minimum cost path.

Computing Cost (Compute Cost):

Suppose you live in a discrete world that is split up into grid cells and your initial location is facing forward, while your goal location is facing to the left.

Assume that for each time step you can make one of two action -- either move forward or turn the vehicle. Each move costs exactly one unit.

This same problem occurs for a self-driving car that lives in a city and has to find its way to its target location by navigating through a network of streets and along the highway.

We are concerned with 2 topics path planning and motion planning. There is a very small difference between these 2 terms. Suppose that there two positions on a binary occupancy grid or in simple words environment, starting position (X pos) and Goal position (Y pos). Finding the path between starting position and goal position is known as **Path Planning**. In **Motion Planning** also we find the path between starting and goal positions but here we are also concerned with the derivatives of path such as velocity, acceleration, rotation etc. With motion planning we are trying to dictate precisely how the robot moves through the environment. With path planning we are just concerned with the path it takes and not have fast the robot accelerates or moves through it.

There are many methods to implement motion planning such as-

- Search based methods
- Sampling based methods
- Grid based search
- Graph based methods
- Interval-based search
- Geometric based algorithms etc.

But in this project, we have considered two methos namely Search based and Sampling based.

Figure 16 Simple environement

Consider a simple environmental map (fig 6.1), there are staring and goal positions on the map. To find minimum distance between start pose and goal pose we can try a straight line between these two positions. If the robot moves this path, it will reach the goal position in the shortest time. But as we know that in real life the environment is filled with many obstacles, therefore we cannot find solution for many problems in this way.

3.4.2 Graph based methods

Graph based algorithms work by discretizing the environment that is breaking it up into discrete points or nodes and then finding the shortest distance to the goal. Let's approach this problem (fig 6.2) in a Radom way. We have to find the shortest path between start pose and goal pose.

The first node in our graph is start pose and it has a cost of 0 since we are already there, so we will put a 0 inside the node and then we move in a random direction and place a node in the graph at new location. The edge between the nodes is how far we have travelled and cost of getting to this node is the length of that one edge. Now we take another step in a completely random direction, place another edge and the cost of this node is the total of three units and we can continue to this process until we reach the goal as shown in fig 6.2.

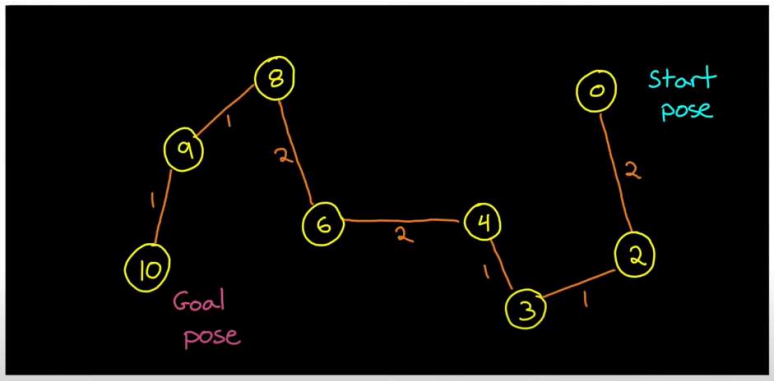


Figure 17 Finding path from start to goal

The cost of this random path is 10 units (2+1+1+2+2+1+1). We have found out a path, but we are not sure if this is the shortest path. So, we have start again and adding a node and connecting it with an edge and recording its cost and if we happen to reach a node that we visited before we can compare the cost between the two different paths that we took to reach it and keep the smallest one. We don't need to build a fully interconnected

graph. We can build a tree which is subset of a graph, in a tree each node only has a single parent. In a tree we can get to anode two different ways, it doesn't make sense to keep the path that is longer if we are finding the shortest path so we can remove the edge for the longer path keeping a tree structure. In this way a tree would start at the location of the robot and the branches would venture out to other states but never recombine. Now to find shorter distance to the goal we can just keep randomly wandering the environment Updating the tree until we find a branch that gets the goal with a cost that is low. This process doesn't guarantee a optimal path but it will continue to approach optimal as number of nodes go to infinity.

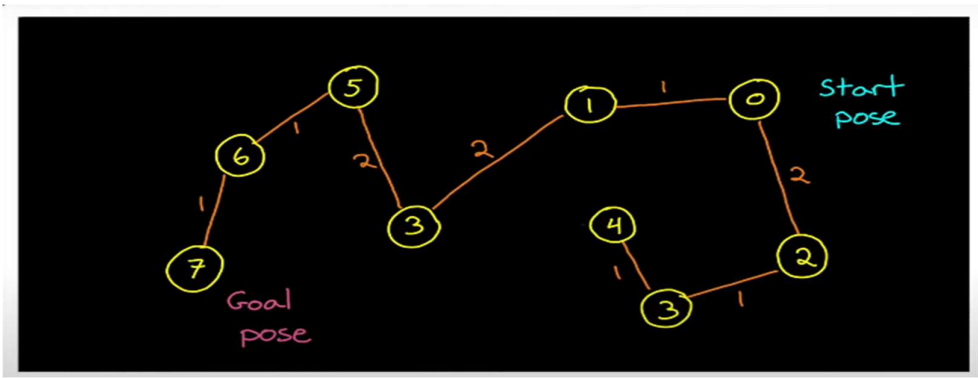


Figure 18 Finding path

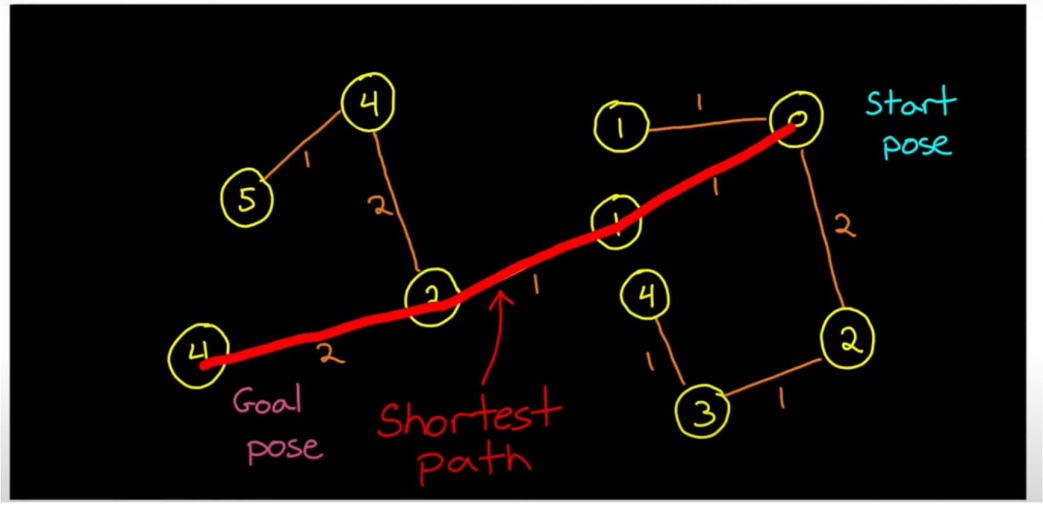


Figure 19 Shortest Path

Random wandering is not the best solution, this is where path planning algorithms comes. These provide more efficient ways to build this tree.

3.4.3 Search based methods

Search based method is one of the motion planning methods. In this method we use graph search methods to find paths over a discrete representation of the problem.

In search-based method we build up the tree by adding nodes in an ordered pattern. One way to do this task is that we start with grid-based map like the occupancy grid. We have to go cell by cell and determine the cost or the distance the robot would have to go in order to reach that cell.

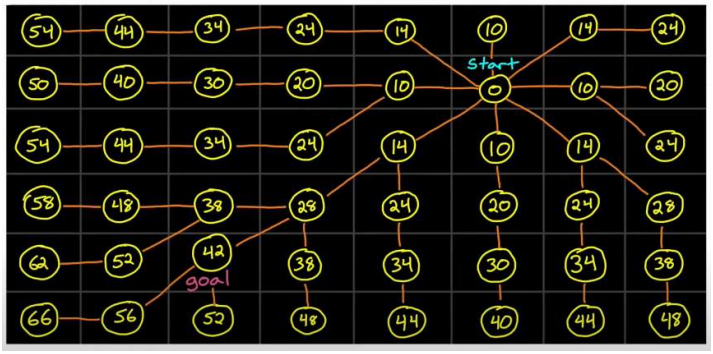


Figure 20 Binary occupancy grid

As we can see in fig 6.6 the entire space is divided into cells. Here we have assumed that adjacent moves cost 10 unit whereas diagonal moves cost 14 units. This method is similar to the random approach that we did in graph-based method except we are methodically working our way through each cell and calculating the cost to reach it and if it's the shortest path to that node then we update the cost and the tree to reflect it. Once we have covered every single cell in the grid the optimal path is simply the sequence of cells that produce the minimum cost at the goal.

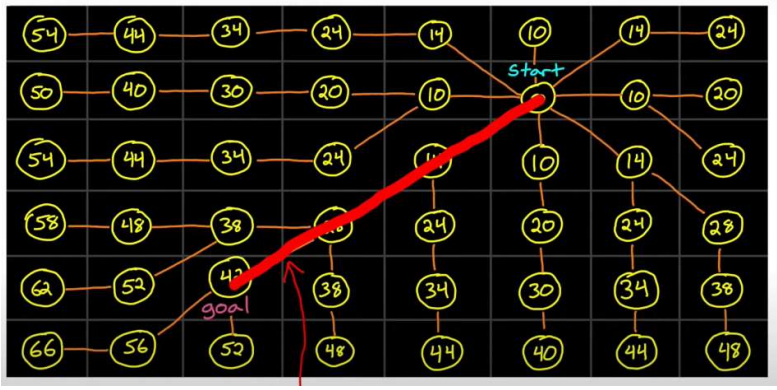


Figure 21 Optimal Use in search-based method

Search based methods provide optimal path, but these are also computationally expensive since it is kind of brute force method.

Some common search base algorithms are-

- A* algorithm
- Breadth first search
- Dijkstra algorithm

The advantage of search-based methods is that these gives us optimal path and but its computationally expensive.

To solve the problem of search-based algorithms the researchers came up with A* (A star) algorithm. This was done in 1968 to give Shakey the robot the ability to determine where to go on its own. This was done first time for a general-purpose mobile robot.

3.4.3.1 Pros and cons

Both methods, search based and graph based have their pros which can debated for hours. The main differences between graph-based method and search-based method are speed and optimization.

Search based methods gives us optimal path and but its computationally expensive.

3.5 A* search algorithm (A Star):

The A* (A-star) algorithm can be used for our path finding problem. It is a variant of the search algorithm that we implemented, that is more efficient because you don't need to expand every node. The A* algorithm was first described by Peter Hart, Nils Nilsson and Bertram Raphael in 1968. If you understand the mechanism for searching by gradually expanding the nodes in the open list, A* is almost the same thing, but not quite.

A* uses a heuristic function, which is a function that has to be set up. If it's all zeroes, A* resorts back to the search algorithm already implemented. If we call the heuristic function h, then each cell results in a value.

A* algorithm is a search-based method. It still adds nodes in a ordered way but it does so by prioritizing the nodes that are more likely to produce the optimal path and searching them first. It does this by keeping a track of heuristic value like the straight-line distance from a node to the goal in addition to the cost of the node and the sum of

these numbers is the absolute minimum cost of the path. Let's take an example given the figure below. In the figure there is a straight line shot to the goal then imagine the total path length for this node would be 48. We have already gone 10 units and now we have a minimum of 38 units left to reach the goal. In this case other node is prioritized even though its current cost is 14 since there is the potential of only having 28 more units to go for a total of 42.

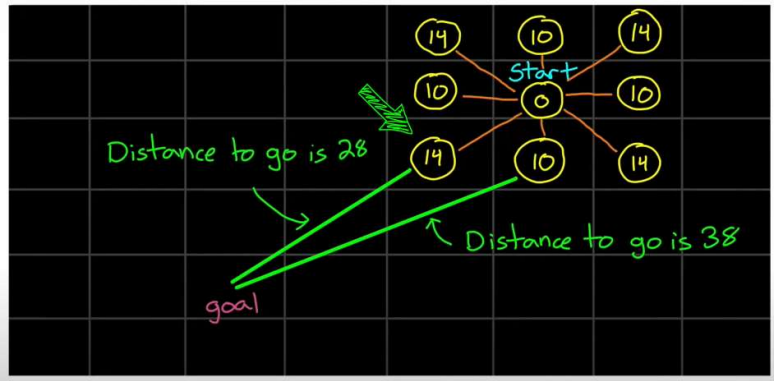


Figure 22 A algorithm*

It makes sense to keep trying this path so in this way A* allows us to search through the nodes in a way that will get us to the goal node without necessarily having to add every node into our tree.

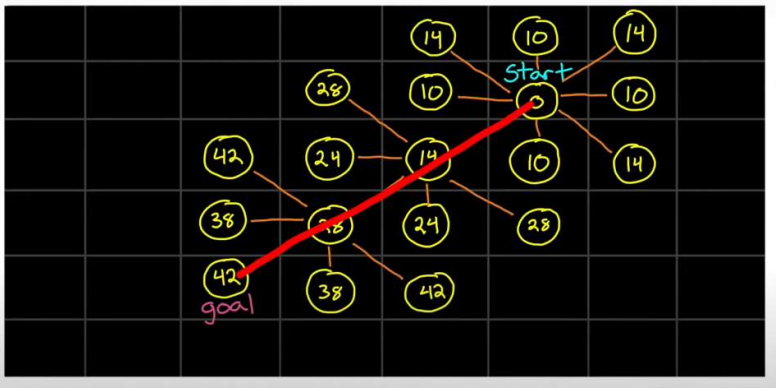


Figure 23 A algo output*

Using A star algorithm, once we reach to the goal node, we know that we took the optimal path since every other path would have a cost-plus distance to go that is greater than the path we found.

3.5.1 Examples of A* in the real world:

Here we can see an actual implementation from the DARPA Urban Challenge. A Stanford car trying to find a way through a maze:

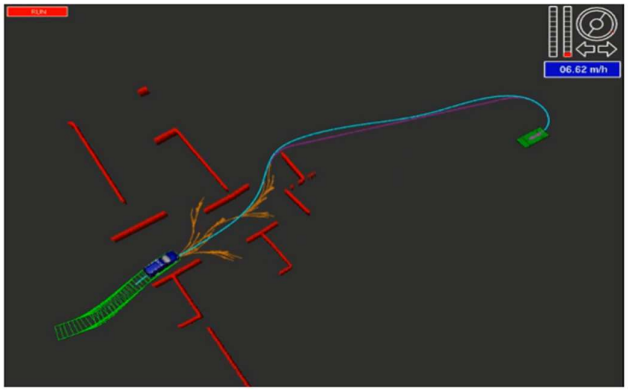


Figure 24 A in real-life*

In the fig 6.9 we can see the maze constantly changing as the car moves. This reflects the fact that the car uses sensors to see obstacles, and obstacles are sometimes occluded. This means the car can only see them when they are nearby. What's really remarkable here is that the car is able to plan highly complex maneuvers towards the goal. At any point in time we can see it's best guess towards an open path to the goal. The orange trees are the A* search trees. The car can turn at different angles. Each step is a different turning angle combined with a different forward motion. However, leaving this aside we have amazing trees that find a path all the way to the goal using A*.

This implementation is so fast that it can plan these paths in less than 10 milliseconds for any location in this maze. It was faster than any other team at the DARPA Grand Challenge or the DARPA Urban Challenge.

The planning is repeated every time the robot cancels the previous plan. We can see additional adjustments in certain places and times. We can see that A* is planning with a simple Euclidean distance heuristic providing the car the ability to find a path to the goal.

The big difference between implementing this and implementing it on a grid is the motion model. You have to implement a robot that is able to turn and you have to write all the math of a robot that is able to turn and go forward. This robot can also revert, so going backwards becomes a distinct different action. But other than that it's essentially the same A* algorithm you just implemented. So, if you want to build a

self-driving car, you now understand how to make a really complex, search algorithm to find a path to a goal.

This is a scene where DARPA trapped the self-driving car using a barrier that went all the way across the street.

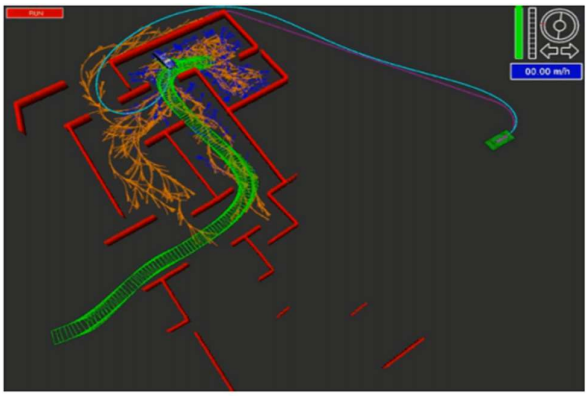


Figure 25 A algo in real environement*

The only way for the car to navigate is to take multiple U-turns, and it had to plan it all by itself using A* planning. The car pulls up to the barrier, realizes there is no valid path, and invokes it's A* planner to come up with a turnaround maneuver that is not particularly elegant, but is super effective. The car was able to turn around by itself using A*, find the optimal plan to do so and then move on. Otherwise, it would have been stuck forever behind the obstacle.

3.5.2 Writing a Search Program (First Search Program)

Now, the question is, can you write a program that computes the shortest path from the start to the goal? The first step towards writing this program is to name the grid cells. Across the top, name the columns zero to five, and along the side you can name the rows zero to four. Think about each grid cell as a data point, called a node and the goal point is called the goal node.

1. From here you will make a list of nodes that you will be investigating further -- that you will be expanding. Call this list open. The initial state of this list is [0,0]. You may want to check off the nodes that you have already picked.
2. Now, you can test whether this node is your final goal node -- which, just by looking at the grid you can tell that it obviously is not. So, what you want to do

next is to expand this node by taking it off the open list and looking at its successors.

3. Then, you can check those cells off on your grid. Most importantly, remember to note how many expansions it takes to get to the goal -- this is called the g-value. By the end of your planning, the g-value will be the length of the path.
4. The process of expanding the cells, starting with the one with the lowest g-value, should yield the same result you found as the number of moves it takes to get from the start to the goal.

3.5.3 A* algorithm implantation

Make an open list containing the first place

When it arrives at its destination:

Make a blank closed list

If it does not reach the destination, consider the area with the lowest f-effect on the open list.

We're done

Else:

List the current node and check its neighbors

To each neighbor of the current node:

If a neighbor has a lower value of g than the current node and is on a closed list:

Replace the neighbor with this new node as the neighbor's parent

Else If (current g is low and neighbor is on open list):

Replace the neighbor with the minimum value of g and change the neighbor's parent to the current node.

Else If the neighbor is not on both lists:

Add to the open list and set g

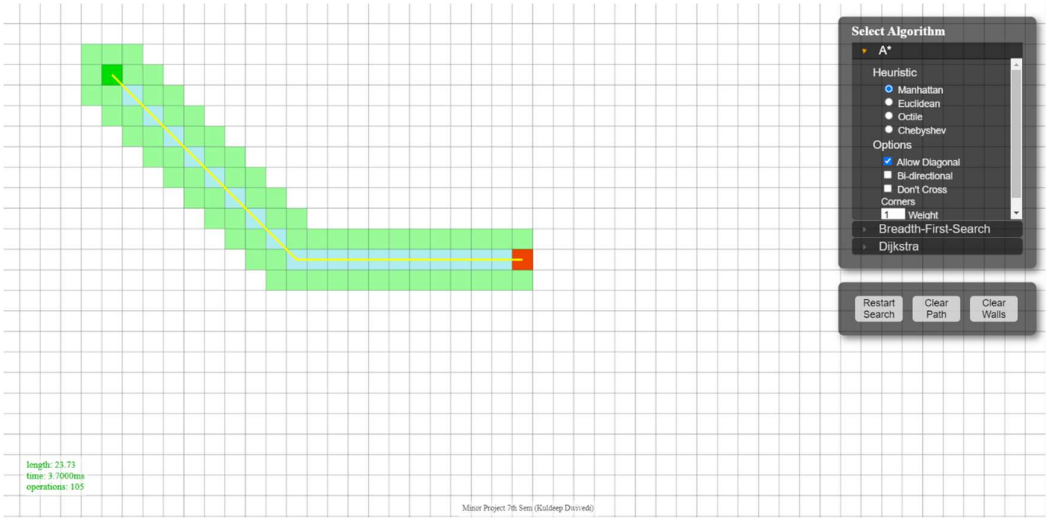


*Figure 26 Path finding using A**

3.5.4 Disadvantage of A* star algorithm

One problem with search-based algorithms such as A* is that they become very computationally expensive as the size and dimensions of the state space increases. We can imagine how the number of grids points grow exponentially as the number of dimension increase which can slow everything down. Therefore these algorithms are not to be used for high dimensional state space for eg things like determining the path for multi-joined robot manipulators or for really large dimensional state space ones that might have millions or more grid cells. For such problems Sampling based algorithms are useful.

Other issue with A* algorithm is that it tends to gives straight paths. Straight path creates issue for a robot when it is trying to take turn.

Considering the disadvantages of A* algorithm we have use two more search algorithms which are Breadth First search (BFS) and Dijkstra Algorithms so that we don't have to depend on only one algorithm. If one fails to give path other will give us the path.

3.6 Breadth First Search

The breadth first search (BFS) is a fundamental search algorithm used to explore nodes and edges of a graph. It runs with a time complexity of O(V+E) and is often used as a building block in other algorithms.

The BFS algorithm is particularly useful for one thing: finding the shortest path on unweighted graph.

A BFS starts at some arbitrary node of a graph and explores the neighbor node first, before moving to the next level neighbors.

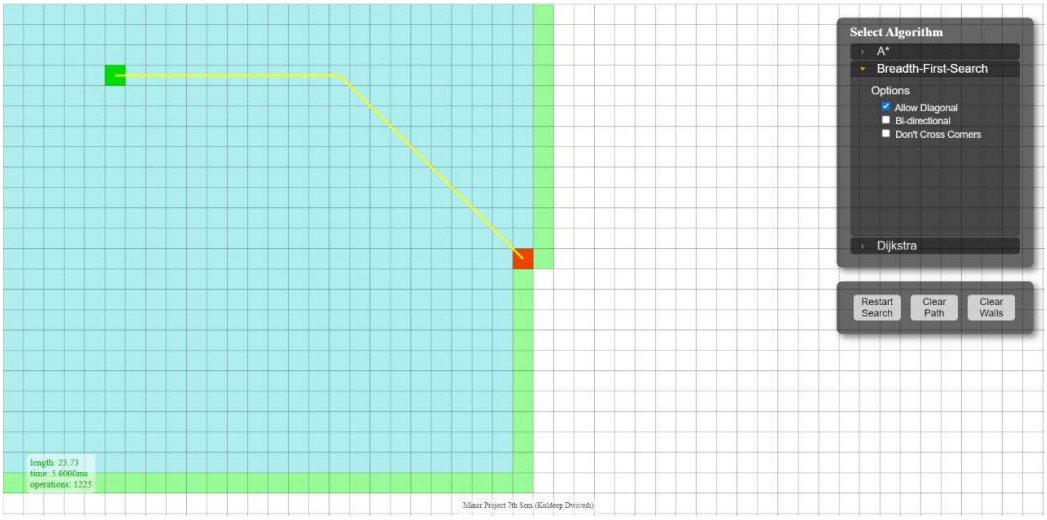


Figure 27 Path finding using BFS

3.7 Dijkstra algorithm

Dijkstra's algorithm is a single source shortest path algorithm which is used for graphs with non-negative edge weights.

Depending on how the algorithm is implemented and what data structures are used the time complexity is typically o(E*Log(V)) which is competitive against other shortest path algorithms.

One constraint for Dijkstra's algorithm is that the graph must only contain non-negative edge weights. This property is especially important because it enables Dijkstra's algorithm to act in a greedy manner by always selecting the next most promising node.

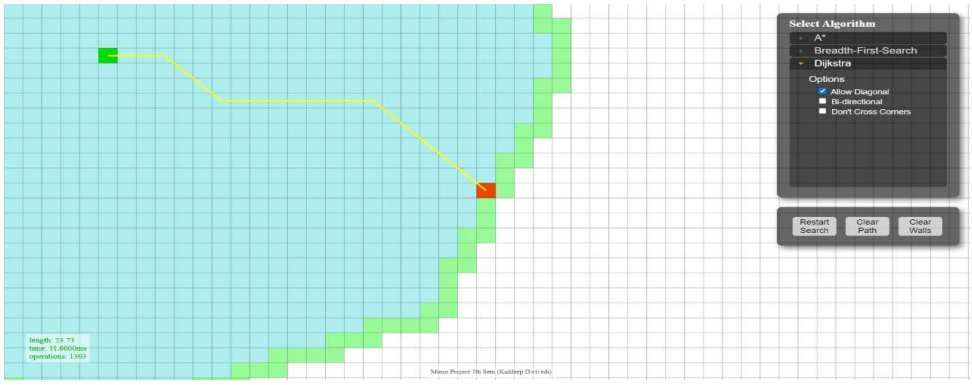


Figure 28 Path finding using Dijkstra's algorithm

Flow Chart

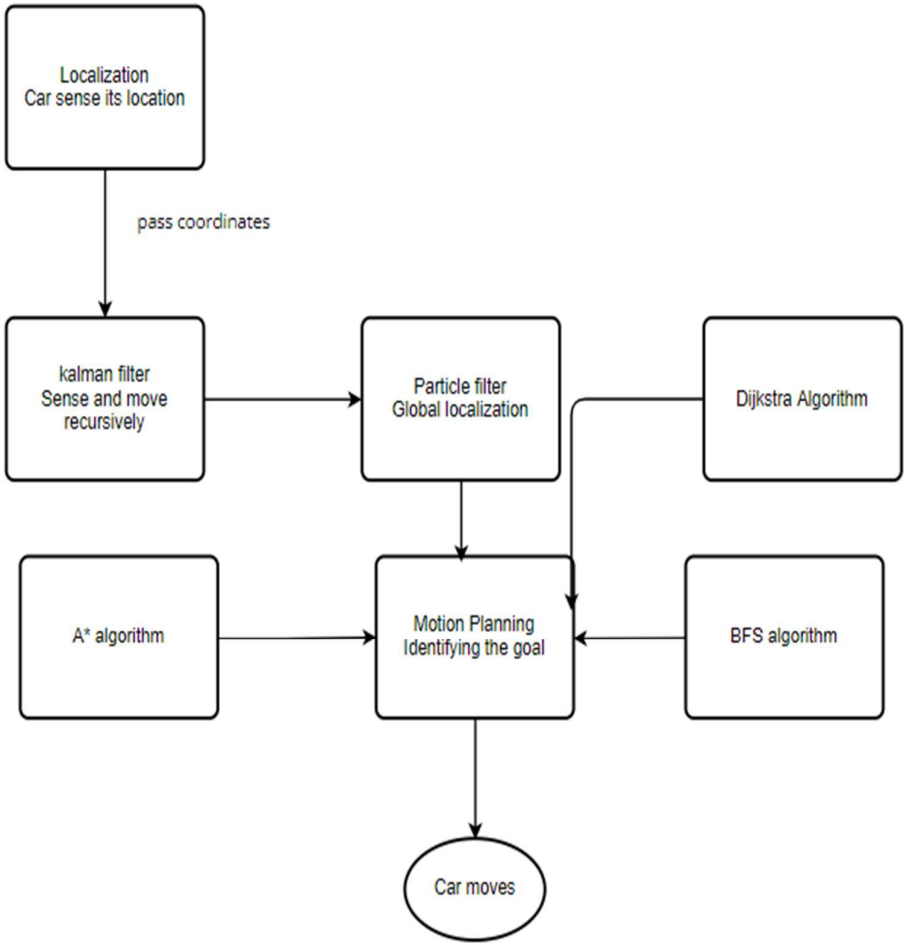


Figure 29 Flow chart

3.8 Future work and improvements

For motion planning we can use **sampling-based methods** such as RRT (Rapidly-exploring random trees) and RRT* algorithms. These algorithms will not necessarily give us optimal path every time but these algorithms are faster than A*, BFS and Dijkstra.

Modern day self driving cars such as tesla uses Neural Network along with dozens of sensors to perform motion planning.

3.9 PID Controller

3.9.1 Introduction

Robots and self-driving cars, in general, do not manoeuvre with flawless precision. Their trajectory can be influenced by their surroundings (such as non-flat surfaces) as well as minor mechanical misalignments (such as mis-aligned wheels wheels).

A human driving a car with erroneously aligned tyres may constantly self-correct to ensure that they are on the straight and narrow. How can a self-driving automobile be taught to do the same?

Assuming we have a desired path for the automobile to follow, we can use the PID Control method to assure that it will re-align itself to the target.

To describe what PID controllers are and why we use them. Let us first see what a controller is. One way to write an open-loop system is like this where inputs act on a plant or the system to be controlled and then some output signal is generated. Often in these open-loop systems the performance of the system isn't good enough to meet the requirements.

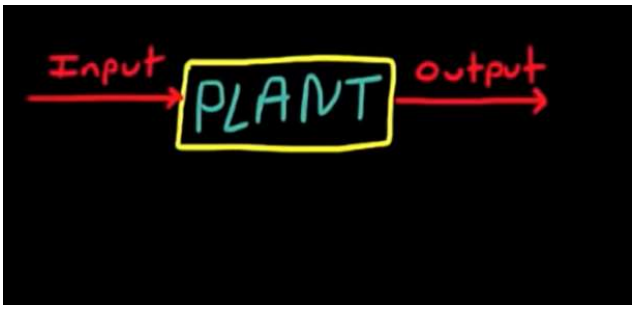


Figure 30 PID controller example

Lets take a example suppose we are a task to build a robot that needs to move from one spot to the red spot as shown in fig 7.2. we could figure out how fast the robot moves and then from that we can determine how long the robot needs to drive for before it gets to the red X. This type of system is called open loop system because the amount of time the robot drives is not adjusted based on the actual position of the robot. Open loop commanding is perfectly fine for systems that don't change much or where accuracy isn't important.

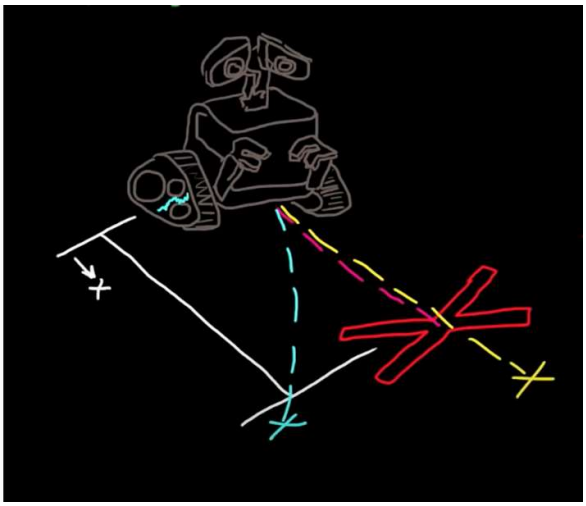


Figure 31 Robot motion in a plane

However in some case if there's some dirt in one of the wheels and robot veers off to the side or if the robot moves slightly faster than you had predicted in both of these cases the robot wont stop on the red X because it has no way of compensating for these errors and making adjustment on its own. The solution for this is **feedback control** which essentially means you're sensing the output of the plant and feeding it back to the system so that system can make adjustments appropriately. In a feedback system there is a Reference signal and this is the desired value or the ultimate goal, we can compare the desired value to the measured value and now what we are left with is the error or the delta between when the robot is and where we want the robot to be. In this case error would be the difference from the reference position which is shown as the distance from the start to the red X (in meters) and the current position of the robot which is measured by the robot (in meters).

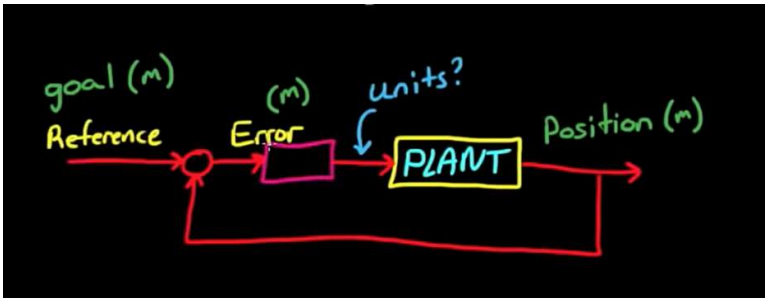


Figure 32 PID controller

Now at this point we have to find out how we can convert an error signal which as same units as the output of the plant into an input signal that has units that may or may not be same as the output. Apart from changing the units, the error needs to be adjusted in

such a way that the input into the robot causes it to eventually reach the Red X. This is exactly what a controller does. A controller takes the error signal and convert it into a command that is then sent into that plant. One of the goals of a control engineer is to design this controller so that as time passes the error or the difference between the current location and the goal is driven to zero. Zero error means that the measured position is exactly where we want it to be which means that the system meets all its requirements.

There are many type of controllers PID is one of the most common controller. We have used implemented PID controller in our project. PID is widely used because it is efficient and effective in a wide variety of applications and in fact it is the majority of controller types and industrial applications.

3.9.2 PID

PID is an acronym and it stand for proportional internal derivative and each of these terms describe how the error term is treated before being summed and sent into the plant.

A PID controller constantly evaluates an error value which is defined as the difference between a desired point and a measured process variable. The controller in a way tries to minimize the error over time by variation of a control variable, for e.g. by changing the position of a control valve, a damper, or the power given to a heating element, to a new value determined by a weighted sum:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d_e(t)}{dt}$$

Here K_p, K_i and K_d are all non-negative constants and these denote the coefficients for the proportional(P), integral(I), and derivative terms(D), respectively.

In this model-

• P accounts for the error's current values. If the error is high and positive, for example, the control output will be large and positive as well.

• I account for previous incorrect values. If the current output isn't strong enough, for example, errors will build up over time, and the controller will respond by taking a stronger action.
• D takes into consideration the error's potential future values depending on its current rate of change.

PID controllers are widely applicable since they rely solely on the measured process variable and not on knowledge of the underlying process. [2] A PID controller can deal with specific process requirements by tuning the three parameters of the model. The controller's responsiveness to an error, the degree to which the system overshoots a setpoint, and the degree of any system oscillation can all be defined in terms of the controller's response. The use of the PID algorithm does not guarantee that the system will be controlled optimally or that it will be stable.

To offer the proper system control, some applications may only require the use of one or two words. The other parameters are set to 0 to achieve this. In the absence of the appropriate control actions, a PID controller is referred to as a PI, PD, P, or I controller. Because derivative action is susceptible to measurement noise, and the absence of an integral term may prevent the system from attaining its target value, PI controllers are rather prevalent.

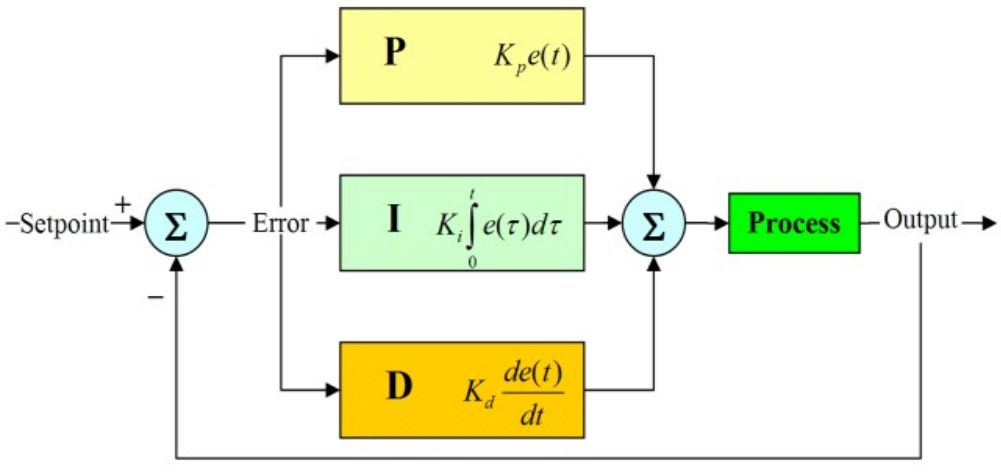


Figure 33 PID Controller block diagram

In one of its block diagrams forms a PID controller is written like this as shown in Figure 7.4. In the proportional path the error term in multiplied by a constant Kp, in the integral path the error term is multiplied by a constant Ki and then integrate it. In the derivative path its multiplied by Kd and then differentiate it. Then these three terms are summed together to produce the controller output. The three K terms are called gains, and these can be adjusted or tuned to a particular plant with a defined set of requirements. By changing these we are adjusting how sensitive the system is to each of these different paths.

Let's understand this concept with few plots (fig 7.5)

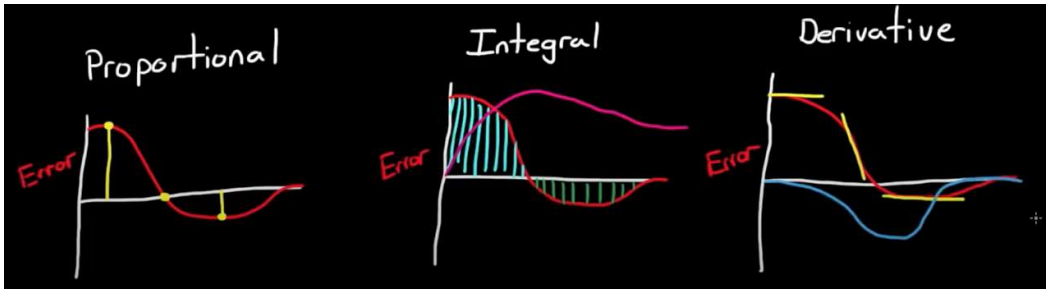


Figure 34 P, I and D vs error plots

The error in the system is changing over time like the red line. In proportional path the output is the error scaled by constant Kp. As we can see when the error is large the proportional path will produce a large output. When the error is zero the output in the path is zero and when it is negative the output is negative.

In the integral path as the error moves over time the integral will continually sum, it up and multiply it by constant Ki. In the plot as we can see that the integral path is the area under the curve where blue is the positive area and green is negative area. Integral path is used to remove constant errors in a control system since no matter how small the constant error, eventually the summation of that error will be significant enough to adjust the controller output.

In the derivative path , the rate of change the error contributes to the output signal. When the change in error is moving relatively slowly like it is at the beginning then the derivative path is small and faster the error changes the larger the derivative path becomes.

To get the output of PID controller we can sum these three paths. However we don't always need all paths so we can remove a path completely by setting its associated gains to zero. When we do this we generally refer to that controller with the letters of the path that are left for example we can have a proportional integral controller PI if we set Kd to zero and just a P controller if Both Kd and Ki are set to zero. We do this because this will controller simple , a simple controller is easy to implement, easy to tune and easy to troubleshoot.

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 35 Localization

4.2 Kalman Filter

For the Google self-driving car this is the essential model for how it makes predictions about where other cars are and how fast they are going. The car uses radar on the front bumper that measures the distance to other vehicles and also gives a noisy estimate of the velocity. Additionally, the car uses its lasers, which are located on the roof, to measure the distance to other cars and gets a noisy estimate of the velocity.

So, when the Google car is on the road it uses radars and lasers to estimate the distance and the velocity of other vehicles on the road using a Kalman filter, where it feeds in range data from the laser and uses state spaces of the relative distance, x and y, and the relative velocity of x and y.

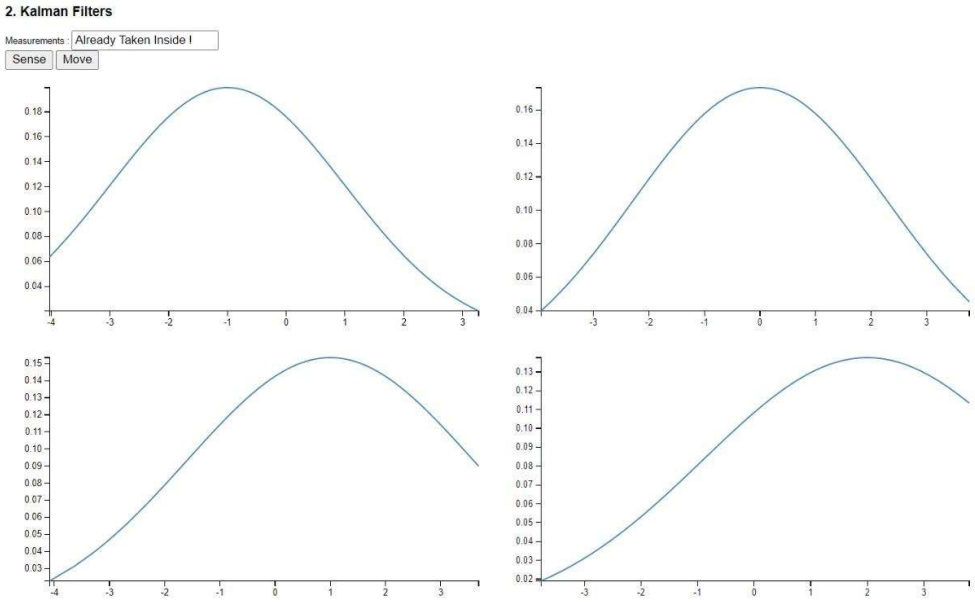


Figure 36 Kalman FIlter

4.3 Motion Planning

One we have located the robot or the car in the environment next step is to find the path so that car can perform motion along that path. Motion planning is defined as the process of finding path from starting position to final position in an environment. We have use used three algorithms to perform motion planning. The algorithms are A star , Breadth First Search and Dijkstra.

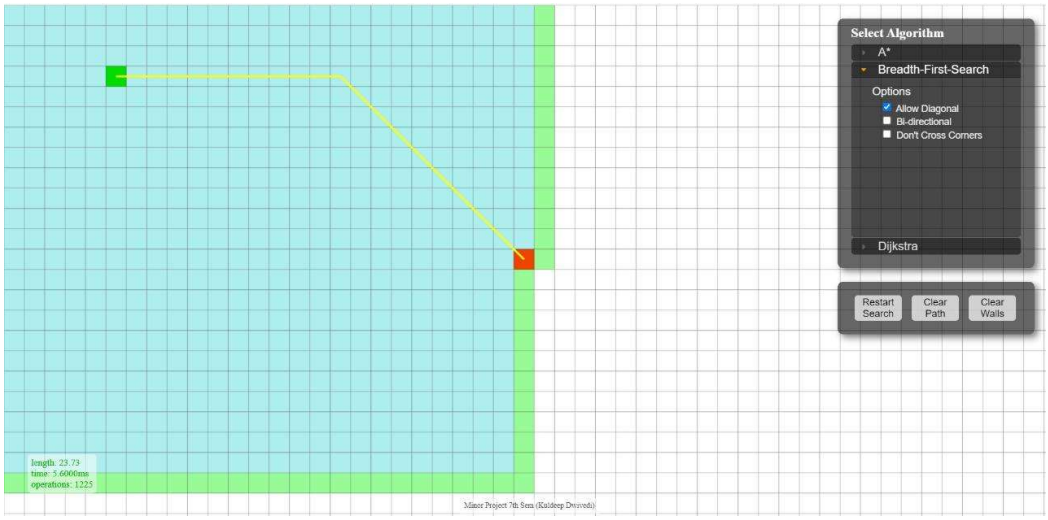


Figure 37 Motion planning 1

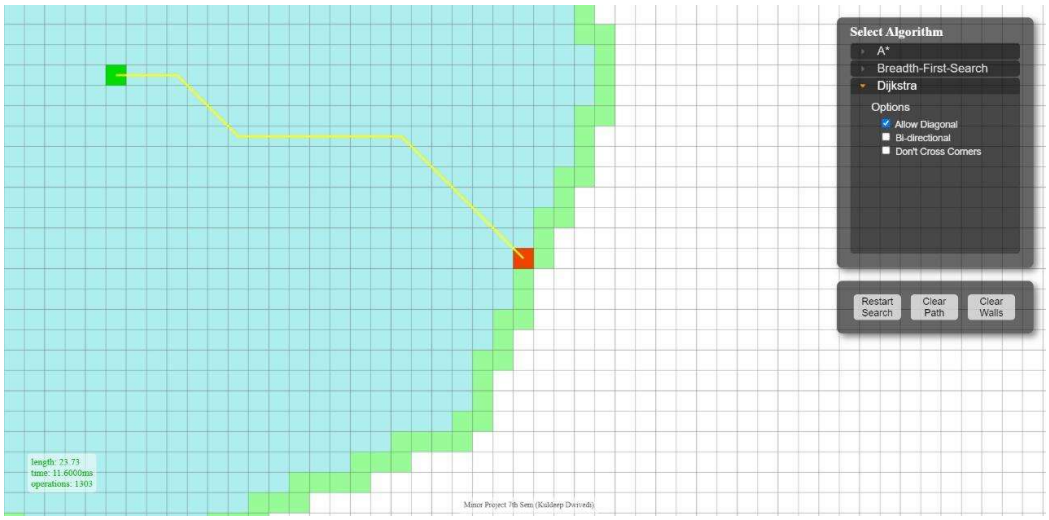


Figure 38 Motion Planning 2

4.4 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.

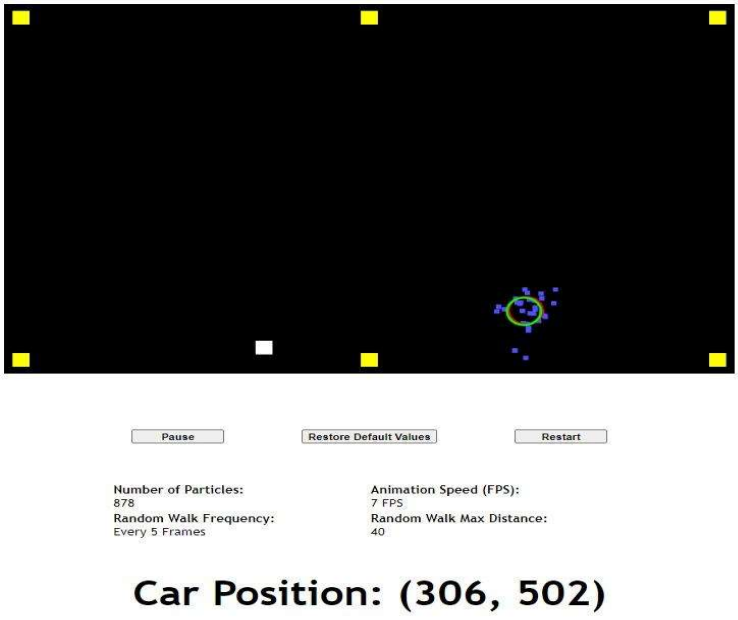


Figure 39 Particle Filter

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 recognizing 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

Appendix

MINOR PROJECT

CHAPTER 1 – INTRODUCTION

AI research is highly technical and it is deeply divided into subfields that often fail to communicate with each other. Some of the division is due to social and cultural factors: subfields have grown up around particular institutions and the work of individual researchers. AI research is also divided by several technical issues. Some subfields focus on the solution of specific problems. Others focus on one of several possible approaches or on the use of a particular tool or towards the accomplishment of particular applications.

The dynamic path finding tool is quite beneficial because it provides real-time navigation instructions based on the car's current location. When compared to Dijkstra and BFS, the A* approach, which is used to implement paths based on current and final locations, is extremely fast. We use a PID controller to discover the smoothest approach to the objective because A* only delivers square paths.

Self-driving car is very sophisticated technology allows the onboard computer to do hundreds of calculations per second. These include the path from 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 huge capability for reducing traffic congestion because they are rarely involved in accidents. Not only that, but because self-driving cars can interact with one another, traffic lights would be obsolete. Better traffic coordination might result in less congestion if people drove slower but made fewer stops.

1.1 Purpose of Plan

The main purpose here 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.

And this can be achieved by 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 Background Information

"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.

1.3 Project Goals and Objectives

The aim of project is to implement A*algorithm to find dynamic path for the car to reach destination and to implement PID controller on the path derived by the algorithm to find the smoothest path for the car to reach the destination.

CHAPTER 2 – SCOPE

2.1 Scope Definition

Project scope is the most important and defining constraints of any project. We plan to develop 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.

2.2 Projected Budget

The estimated budget of the project is minimal as all that is required for the project is Visual Studio software and certain databases, all of which are easily available on internet.

CHAPTER 3 – CONSTRAINTS

3.1 Project Constraints

- The project is constrained by time restrictions. Each step of SDLC has a time limit.
- Because all of the software used are not paid software, there are certain financial limits. The software utilized is a free and trial version, which can reduce the program's efficiency and accuracy.

- Because of the various methodologies used to classify the multimedia data, it may be impossible to achieve 100% accuracy; thus, accuracy for this project may range between 70% and 80%.
- We will have to manage our time between planning the project and implementing it simultaneously. Since it is a group project the team members will have to sync their clock accordingly.

CHAPTER 4 – PROJECT MANAGEMENT TECHNIQUE

4.1 Project Timeline

WEEK	TASK
1	Search for the topic.
2	Research in the particular field.
3	Reading the research papers.
4	Reading the research papers.
5	Finalizing the Topics.
6	Understanding Artificial Intelligence.
7	Understanding Various AI techniques.
8	Gathering Requirements.
9	Making rough plan/timeline.
10	Discussing various ideas.
11	Exploring scope.
12	Analyzing risks.

4.2 Project Roles and Responsibilities

Responsibilities	Participants

Localization and Particle Filter Kuldeep Dwivedi
Motion Planning and PID Controller Akshaya Kumar Raghav
Kalman FiltersMohit
4.3 Risk Management

4.3.1 Project's Risk Assessment

Risk in Project	Risk Level	Event/Activity	Mitigation Strategy
Testing Time	Medium	Testing time can exceed the expected time limit.	Use paid tools to get efficient results.
Time	Medium	It can take a lot of time to train the datasets.	Limit the size of the datasets to minimal without losing accuracy.

4.4 Future Scope
The software used will play an important role in the manufacturing of the self-driving car. The approaches utilized for particle tracking, motion planning, and smoothing minimize error rates to the greatest extent possible, manufacturing in the manufacturing a dependable self-driving automobile. A lot goes into implementing the actual car itself, aside from the software implementation.
Efforts can be made in the future to improve the error rates so that these automobiles can be used in ordinary traffic.

CHAPTER 5 – LITERATURE REVIEW

2.1 Introduction
Machine intelligence is known as artificial intelligence (AI). An ideal machine in computer science is a flexible, rational agent that analyze its surrounding and makes decisions to maximize its chances 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 likely to be used. AI contains a negative connotation, especially among the general public, because it is associated with "cutting-edge" machines (or even "mysterious"). This subjective line between what defines "artificial intelligence" continues to blur with time; for example, optical character recognition is no longer considered an exemplar of "artificial intelligence," since it is now considered a commonplace routine 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.
Artificial intelligence research is very technical and specialized, and it is divided in numerous subparts which frequently can't able to interact with one another. Some of were divided based on social and cultural factors: subparts have developed specific institutions and researchers' activity. Several technological difficulties also split AI research. Some subfields concentrate on solving specific challenges. Others concentrate on the alternative approaches, to use a specific instrument, or the completion of specific tasks.

2.2 Goals
The regular problem of imitating intelligence had been divided into several sub-problems. These are the appropriate characteristics or skills that experts would love to

observe in an expert system. The following characteristics have gotten greatest attention.

2.2.1 Deduction, reasoning and problem solving

Initially, AI experts devised algorithms which mimicked humans' step for reasoning while solving puzzles or deducing logical conclusions. AI research had also created extremely fantastic plans for managing the uncertain or partial information, using notions from probabilistic computation and methods from economics.

Most of these techniques can require massive computing resources for tough problems – most undergo "combinatorial explosion," in which the space or time required skyrockets if problem exceeds a specific size. For AI researchers, finding efficient and more effective algorithms is a top priority.

Humans solve the majority of their issues by making quick, intuitive decisions rather than the deliberate, step-by-step deductions that early AI research could replicate. Embodied agent methods highlight the significance of sensorimotor skills to higher thinking; neural net research aims to emulate mechanisms inside the human mind which increases the talent; statistical side to AI copy the probabilistic character of the human ability.

2.2.2 Knowledge representation

Knowledge is represented by an metaphysics as a set of rule around a subject and their relationships. Common sense knowledge and knowledge representation are of primary importance. Research in AI revolves around representation of the knowledge and engineering knowledge. Machines are supposed to address several problems that will necessitate a broad understanding of the world. Objects, attributes, categories, interactions among objects; states, time; reasons and consequences; meta knowledge; and many more, less extensively investigated, areas are something that AI must represent. An ontology is a representation of "what exists": the collection of things, relations, concepts, and so that the computer is aware of. Upper ontologies are most common, attempting to offer a base for every other knowledge.

2.2.3 Planning

Automatic arrangement and smart systems must be capable of setting and achieving objectives. They require a mechanism to envisage future and make computations that increases the utility (or "value") of the options accessible to them.

In traditional scheduling issues, the system might presume that the system is the only object operating in the ecosystem and that the effects of its actions would be predictable. If the agent is , however, it must frequently check if the ecosystem around it matches with the computation made by the system and adjust its strategy as needed, forcing the system to reason with respect to uncertainty.

To get to a result, multi-agent planning employs the collaboration and competition of multiple agents. Self-learning algorithms and intelligence both exploit emergent behaviour like this.

2.2.4 Learning

ML is the research of computer algos which develop themselves over time, and it has been at the heart of AI research from its inception.

The capacity to detect patterns in a packets of data is known as "unsupervised learning." The classification and numerical regression are part of supervised learning. It has been observed that there are the number of instances of items from various categories,

classification is used to decide which category something belongs in. In reinforcement learning, the agent is rewarded for positive replies and penalised for negative ones. The agent develops a strategy for working in its issue space based on this series of rewards and punishments. These types of learning examined using ideas like utility and decision theory. Computational learning theory is an area of theoretical computer science that deals with the mathematical study of ML algos and their performance.

2.2.5 Natural language processing (communication)

NLP enables machines to read and comprehend all kind of languages spoken by people. Information retrieval , question answering, and machine translation are some simple uses of natural language processing.

Semantic indexing is a typical way of extracting and analysing meaning from all kind of natural language. Indexing vast quantities of abstractions of the user's input has become appropriately more efficient as processing speeds have increased and the actual cost of data storage has decreased.

2.2.6 Perception

Machine perception is the capability to derive features of the world using sensor input. The ability to analyse visual information is known as computer vision. Speech recognition, facial recognition, and object recognition are a few examples of subproblems.

2.2.7 Social Expertise

The research and development of various system and devices that can recognise, process, and imitate human-like feelings is known as affective computing. Computer science and cognitive science are all part of this interdisciplinary field. While the field's roots may be traced all the way back to early philosophical explorations into emotion, Rosalind Picard's 1995 papeR on emotional computing was the start of the modern discipline of computer science. The ability to replicate empathy is the motive of research's motivations. The machine may able to read people's emotions and modify its behaviour accordingly, offering suitable responses to those emotions.

The intelligent agent's emotional & social talents serve two purposes. To begin with, it should be capable to analyse people's reaction by comprehending their motivations and emotional states of action. Furthermore, in order to promote human-computer connection, an intelligent machine might be able to exhibit emotions—even if it does not experience them—in order to look responsive to the dynamics of human interaction.

2.3 Tools Used to Implement AI

AI has generated a significant variety of tools to handle the most difficult issues in computer science over the period of 50 years of research. Below are a handful of the most common of these techniques.

2.3.1 Search and optimization

Over a period of research, Artificial Intelligence has generated a variety of tools to handle the most difficult issues in computer science. Below are a handful of the most common of these techniques. Many AI issues can theocratically be solved by examining all possible number of plausible options. Reasoning can be boiled to a search. In a process known as means-ends analysis, algorithms look through trees of objectives in an attempt to determine a path to a target goal. Many algorithms use the concept of optimization-based search methods.

For most real-world issues, simple and exhaustive searches are rarely sufficient; the search space (the number of places to seek) quickly increases to astronomical proportions. As a result, the search is either too slow or never finishes. For many

situations, the approach is to employ "heuristics" or "rules of thumb" that reject options that are unlikely to lead to the desired outcome (called "pruning the search tree"). Heuristics provide the software with a "best guess" on the solution's location. Heuristics reduces the number of searches in smaller size.
In the late 90's, a new type of searching technique emerged, which was based on the concept of mathematical theory of optimization. For all the situations, it is easy to start with a random guess and then revise it incrementally until there are no more refinements available. This can be compared to blind hill climbing. Simulated annealing, beam search, and random optimization are some of the other optimization strategies.

2.3.2 Logic
The use of Logic to convey knowledge and solve problems, and related issues. The SATplan algorithm, for example, uses logic for planning, while inductive logic programming is a learning tool. In research, several kind of logic were used. The logic of assertions that can be true or false is known as propositional or sentential logic. First-order logic also supports quantifiers and predicates, allowing for the expression about objects, their qualities, and their relationships. Fuzzy logic is a type of first-order logic in which the truth of a statement is represented as a number between 0 and 1, rather than just True (1) or False (0). Fuzzy systems are commonly employed in current industrial and consumer product control systems and can be used for uncertain reasoning. This strategy distinguishes neglegence from probabilistic statements made with great confidence by an actor.
Default logics, non-monotonic logics, and circumscription are logics that aid in default reasoning and the qualification problem. Description logics, situation calculus, event calculus, and fluent calculus (for representing events and time), causal calculus, belief calculus, and modal logics are some of the extensions of logic that have been devised to handle certain categories of knowledge.
2.3.3 Probabilistic methods for uncertain reasoning
Many AI challenge necessitate the agent working with incomplete or ambiguous data. Using concepts from probability and economics, AI researchers have built a number of sophisticated tools to handle these challenges.
Bayesian networks are versatile tools that may be applied to a number of tasks, including reasoning, planning, and perception. Filtering, prediction, smoothing, and finding explanations for packets of data can all be done with probabilistic algorithms, which aid perception systems in analysing processes that occur across time.

2.3.4 Classifiers and Statistical Methods
Classifiers and controllers are the two most basic forms of AI application. However, before inferring actions, controllers classify circumstances, and categorization is a key component of many artificial intelligence system. Pattern matching is used by classifiers to find the best match. They are extremely appealing for use in artificial intelligence since they may be modified based on examples. Observations or patterns are the terms used to describe these examples. Each pattern in supervised learning belongs to a priest class. A class might be thought of as a decision to be made. A data set is made up of all the observations and their class designations.
A classifier can be instructed in a number of ways, including ML methods. The neural network, kernel methods such as the support vector machine, k-nearest neighbour algorithm, Gaussian mixture model, naive Bayes classifier, and decision tree are the most extensively used classifiers. This classifiers' performance has been compared

across a huge variety of tasks. Choosing a good classifier for a particular situation is still more of an art than a science.

2.3.5 Neural networks

The associated collection of nodes, similar to huge network of axon in the human brain, is assigned to as a neural network. The study of Walter Pitts and Warren McCullough, a decade before the area of artificial intelligence research was created, pioneered the study of non-learning artificial neural networks. Frank Rosenblatt devised the perceptron, a single-layer learning network that is analogous to linear regression. Alexey Grigorevich Ivakhnenko, Teuvo Kohonen, Stephen Grossberg, Kunihiko Fukushima, Christoph von der Malsburg, David Willshaw, Shun-Ichi Amari, Bernard Widrow, John Hopfield, and others were among the early pioneers.

Perceptrons, multi-layer perceptrons are among the most popular feedforward networks. Using techniques like Hebbian learning, GMDH, or competitive learning, neural networks can be used to solve problems like intelligent control.

Today, neural networks are frequently trained using the backpropagation algorithm, which was first proposed by Seppo Linnainmaa in 1970 as the reverse mode of automatic differentiation and was later applied to neural networks by Paul Werbos.

2.3.6 Control theory

Control theory is a discipline of mathematics and engineering which studies the behaviour of dynamical systems having inputs as well as how feedback affects their behaviour.

2.3.7 Languages

AI researchers have developed several specialized languages for artificial intelligence research, including Lisp and Prolog.

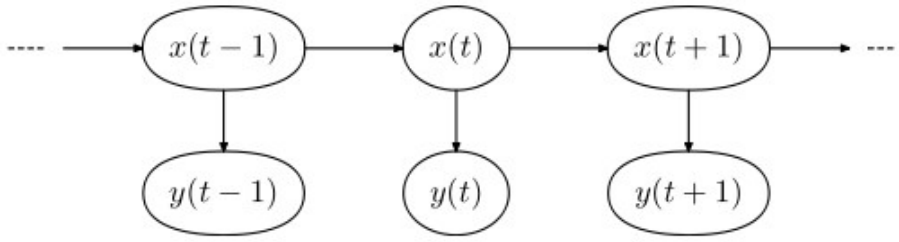
LOCALIZATION

Introduction

In order to drive the car must first know its location within its environment however it is not possible to use an instrument to measure exactly where a driverless car is. Even the best GPS sensors have a margin of error of around 5 meters (and think about what that means for a car given that the standard lane width is around 3.5 meters). Finding where a robot is when it can't measure location directly is called localization. There are several AI techniques that allow a driverless car to incorporate temporal, noisy input to generate a probabilistically sound estimate.

For localization, the underlying model is generally some form of Bayesian model similar to a hidden markov model where the state space of the the unknown "location" variables are continuous. At each time point *t* there are two variables: an unknown variable which is the location of the car, *x(t)*, and observations about the car's location based on the sensor inputs at that given time, *y(t)*. The model assumes that x(t) is generated from x(t - 1) with some unknown distribution and that y(t) is generated from x(t) with some unknown distribution.

Bayesian Model



It mainly used a particle filters algorithm, a randomized algorithm which repeatedly samples possible scenarios, to come up with a best estimate for the where it is: in the diagram above the variables *x(t)*.

3.2 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: the

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."

KALMAN FILTER

Introduction

A Kalman filter is an optimal estimator - i.e. infers parameters of interest from indirect, inaccurate and uncertain observations. It is recursive so that new measurements can be processed as they arrive. (cf batch processing where all data must be present).

Optimization meaning: If all noise is Gaussian, the Kalman filter minimizes the mean square error of the estimated parameters.

What if the noise is NOT Gaussian?

Given only the mean and standard deviation of noise, the Kalman filter is the best linear estimator.

Non-linear estimators may be better.

Why is Kalman Filtering so popular?

- Good results in practice due to optimality and structure.
- Convenient form for online real time processing. ·
- Easy to formulate and implement given a basic understanding.

- Measurement equations need not be inverted.

Why use the word "Filter"?

The process of finding the "best estimate" from noisy data amounts to "filtering out" the noise.

However a Kalman filter also doesn't just clean up the data measurements, but also projects these measurements onto the state estimate.

Co Variance:

Co variance is determined as:

The covariance of two random variables x_1 and x_2 is

$$\begin{aligned} \mathrm{cov}(x_1, x_2) &\equiv E[(x_1 - \bar{x}_1)(x_2 - \bar{x}_2)] \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x_1 - \bar{x}_1)(x_2 - \bar{x}_2)p(x_1, x_1)dx_1 dx_2 \\ &\equiv \sigma^2_{x_1 x_2} \end{aligned}$$

where p is the joint probability density function of x_1 and x_2.

The ***correlation coefficient*** is the normalised quantity

$$\rho_{12} \equiv \frac{\sigma^2_{x_1 x_2}}{\sigma_{x_1}\sigma_{x_2}}, \quad -1 \le \rho_{12} \le +1$$

The covariance of a *column vector* $\mathbf{x}=[x_1 .. x_n]'$ is defined as

$$\begin{aligned} \mathrm{cov}(\mathbf{x}) &\equiv E[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})'] \\ &= \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})'\, p(\mathbf{x})\, dx_1 .. dx_n \\ &\equiv \mathbf{P}_{\mathbf{xx}} \end{aligned}$$

and is a *symmetric n* by *n* matrix and is *positive definite* unless there is a linear dependence among the components of $\mathbf{x}$.

The $(i,j)^{th}$ element of $\mathbf{P}_{\mathbf{xx}}$ is $\sigma^2_{x_i x_j}$

Interpreting a covariance matrix:

diagonal elements are the variances, off-diagonal encode correlations.

Diagonalising a Covariance Matrix :

cov(x) is symmetric => can be diagonalised using an orthonormal basis. By changing coordinates (pure rotation) to these unity orthogonal vectors we achieve decoupling of error contributions. The basis vectors are the eigenvectors and form the axes of error ellipses. The lengths of the axes are the square root of the eigenvalues and correspond to standard deviations of the independent noise contribution in the direction of the eigenvector.

KALMAN FILTER DEFINITION

We require discrete time linear dynamic system description by vector difference equation with additive white noise that models unpredictable disturbances.

STATE DEFINITION -

The state of a deterministic dynamic system is the smallest vector that summarises the past of the system in full. Knowledge of the state allows theoretically prediction of the future (and prior) dynamics and outputs of the deterministic system in the absence of noise.

PARTICLE FILTER

Introduction

In recent years, particle filters have solved several hard perceptual problems in robotics. Early successes of particle filters were limited to low-dimensional estimation problems, such as the problem of robot localization in environments with known maps.

More recently, researchers have begun exploiting structural properties of robotic domains that have led to successful particle filter applications in spaces with as many as 100,000 dimensions. The fact that every model—no mater how detailed—fails to capture the full complexity of even the most simple robotic environments has lead to specific tricks and techniques essential for the success of particle filters in robotic domains. This section covers some of these recent innovations, and provides pointers to in-depth articles on the use of particle filters in robotics.

Particle filters address the more general case of (nearly) unconstrained Markov chains. The basic idea is to approximate the posterior of a set of sample states x belonging to Xi, or particles. Here each is a concrete state sample of index i , where i's range is the size of the particle filter.

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 | xi) .

Call the resulting set of particles Xt.

Particle filters are attractive to robot cists for more than one reason. First and foremost, they can be applied to almost any probabilistic robot model that can be formulated as a Markov chain. Furthermore, particle filters are anytime such that they do not require a fixed computation time; instead, their accuracy increases with the available computational resources. This makes them attractive to roboticists, who often face hard real-time constraints that have to be met using hard-to-control computer hardware.

Finally, they are relatively easy to implement. The implementer does not have to linearize non-linear models, and worry about closed-form solutions of the conditional statements of different form , as would be the case in Kalman filters, for example. The main criticism of particle filter has been that in general, populating a d-dimensional space requires exponentially many particles in d.

Most successful applications have therefore been confined to low-dimensional state spaces. The utilization of structure (e.g., conditional independences), present in many robotics problems, has only recently led to applications in higher dimensional spaces.

PARTICLE FILTERS IN LOW DIMENSIONAL SPACES

In robotics, the 'classical' successful example of particle filters is mobile robot localization. Mobile robot localization addresses the problem of estimation of a mobile robot's pose relative to a given map from sensor measurements and controls. The pose is typically specified by a two-dimensional Cartesian coordinate and the robot's rotational heading direction.

The problem is known as position tracking if the error can be guaranteed to be small at all times. More general is the global localization problem, which is the problem of localizing a robot under global uncertainty. The most difficult variant of the localization, however, is the kidnapped robot problem, in which a well-localized robot is tele-ported to some other location without being told. This problem was reported, for example, in the context of the Robocup soccer competition, in which judges picked up robots at random occasions and placed them somewhere else. Other localization problems involve multiple robots that can observe each other.

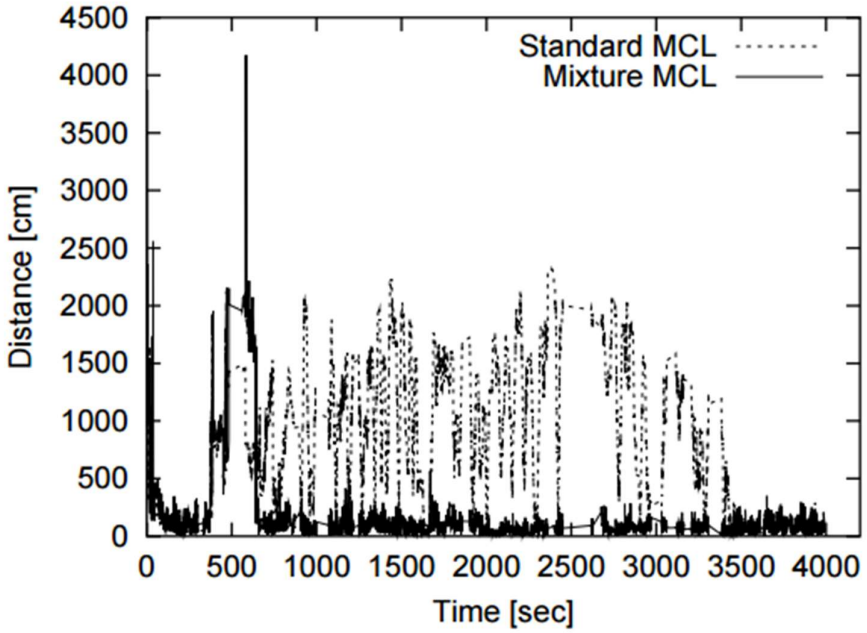


Figure illustrates particle filters in the context of global localization of a robot in a known environment. Shown there is a progression of three situations, in which a number of particles approximate the posterior (1) at different stages of robot operation. Each particle is a sample of a three dimensional pose variable, comprising the robot's Cartesian coordinates and its orientation relative to the map. The progression of snapshots in Figure 1 illustrate the development of the particle filter approximation over time, from global uncertainty to a well-localized robot.

In the context of localization, particle filters are commonly known as Monte Carlo localization (MCL). MCL's original development was motivated by the condensation algorithm, a particle filter that enjoyed great popularity in computer vision applications. In most variants of the mobile localization problem, particle filters have been consistently found to outperform alternative techniques, including parametric probabilistic techniques such as the Kalman filter and more traditional techniques.

PARTICLE FILTERS IN HIGH DIMENSIONAL SPACES

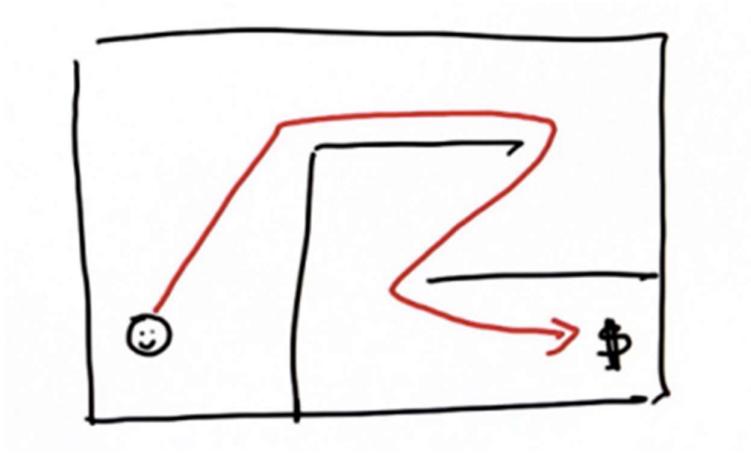
An often criticized limitation of plain particle filters is their poor performance in higher dimensional spaces. This is because the number of particles needed to populate a state space scales exponentially with the dimension of the state space, not unlike the scaling limitations of vanilla HMMs. However, many problems in robotics possess structure

that can be exploited to develop more efficient particle filters. One such problem is the simultaneous localization and mapping problem, or SLAM [8, 27, 36, 45]; see [48] for an overview. SLAM addresses the problem of building a map of the environment with a moving robot. The SLAM problem is challenging because errors in the robot's localization induce systematic errors in the localization of environmental features in the map. The absence of an initial map in the SLAM problem makes it impossible to localize the robot during mapping using algorithms like MCL. Furthermore, the robot faces a challenging data association problem of determining whether two environment features, observed at different point in time, correspond to the same physical feature in the environment. To make matters worse, the space of all maps often comprises hundreds of thousand of dimensions. In the beginning of mapping the size of the state space is usually unknown, so SLAM algorithms have to estimate the dimensionality of the problem as well. On top of all this, most applications of SLAM require real-time processing.

MOTION PLANNING

Introduction

The fundamental problem in motion planning is that a robot might live in a world that looks like the one pictured below and will want to find a goal like $. The robot has to have a plan to get to its goal, as indicated by the red line route.



This same problem occurs for a self-driving car that lives in a city and has to find its way to its target location by navigating through a network of streets and along the highway.

Given:

- Map Starting Location
- Goal Location Cost (time it takes to drive a certain route)
- **Goal**: Find the minimum cost path.

Computing Cost (Compute Cost) :

Suppose you live in a discrete world that is split up into grid cells and your initial location is facing forward, while your goal location is facing to the left.

Assume that for each time step you can make one of two action -- either move forward or turn the vehicle. Each move costs exactly one unit.

Writing a Search Program (First Search Program)

Now, the question is, can you write a program that computes the shortest path from the start to the goal? The first step towards writing this program is to name the grid cells. Across the top, name the columns zero to five, and along the side you can name the rows zero to four. Think about each grid cell as a data point, called a node and the goal point is called the goal node.

1. From here you will make a list of nodes that you will be investigating further -- that you will be expanding. Call this list open. The initial state of this list is [0,0]. You may want to check off the nodes that you have already picked.
2. Now, you can test whether this node is your final goal node -- which, just by looking at the grid you can tell that it obviously is not. So, what you want to do next is to expand this node by taking it off the open list and looking at its successors
3. Then, you can check those cells off on your grid. Most importantly, remember to note how many expansions it takes to get to the goal -- this is called the g-value. By the end of your planning, the g-value will be the length of the path
4. The process of expanding the cells, starting with the one with the lowest g-value, should yield the same result you found as the number of moves it takes to get from the start to the goal.

6.2 A* search algorithm (A Star) :

The A* (A-star) algorithm can be used for our path finding problem. It is a variant of the search algorithm that we implemented, that is more efficient because you don't need to expand every node. The A* algorithm was first described by Peter Hart, Nils Nilsson and Bertram Raphael in 1968. If you understand the mechanism for searching by gradually expanding the nodes in the open list, A* is almost the same thing, but not quite.

A* uses a heuristic function, which is a function that has to be set up. If it's all zeroes, A* resorts back to the search algorithm already implemented. If we call the heuristic function h, then each cell results in a value.

PID CONTROLLER

Introduction

A **proportional–integral–derivative controller** (**PID controller**) is a control loop feedback mechanism (controller) commonly used in industrial control systems. A PID controller continuously calculates an *error value* as the difference between a desired setpoint and a measured process variable. The controller attempts to minimize the error over time by adjustment of a *control variable*, such as the position of a control valve, a damper, or the power supplied to a heating element, to a new value determined by a weighted sum:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

Where K_p, K_i, and K_d, all non-negative, denote the coefficients for the proportional, integral, and derivative terms, respectively (sometimes denoted *P, I,* and *D*). In this model,

- *P* accounts for present values of the error. For example, if the error is large and positive, the control output will also be large and positive.
- *I* accounts for past values of the error. For example, if the current output is not sufficiently strong, error will accumulate over time, and the controller will respond by applying a stronger action.
- *D* accounts for possible future values of the error, based on its current rate of change.[1]

As a PID controller relies only on the measured process variable, not on knowledge of the underlying process, it is broadly applicable.[2] By tuning the three parameters of the model, a PID controller can deal with specific process requirements. The response of the controller can be described in terms of its responsiveness to an error, the degree to which the system overshoots a setpoint, and the degree of any system oscillation. The use of the PID algorithm does not guarantee optimal control of the system or even its stability.

Some applications may require using only one or two terms to provide the appropriate system control. This is achieved by setting the other parameters to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions. PI controllers are fairly common, since derivative action is sensitive to measurement noise, whereas the absence of an integral term may prevent the system from reaching its target value.

7.2 PID Control Theory

The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$ as the controller output, the final form of the PID algorithm is:

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt}$$

Where,

K_p: Proportional gain, a tuning parameter

K_i: Integral gain, a tuning parameter

K_d: Derivative gain, a tuning parameter

e: Error $= SP - PV$

SP: Set Point

PV: Process Variable

t: Time or instantaneous time (the present)

τ: Variable of integration; takes on values from time 0 to the present t.

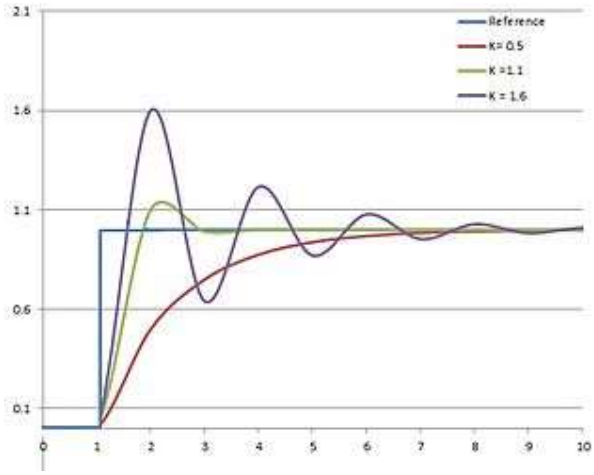
Equivalently, the transfer function in the Laplace Domain of the PID controller is

$L(s) = K_p + K_i/s + K_d s$

where

s: complex number frequency

Proportional term



Plot of PV vs time, for three values of K_p (K_i and $K_d$held constant)

The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain constant.

The proportional term is given by:

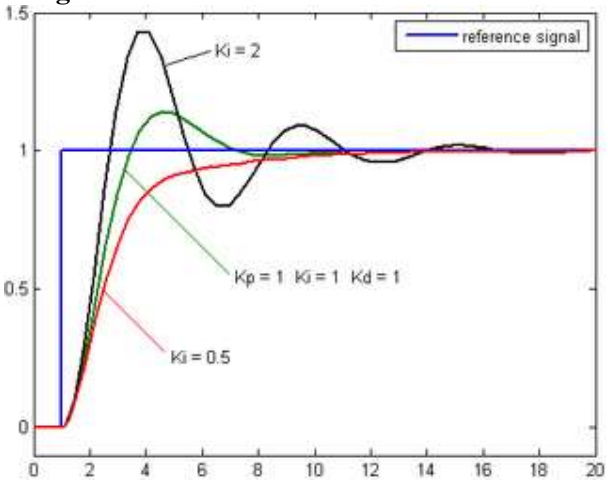
$$P_{\text{out}} = K_p e(t)$$

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable (see the section on loop tuning). In contrast, a small gain results in a small output response to a large input error, and a less responsive or less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances. Tuning theory and industrial practice indicate that the proportional term should contribute the bulk of the output change.

Steady-state error

Because a non-zero error is required to drive it, a proportional controller generally operates with a so-called *steady-state error*. Steady-state error (SSE) is proportional to the process gain and inversely proportional to proportional gain. SSE may be mitigated by adding a compensating bias term to the setpoint or output, or corrected dynamically by adding an integral term.

Integral term



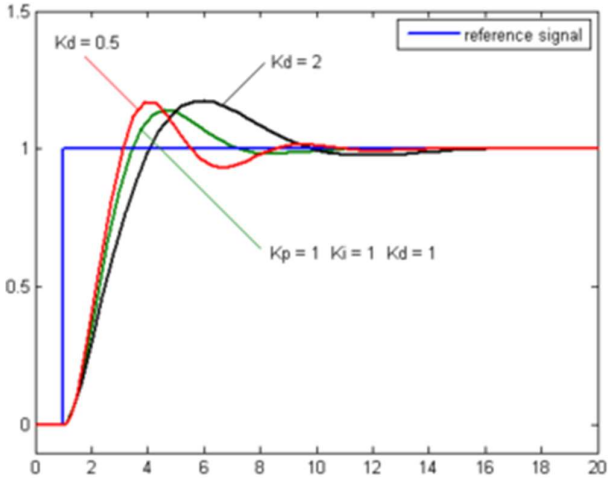
Plot of PV vs time, for three values of K_i (K_p and $K_d$held constant)

The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain (K_i) and added to the controller output.
The integral term is given by:

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

The integral term accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the setpoint value (see the section on loop tuning).

Derivative term



Plot of PV vs time, for three values of K_d (K_p and K_i held constant)

The derivative of the process error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term to the overall control action is termed the derivative gain, K_d.
The derivative term is given by:

$$D_{\text{out}} = K_d \frac{de(t)}{dt}$$

Derivative action predicts system behavior and thus improves settling time and stability of the system. An ideal derivative is not causal, so that implementations of PID controllers include an additional low pass filtering for the derivative term, to limit the high frequency gain and noise.[14] Derivative action is seldom used in practice though - by one estimate in only 25% of deployed controllers - because of its variable impact on system stability in real-world applications.

Loop tuning

Tuning a control loop is the adjustment of its control parameters (proportional band/gain, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. Stability (no unbounded oscillation) is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and requirements may conflict with one another.
PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within

the limitations of PID control. There are accordingly various methods for loop tuning, and more sophisticated techniques are the subject of patents; this section describes some traditional manual methods for loop tuning.

Designing and tuning a PID controller appears to be conceptually intuitive, but can be hard in practice, if multiple (and often conflicting) objectives such as short transient and high stability are to be achieved. PID controllers often provide acceptable control using default tunings, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning. Usually, initial designs need to be adjusted repeatedly through computer simulations until the closed-loop system performs or compromises as desired.

Some processes have a degree of nonlinearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load; this can be corrected by gain scheduling (using different parameters in different operating regions).

Stability

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e., its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by *excess* gain, particularly in the presence of significant lag.

Generally, stabilization of response is required and the process must not oscillate for any combination of process conditions and setpoints, though sometimes marginal stability (bounded oscillation) is acceptable or desired.

Mathematically, the origins of instability can be seen in the Laplace domain. The total loop transfer function is:

$$H(s) = \frac{K(s)G(s)}{1 + K(s)G(s)}$$

Where,

$K(s)$: PID transfer function

$G(s)$: Plant transfer function

The system is called unstable where the closed loop transfer function diverges for some s. This happens for situations where $K(s)G(s) = -1$. Typically, this happens when $|K(s)G(s)| = 1$ with a 180 degree phase shift. Stability is guaranteed when $K(s)G(s) < 1$ for frequencies that suffer high phase shifts. A more general formalism of this effect is known as the Nyquist stability criterion.

Optimum behavior

The optimum behavior on a process change or setpoint change varies depending on the application.

Two basic requirements are *regulation* (disturbance rejection – staying at a given setpoint) and *command tracking* (implementing setpoint changes) – these refer to how well the controlled variable tracks the desired value. Specific criteria for command tracking include rise time and settling time. Some processes must not allow an overshoot of the process variable beyond the setpoint if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new setpoint.

DrillBit

The Report is Generated by DrillBit Plagiarism Detection Software

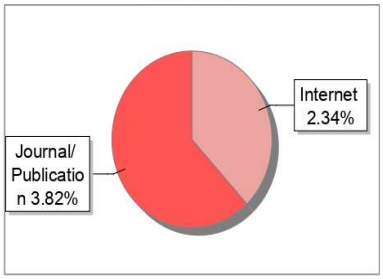
Submission Information

Author Name	153
Title	group
Paper/Submission ID	518451
Submission Date	2022-05-13 12:27:42
Total Pages	75
Document type	Project Work

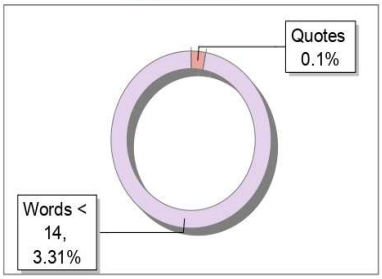
Result Information

Similarity **8 %**

Sources Type



Report Content



Exclude Information

Quotes	Not Excluded
References/Bibliography	Not Excluded
Sources: Less than 14 Words Similarity	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

8	51	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	amedleyofpotpourri.blogspot.com	1	Internet Data
2	www.actamechatronica.eu	1	Publication
3	amedleyofpotpourri.blogspot.com	1	Internet Data
4	Age and Highway Accidents by Harr-1938	1	Publication
5	Victimization immunity and lifestyle A comparative study of over-dispersed burg by Park-2016	<1	Publication
6	www.datascienceassn.org	<1	Publication
7	Robustness analysis and synthesis for nonlinear uncertain systems by Wei-Mi-1997	<1	Publication
8	Intelligent Modeling of Asphaltene Precipitation in Live and Tank Crud by Khaksa-2012	<1	Publication
9	iosrjournals.org	<1	Publication
10	en.wikipedia.org	<1	Internet Data
11	Osteoporosis Vitamin D	<1	Publication
12	amedleyofpotpourri.blogspot.com	<1	Internet Data

13	Determination of Elastic, Piezoelectric, and Dielectric Constants by - 1984	<1	Publication
14	A Review on Various Techniques used in Predicting Pollutants by Baby- 2018	<1	Publication
15	Student Paper Published in Procedia Manufacturing	<1	Internet Data
16	Reliability of Systems with Consecutive Minimal Cutsets by Shanthikumar-1987	<1	Publication
17	etd.staifas.ac.id	<1	Internet Data
18	www.biomedcentral.com	<1	Publication
19	Dynamic Local Vehicular Flow Optimization Using Real-Time Traffic Con, by Lee, Sookyoung You- 2019	<1	Publication
20	basim.xyz	<1	Internet Data
21	ACM Press the 1st ACM SIGCOMM Symposium- Santa Clara, California (, by Chen, Chen Liu, Ch- 2015	<1	Publication
22	www.readingmatrix.com	<1	Publication
23	www.javatpoint.com	<1	Internet Data
24	Posterior Cortical Atrophy with Right Lower Egocentric Quadrantic Neglect and Lo by Tariq-2020	<1	Publication
25	Leveraging Gaussian Process Regression and Many-Objective Optimization Through V by Cao-2019	<1	Publication
26	Embedded Based Miniaturized Universal Electrochemical Sensing Platform by Chen-2016	<1	Publication
27	Demand-Pull Instruments and the Development of Wind Power in Europe A Counterfa by Baudry-2018	<1	Publication

28	Abnormal induction of 3-hydroxy-3-methylglutaryl coenzyme A reductase in leukocy by Fogelman-1975	<1	Publication
29	A 4-Wire Solid-State Switching System Common Control Equipment by Nennerfelt-1964	<1	Publication
30	PDF File Data summit.sfu.ca	<1	Internet Data
31	SOUND TRANSMISSION THROUGH DRY LINED WALLS by R-1996	<1	Publication
32	Distance-Vector based Opportunistic Routing for Underwater Acoustic S, by Guan, Quansheng Ji- 2019	<1	Publication
33	IEEE 2012 50th Annual Allerton Conference on Communication, Control, by	<1	Publication
34	www.network.bepress.com	<1	Publication
35	www.network.bepress.com	<1	Publication
36	www.irdindia.in	<1	Publication
37	www.dx.doi.org	<1	Publication
38	Understanding attitudes to priorities at side road junctions, by Flower, Jonathan P- 2019	<1	Publication
39	The evolving roles of pericyte in early brain injury after subarachnoid hemorrha by Chen-2015	<1	Publication
40	PDF File Data - dai.fmph.uniba.sk	<1	Internet Data
41	Model-based cell number quantification using online single-oxygen sensor data fo by Lambrechts-2014	<1	Publication
42	docs.rwu.edu	<1	Publication
43	Die Stecknadel im Heuhaufen by R-2003	<1	Publication

44	Descriptive dynamic models for goal-directed arm movements by D-1984	<1	Publication
45	Carework, gender inequality and technological advancement in the age of Covid by MacLeavy-2020	<1	Publication
46	An experimental model for slopes subject to weathering by Voulgari-2015	<1	Publication
47	IEEE 2018 15th IEEE Annual Consumer Communications Networking Con, by AlQerm, Ismail Shi- 2018	<1	Publication
48	IEEE 2012 IEEE 33rd Real-Time Systems Symposium (RTSS) - San Juan, P by	<1	Publication
49	Smart Urban Mobility Innovations A Comprehensive Review and Evaluation by Butler-2020	<1	Publication
50	Methods for robot localization on a map by Shikhman-2019	<1	Publication
51	Detecting Financial Restatements Using Data Mining Techniques by Dutta-2017	<1	Publication

COMMENT BY EXTERNAL EXAMINER

Name and Signature of the External Examiner: