A PROJECT REPORT ON LOCALIZATION OF AN AUTONOMOUS VEHICLE

MEMBERS

Kushal Koshti
Yash Kalal
Chaitanya Venkat
Raj Nath Sinha
Jainam Jain
Prakhar Dubey
Somya Gupta
Dhruv Oza
Prajwal Sonawane
Prabhakar jaiswal

Acknowledgement

It gives immense pleasure in bringing out this synopsis of the project entitled “Localization of an autonomous bot”

Firstly we would like to thank our teachers and mentors who gave us their valuable suggestions and ideas when we were in need of them. They encouraged us to work on this project. This work would not have been possible without the support of DRISHTI, and our mentors.

We are grateful to all of those with whom we have had the pleasure to work during this and other related projects. Each of the members of the club has provided us extensive personal and professional guidance and taught us a great deal about both academics and life in general, as without their inspiration and valuable suggestion it would not have been possible to develop the project within the prescribed time.

Abstract

The most important thing while autonomously navigating through the local environment is to know exactly where your robot is. Localization is the process of determining where a mobile robot is located with respect to its environment. This is achieved by using sensors such as LIDAR, IMU and data received from GPS. In this project We applied the Kalman Filter on the data received by IMU, LIDAR and GPS and estimate the co- ordinates of a self-driving car and visualize its real trajectory versus the ground truth trajectory.

The Kalman Filter is a set of mathematical equations that when operating together implement a predictor-corrector type of estimator that is optimal in the sense that it minimizes the estimated error co-variance when some presumed conditions are met. We derived the Kalman Filter for linear systems and extend it to a non linear system such as a self- driving car.

Contents

1 Localization

1.1 Introduction

Localization involves one question: Where is the robot now? Or, robo-centrically, where am I, keeping in mind that "here" is relative to some landmark (usually the point of origin or the destination) and that you are never lost if you don't care where you are.

The robot localization problem is a key problem in making truly autonomous robots. If a robot does not know where it is, it can be difficult to determine what to do next. In order to localize itself, a robot has access to relative and absolute measurements giving the robot feedback about its driving actions and the situation of the environment around the robot. Given this information, the robot has to determine its location as accurately as possible. What makes this difficult is the existence of uncertainty in both the driving and the sensing of the robot. The uncertain information needs to be combined in an optimal way.

Although a simple question, answering it isn't easy, as the answer is different depending on the characteristics of your robot. Localization techniques that work fine for one robot in one environment may not work well or at all in another environment. For example, localizations which work well in an outdoors environment may be useless indoors.

All localization techniques generally provide two basic pieces of information:

- what is the current location of the robot in some environment?
- what is the robot's current orientation in that same environment?

The first could be in the form of Cartesian or Polar coordinates or geographic latitude and longitude. The latter could be a combination of roll, pitch and yaw or a compass heading.

1.2 The Challenge of Localization

If one could attach an accurate GPS (global positioning system) sensor to a mobile robot, much of the localization problem would be obviated. The GPS would inform

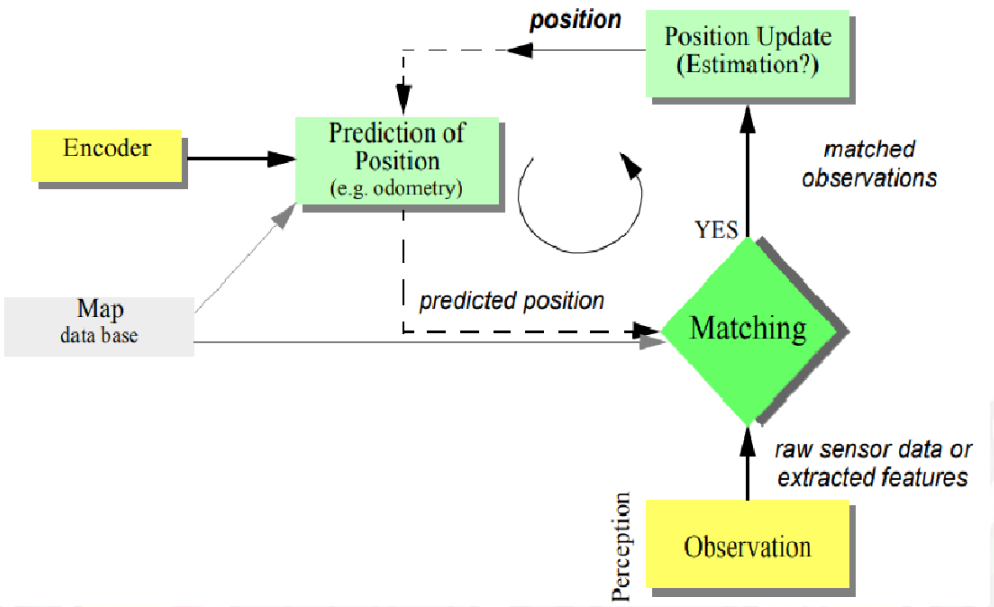


Figure 1: General Schematic for mobile robot localization

the robot of its exact position, indoors and outdoors, so that the answer to the question, "Where am I?", would always be immediately available. Unfortunately, such a sensor is not currently practical.

The existing GPS network provides accuracy to within several meters, which is unacceptable for localizing human-scale mobile robots as well as miniature mobile robots such as desk robots and the body-navigating nanorobots of the future. Furthermore, GPS technologies cannot function indoors or in obstructed areas and are thus limited in their workspace. But, looking beyond the limitations of GPS, localization implies more than knowing one's absolute position in the Earth's reference frame.

Furthermore, during the cognition step a robot will select a strategy for achieving its goals. If it intends to reach a particular location, then localization may not be enough. The robot may need to acquire or build an environmental model, a map, that aids it in planning a path to the goal. Once again, localization means more than simply determining an absolute pose in space; it means building a map, then identifying the robot's position relative to that map.

Clearly, the robot's sensors and effectors play an integral role in all the above forms of

localization. It is because of the inaccuracy and incompleteness of these sensors and effectors that localization poses difficult challenges. This section identifies important aspects of this sensor and effector suboptimality.

1.2.1 Sensor Noise

Sensors are the fundamental robot input for the process of perception. Sensor noise induces a limitation on the consistency of sensor readings in the same environmental state and, therefore, on the number of useful bits available from each sensor reading. Often, the source of sensor noise problems is that some environmental features are not captured by the robot's representation and are thus overlooked.

Sensor noise reduces the useful information content of sensor readings. Clearly, the solution is to take multiple readings into account, employing temporal fusion or multisensor fusion to increase the overall information content of the robot's inputs.

1.2.2 Sensor Aliasing

A second shortcoming of mobile robot sensors causes them to yield little information content, further exacerbating the problem of perception and, thus, localization.

In robots, the nonuniqueness of sensor readings, or sensor aliasing, is the norm and not the exception. Consider a narrow-beam rangefinder such as an ultrasonic or infrared rangefinder. This sensor provides range information in a single direction without any additional data regarding material composition such as color, texture, and hardness. Even for a robot with several such sensors in an array, there are a variety of environmental states that would trigger the same sensor values across the array.

Formally, there is a many-to-one mapping from environmental states to the robot's perceptual inputs. Thus, the robot's percepts cannot distinguish from among these many states. With sonar alone, these states are aliased and differentiation is impossible.

The problem posed to navigation because of sensor aliasing is that, even with noise-free sensors, the amount of information is generally insufficient to identify the robot's position from a single-percept reading. Thus techniques must be employed by the robot programmer that base the robot's localization on a series of readings and, thus, sufficient information to recover the robot's position over time.

1.2.3 Efector Noise

The challenges of localization do not lie with sensor technologies alone. Just as robot sensors are noisy, limiting the information content of the signal, so robot effectors are also noisy. In particular, a single action taken by a mobile robot may have several different possible results, even though from the robot's point of view the initial state before the action was taken is well known.

In short, mobile robot effectors introduce uncertainty about future state. Therefore the simple act of moving tends to increase the uncertainty of a mobile robot. There are, of course, exceptions. Using cognition, the motion can be carefully planned so as to minimize this effect, and indeed sometimes to actually result in more certainty. Furthermore, when the robot's actions are taken in concern with careful interpretation of sensory feedback, it can compensate for the uncertainty introduced by noisy actions using the information provided by the sensors.

To understand the precise nature of the effector noise that impacts mobile robots, it is important to note that, from the robot's point of view, this error in motion is viewed as an error in odometry, or the robot's inability to estimate its own position over time using knowledge of its kinematics and dynamics. The true source of error generally lies in an incomplete model of the environment. For instance, the robot does not model the fact that the floor may be sloped, the wheels may slip, and a human may push the robot. All of these unmodeled sources of error result in inaccuracy between the physical motion of the robot, the intended motion of the robot, and the proprioceptive sensor estimates of motion.

In odometry (wheel sensors only) and dead reckoning (also heading sensors) the position update is based on proprioceptive sensors. The movement of the robot, sensed with wheel encoders or heading sensors or both, is integrated to compute position. Because the sensor measurement errors are integrated, the position error accumulates over time. Thus the position has to be updated from time to time by other localization mechanisms. Otherwise the robot is not able to maintain a meaningful position estimate in the long run.

Using additional heading sensors (e.g., gyroscope) can help to reduce the cumulative errors, but the main problems remain the same. There are many sources of odometric error, from environmental factors to resolution:

- Limited resolution during integration (time increments, measurement resolu-

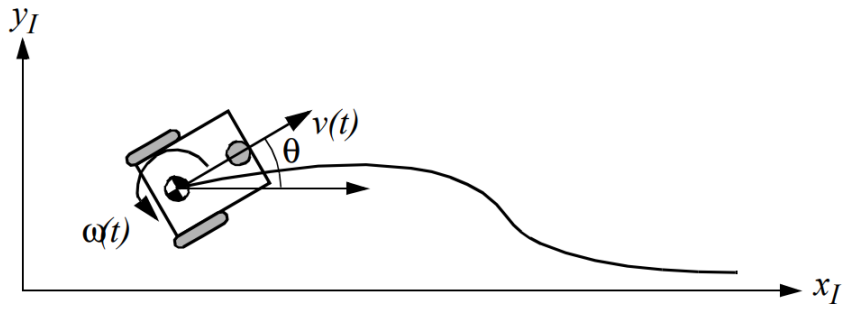


Figure 2: movement of a differential drive robot

tion, etc.);

- Misalignment of the wheels (deterministic);
- Uncertainty in the wheel diameter and in particular unequal wheel diameter (deterministic);
- Variation in the contact point of the wheel;
- Unequal floor contact (slipping, nonplanar surface, etc.).

Some of the errors might be deterministic (systematic), thus they can be eliminated by proper calibration of the system. However, there are still a number of nondeterministic (random) errors which remain, leading to uncertainties in position estimation over time. From a geometric point of view one can classify the errors into three types:

1. Range error: integrated path length (distance) of the robot's movement-¿sum of the wheel movements
2. Turn error: similar to range error, but for turns-¿difference of the wheel motions
3. Drift error: difference in the error of the wheels leads to an error in the robot's angular Orientation

Over long periods of time, turn and drift errors far outweigh range errors, since their contribution to the overall position error is nonlinear. Consider a robot whose position is initially perfectly well-known, moving forward in a straight line along the -axis. The error in the -position introduced by a move of meters will have a component of , which can be quite large as the angular error grows.

Over time, as a mobile robot moves about the environment, the rotational error between its internal reference frame and its original reference frame grows

1.3 Probabilistic Map-Based Localization

1.3.1 Introduction

Multiple-hypothesis position representation is advantageous because the robot can explicitly track its own beliefs regarding its possible positions in the environment. Ideally, the robot's belief state will change, over time, as is consistent with its motor outputs and perceptual inputs.

1.3.2 Types of Probabilistic Localization

There are two classes of probabilistic localization.

- The first class, Markov localization, uses an explicitly specified probability distribution across all possible robot positions.
- The second method, Kalman filter localization, uses a Gaussian probability density representation of robot position and scan matching for localization.

1.3.3 Markov Localization

In the case of Markov localization, the robot's belief state is usually represented as separate probability assignments for every possible robot pose in its map. The action update and perception update processes must update the probability of every cell in this case.
This fundamental difference in the representation of belief state leads to the following advantages and disadvantages of Markov Localization

Markov localization allows for localization starting from any unknown position and can thus recover from ambiguous situations because the robot can track multiple, completely disparate possible positions. However, to update the probability of all positions within the whole state space at any time requires a discrete representation of the space, such as a geometric grid or a topological graph. The required memory and computational power can thus limit precision and map size.

1.3.4 Kalman Filter Localization

Unlike Markov localization, Kalman filter localization does not independently consider each possible pose in the robot's configuration space. Interestingly, the Kalman filter localization process results from the Markov localization axioms if the robot's position uncertainty is assumed to have a Gaussian form.

Consider a mobile robot moving in a known environment. As it starts to move, say from a precisely known location, it can keep track of its motion using odometry. Due to odometry uncertainty, after some movement the robot will become very uncertain about its position . To keep position uncertainty from growing unbounded, the robot must localize itself in relation to its environment map.

To localize, the robot might use its on-board sensors (ultrasonic, range sensor, vision) to make observations of its environment. The information provided by the robot's odometry, plus the information provided by such exteroceptive observations, can be combined to enable the robot to localize as well as possible with respect to its map. The processes of updating based on proprioceptive sensor values and exteroceptive sensor values are often separated logically,
leading to a general two-step process for robot position update.

- Action update represents the application of some action model to the mobile robot's proprioceptive encoder measurements and prior belief state to yield a new belief state representing the robot's belief about its current position.
- Perception update represents the application of some perception model to the mobile robot's exteroceptive sensor inputs and updated belief state to yield a refined belief state representing the robot's current position.

In general, the action update process contributes uncertainty to the robot's belief about position: encoders have error and therefore motion is somewhat nondeterministic. By contrast, perception update generally refines the belief state. Sensor measurements, when compared to the robot's environmental model, tend to provide clues regarding the robot's possible position.

Kalman filter localization represents the robot's belief state using a single, well-defined Gaussian probability density function, and thus retains just a mu and sigma parameterization of the robot's belief about position with respect to the map. Updating the parameters of the Gaussian distribution is all that is required.

This fundamental difference in the representation of belief state leads to the following advantages and disadvantages of Kaman Filter Localization

Kalman filter localization tracks the robot from an initially known position and is inherently both precise and efficient. In particular, Kalman filter localization can be used in continuous world representations. However, if the uncertainty of the robot becomes too large (e.g., due to a robot collision with an object) and thus not truly unimodal, the Kalman filter can fail to capture the multitude of possible robot positions and can become irrevocably lost.

The Kalman Filter Localization Method

2 Kalman Filters

2.1 Introduction

Kalman Filtering can be understood as a way of making sense of a noisy world. When we want to determine where a robot is located, we can rely on two things: We know how the robot moves from time to time since we command it to move in a certain way. This is called state transitioning (i.e. how the robot moves from one state to the other). And we can measure the robot's environment using its various sensors such as cameras, lidar, or echolot. The problem is that both sets of information are subject of random noise. We do not know exactly how exactly the robot transitions from state to state since actuators are not perfect and we cannot measure the distance to objects with infinite precision. This is where Kalman Filtering comes to play.

Kalman filter is an efficient localization method because of key simplifications when representing the probability density function of the robot's belief state and even its individual sensor readings. But the benefit of this simplification is a resulting optimal recursive data-processing algorithm. It incorporates all information, regardless of precision, to estimate the current value of the robot's position.

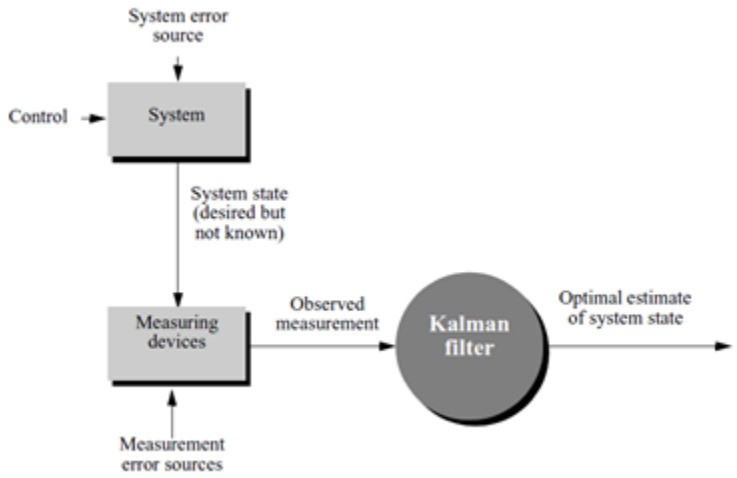


Figure 3: movement of a differential drive robot

Figure represents the general scheme of Kalman filter estimation, where a system has a control signal and system error sources as inputs. A measuring device enables measuring some system states with errors.

The Kalman filter is a mathematical mechanism for producing an optimal estimate of the system state based on the knowledge of the system and the measuring device, the description of the system noise and measurement errors and the uncertainty in the dynamics models.Thus the Kalman filter fuses sensor signals and systemknowledge in an optimal way. Within the Kalman filter theory the system is assumed to be linear and white with Gaussian noise.

Kalman Filtering allows us to combine the uncertainties regarding the current state of the robot (i.e. where it is located and in which direction it is looking) and the uncertainties regarding its sensor measurements and to ideally decrease the overall uncertainty of the robot. Both uncertainties are usually described by a Gaussian probability distribution, or Normal distribution. A Gaussian distribution has two parameters: mean and variance. The mean expresses, what value of the distribution has the highest probability to be true, and the variance expresses how uncertain we are regarding this mean value.

2.2 Application of Kalman filter

The Kalman filter is an optimal and efficient sensor fusion technique. Application of the Kalman filter to localization requires posing the robot localization problem as a sensor fusion problem. The basic probabilistic update of robot belief state can be segmented into two phases, perception update and action update.

The key difference between the Kalman filter approach and our earlier Markov localization approach lies in the perception update process. In Markov localization, the entire perception, that is, the robot's set of instantaneous sensor measurements, is used to update each possible robot position in the belief state individually using the Bayes formula. By contrast, perception update using a Kalman filter is a multistep process.
Given a set of possible features, the Kalman filter is used to fuse the distance estimate from each feature to a matching object in the map. Instead of carrying out this matching process for many possible robot locations individually as in the Markov approach, the Kalman filter accomplishes the same probabilistic update by treating the whole, unimodal, and Gaussian belief state at once

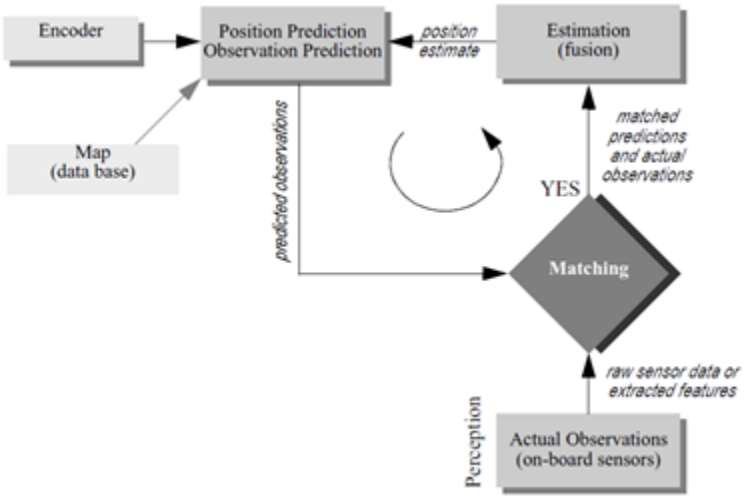


Figure 4: schematic for Kalman filter mobile robot localization

The first step is action update or position prediction, the straightforward application of a Gaussian error motion model to the robot's measured encoder travel.

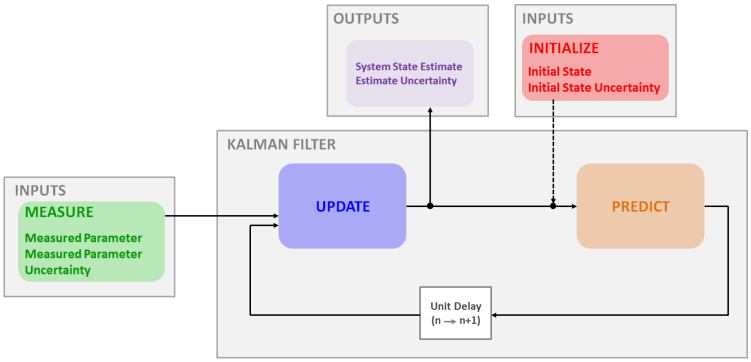


Figure 5: update and predict functions

The robot then collects actual sensor data and extracts appropriate features in the observation step. At the same time, based on its predicted position in the map, the robot generates a measurement prediction which identifies the features that the robot expects to find and the positions of those features.

In matching the robot identifies the best pairings between the features actually extracted during observation and the expected features due to measurement prediction. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required Finally, the Kalman filter can fuse the information provided by all of these matches to update the robot belief state in estimation.

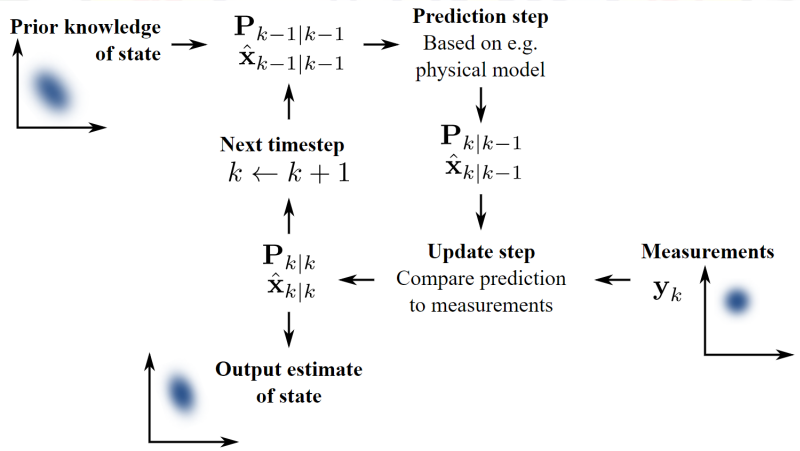


Figure 6: general steps for 1D kalman filter

The measurement is a random variable, described by the Probability Density Function (PDF). The measurements mean is the Expected Value of the random variable.

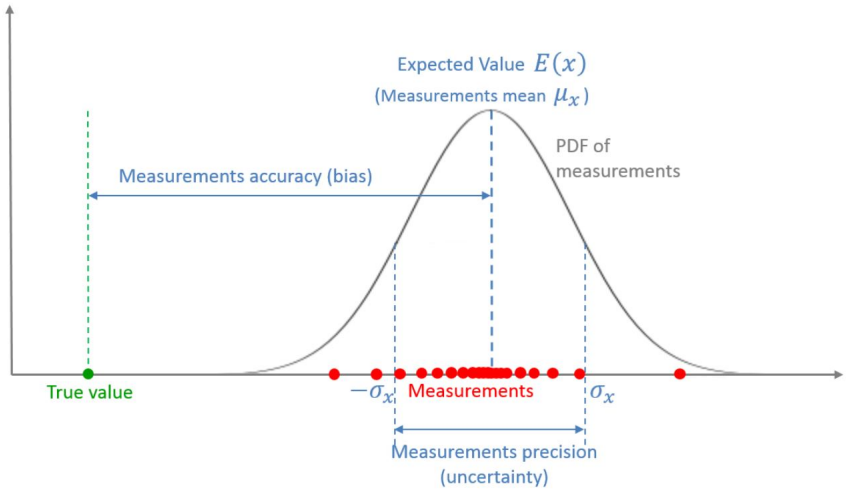
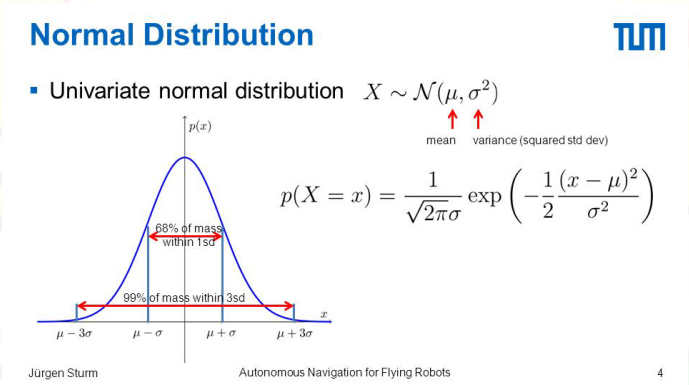
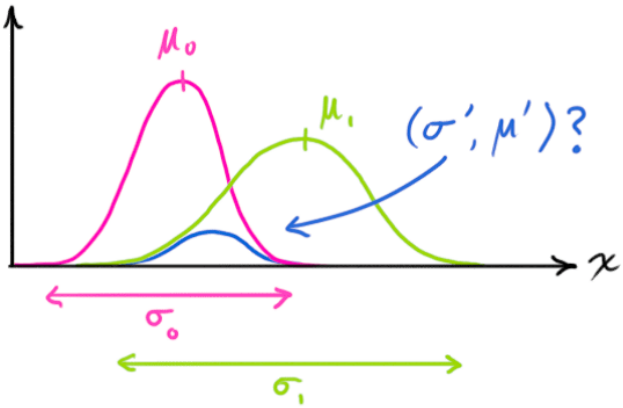


Figure 7: mean and variance



$$F(x) = e^{-(x-\mu)^2/2\sigma^2} \tag{1}$$

In order to predict and update our algorithm we need to calculate new mean and new variance.



2.3 Equations used in KF

2.3.1 Predict:

$$\hat{x}_k = F_k\hat{x} + \mu \tag{2}$$

$$P_k = F_k P_{k-1} + Q_k \tag{3}$$

2.3.2 Update:

$$S_k = H_k P_{k|k-1} + H_k^T + R_k \tag{4}$$

$$K_k = P_{k|k-1} + H_k^T + S_k^{-1} \tag{5}$$

$$\hat{y} = z_k - H_k\hat{x}_{k|k-1} \tag{6}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\hat{y}_k \tag{7}$$

$$P_{k|k} = (I - K_k H_k)P_{k|k-1} \tag{8}$$

3 Extended kalman Filters

3.1 Introduction

Extended Kalman Filtering is (as the name suggests) an extension of "Normal" Kalman Filtering.One additional assumption that was made implicitly when using Kalman Filters: The state transition model and the measurement model must be linear. From a mathematical standpoint this means that we can use the simplicity and elegance of Linear Algebra to update the robot's state and the robot's measurements. In practice, this means that the state variables and measured values are assumed to change linearly over time.

For instance, if we measure the robot's position in x-direction. We assume that if the robot was at position x_1 at time t_1, it must be at position x_1+v*(t_2–t_1) at time t_2. The variable v denotes the robot's velocity in x-direction. If the robot is actually accelerating, or doing any other kind of nonlinear motion (e.g driving around in a circle), the state transition model is slighly wrong. Under most circumstances, it is not wrong by much, but in certain edge cases, the assumption of linearity is simply too wrong.

Also assuming a linear measurement model comes with problems. Assume you are driving along a straight road and there is a lighthouse right next to the road in front of you. While you are quite some distance away, your measurement of your distance to the lighthouse and the angle in which it lies from your perspective changes pretty much linearly (the distance decreases by roughly the speed your car has and the angle stays more or less the same). But the closer you get and especially while you drive past it, the angle, on one hand, changes dramatically, and the distance, on the other hand, does not change very much.

This is why we cannot use Linear Kalman Filtering for Robby when he is navigating his 2-D world with landmarks scattered across his 2-D plane.

Extended Kalman Filter to the rescue! It removes the restriction of linear state transition and measurement models. Instead it allows you to use any kind of nonlinear function to model the state transition and the measurements you are making with your robot. In order to still be able to use the efficient and simple Linear Algebra magic in our filter, we do a trick: We linearize the models around the current robot state. This means that we assume the measurement model and the state transition

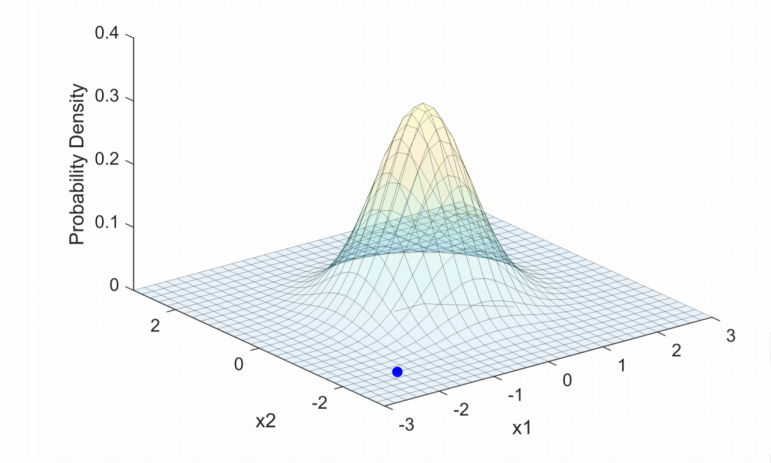


Figure 8: Gaussian in higher dimention

model to be approximately linear around the state at which we are right now .

But after every time step, we update this linearization around the new state estimate. While this approach forces us to make a linearization of this nonlinear function after every time step, it turns out to be not computationally expensive.
So there you have it. Extended Kalman Filtering is basically "Normal" Kalman Filtering just with additional linearization of the now nonlinear state transition model and measurement model.

3.2 Derivation of EKF

There were 2 main assumptions in kalman filter.The two assumptions are-:
1. Kalman Filter will always work with Gaussian Distribution.
2. Kalman Filter will always work with Linear Functions.
The prediction and update step both will contain Linear Functions only.

Then what is the problem now with KF? Most real world problems involve non linear functions. In most cases, the system is looking into some direction and taking measurement in another direction. This involves angles and sine, cosine functions which are non linear functions which then lead to problems. How does non linear function create problems?if input in a Gaussian is a linear function then the output is also a Gaussian.

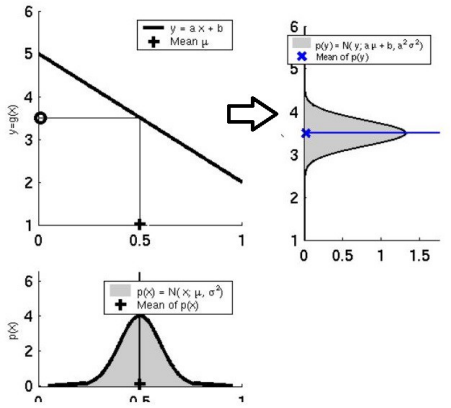


Figure 9: Gaussian + Linear Function = Gaussian

If you feed a Gaussian with a Non linear function then the output is not a Gaussian. Non Linear functions lead to Non Gaussian Distributions.

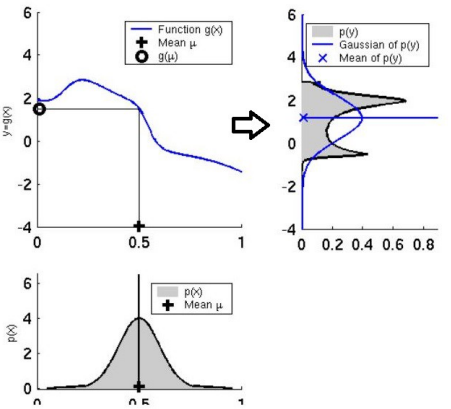


Figure 10: Gaussian + Non Linear Function = Non Gaussian

So if we apply a non linear function it will not end up as a Gaussian Distribution on which we can't apply Kalman Filter anymore. Non linearity destroys the Gaussian and it does not makes sense to compute the mean and variances.

but the functions are non linear so, we will make them Linear by approximation. Here, we will take help of a powerful tool called Taylor Series, which will help us to get a Linear Approximation of the Non Linear function. After applying the approximation what we get is an Extended Kalman Filter.

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots.$$

Figure 11: Taylor Series

we are interested in linearizing, so we are just interested in the first derivative of Taylor series. For every non linear function, we just draw a tangent around the mean and try to approximate the function linearly.

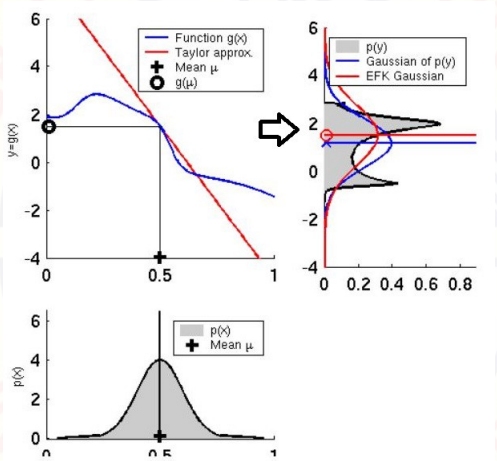


Figure 12: Scenario after applying Taylor's Approximation to Linearize our function

3.3 Equations for EKF

In the extended Kalman filter, the state transition and observation models don't need to be linear functions of the state but may instead be differentiable functions.

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{9}$$

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{10}$$

Here w(k) and v(k) are the process and observation noises which are both assumed to be zero mean multivariate Gaussian noises with covariance Q(k) and R(k) respectively. u(k) is the control vector.

The function "f" can be used to compute the predicted state from the previous estimate and similarly the function "h" can be used to compute the predicted measurement from the predicted state. However, "f" and "h" cannot be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian) is computed.

At each time step, the Jacobian is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the non-linear function around the current estimate.

3.3.1 Predict

Predicted state estimate

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k) \tag{11}$$

Predicted covariance estimate

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^\top + \boldsymbol{Q}_k \tag{12}$$

3.3.2 Update

Innovation or measurement residual

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) \tag{13}$$

Innovation (or residual) covariance

$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top + \boldsymbol{R}_k \tag{14}$$

Near-optimal Kalman gain

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top \boldsymbol{S}_k^{-1} \tag{15}$$

Updated state estimate

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k \tag{16}$$

Updated covariance estimate

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)\boldsymbol{P}_{k|k-1} \tag{17}$$

3.3.3 Jacobians

Where the state transition and observation matrices are defined to be the following Jacobians

$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k} \tag{18}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{19}$$

4 Sensors

4.1 Introduction

The term "input device" in the definition of a Sensor means that it is part of a bigger system which provides input to a main control system (like a Processor or a Microcontroller).
Another unique definition of a Sensor is as follows: It is a device that converts signals from one energy domain to electrical domain.

We live in a World of Sensors. You can find different types of Sensors in our homes, offices, cars etc. working to make our lives easier by turning on the lights by detecting our presence, adjusting the room temperature, detect smoke or fire, make us delicious coffee, open garage doors as soon as our car is near the door and many other tasks.
All these and many other automation tasks are possible because of Sensors.

4.2 Types of Sensors

In tis project we are using 3 sensors which are as follows

1. Internal Measurement Unit (IMU)
2. Global Positioning System (GPS)
3. Light Detection And Ranging (LiDAR)

4.2.1 IMU

The term IMU stands for "Inertial Measurement Unit," and we use it to describe a collection of measurement tools. When installed in a device, these tools can capture data about the device's movement. IMUs contain sensors such as accelerometers, gyroscopes, and magnetometers.

How Does an IMU Work?

IMUs can measure a variety of factors, including speed, direction, acceleration, specific force, angular rate, and (in the presence of a magnetometer), magnetic fields surrounding the device.
Each tool in an IMU is used to capture different data types:

- Accelerometer: measures velocity and acceleration
- Gyroscope: measures rotation and rotational rate
- Magnetometer: establishes cardinal direction (directional heading)

IMUs combine input from several different sensor types in order to accurately output movement.

What is an IMU Used For?

You commonly see IMUs used in navigational devices or as components of navigational equipment, such as:

- Manned and unmanned aircraft. A connected (or onboard) computer can use an IMU's measurements to calculate altitude and relative position to a reference frame, making them exceedingly useful in aircraft applications.
- GPS positioning systems IMUs serve as a supplement to GPS positioning systems, allowing the navigational device to continue with an estimated position and heading if it loses satellite connection.
- Most smartphones, tablets and fitness tracking devices contain a low-cost IMU
- IMUs are involved in sports training applications that need to measure, for example, the precise angle and force of a swing in golf or baseball.
- IMUs drive the self-balancing systems of personal transportation devices like Segways and hoverboards.

Disadvantages of IMUs

The principal disadvantage of an IMU is that they are prone to error that accumulates over time, also known as "drift." Because the device is always measuring changes relative to itself (not triangulating against an absolute or known outside device), the IMU constantly rounds off small fractions in its calculations, which accumulate over time. Left uncorrected, these tiny imprecisions can add up to significant errors.

Still, when coupled with a corrective technology or a human operator, IMUs can be a beneficial supplement to other sensors. In precision applications, you can suspend the sensors from shock absorbers to mitigate errors as well as protect the unit.

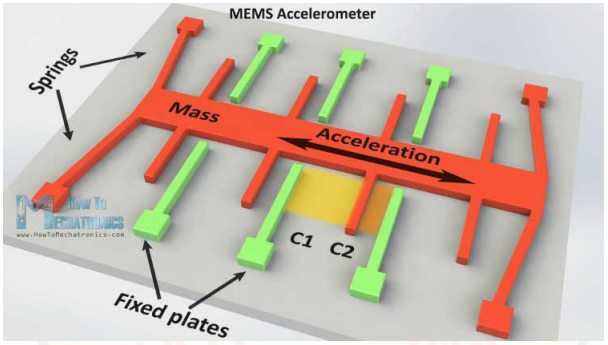


Figure 13: IMU

IMU Sensors Explained

IMUs are sensing devices that incorporate at least two (and often three) types of sensors to measure a host device's location in three-dimensional space. They are a valuable supplement to GPS or other navigational technologies. Additionally, they are present in a wide variety of consumer electronics that contain motion sensors (such as smartphones).

In most consumer applications, drift of negligible importance since an IMU doesn't require a high degree of long-term precision. In more precise applications, users should ensure the accuracy of the sensor readings and enact occasional controls to correct for drift, especially in positioning.

4.2.2 GPS

Global Positioning System (GPS) is a satellite-based system that uses satellites and ground stations to measure and compute its position on Earth. GPS is also known as Navigation System with Time and Ranging (NAVSTAR) GPS.

GPS receiver needs to receive data from at least 4 satellites for accuracy purpose. GPS receiver does not transmit any information to the satellites.GPS receivers are generally used in smartphones, fleet management system, military etc. for tracking or finding location.

How GPS Works?

GPS receiver uses a constellation of satellites and ground stations to calculate accurate location wherever it is located. These GPS satellites transmit information signal over radio frequency (1.1 to 1.5 GHz) to the receiver. With the help of this received information, a ground station or GPS module can compute its position. This is done by measuring the time required for the signal to travel from satellite to the receiver.

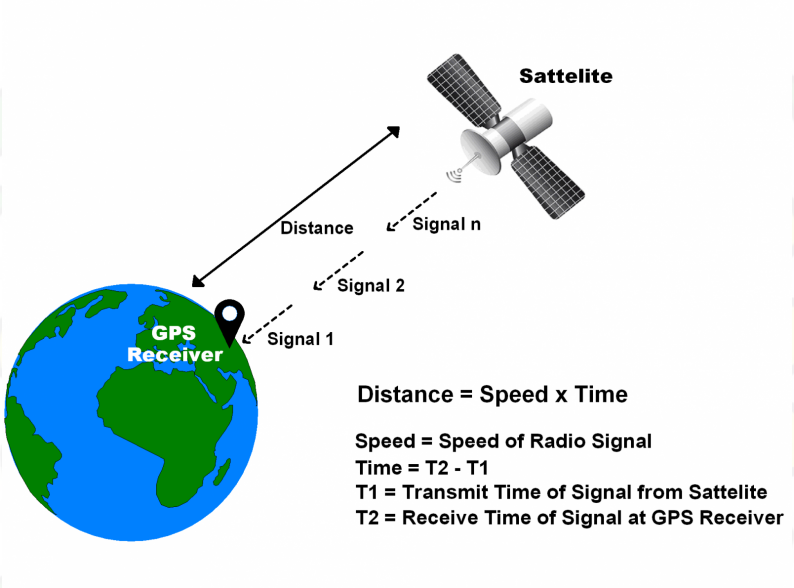


Figure 14: GPS distance calculation

Distance = Speed x Time
Where,
Speed = Speed of Radio signal which is approximately equal to the speed of light i.e.

$$c = 3 * 10^8$$

Time = Time required for a signal to travel from the satellite to the receiver.
By subtracting the sent time from the received time, we can determine the travel time.

To determine distance, both the satellite and GPS receiver generate the same pseudocode signal at the same time.
The satellite transmits the pseudocode; which is received by the GPS receiver.

These two signals are compared and the difference between the signals is the travel time.

Now, if the receiver knows the distance from 3 or more satellites and their location (which is sent by the satellites), then it can calculate its location by using Trilateration method.

4.2.3 LiDAR

LIDAR - or Light Detection And Ranging - originated in the 1960s, shortly after the advent of lasers and was first used by the American National Center for Atmospheric Research in meteorology to measure clouds. Since then, the technique - also known as laser scanning or 3D scanning - has been used in applications from geography to forestry, and from atmospheric physics to laser altimetry.

It has been widely used in archaeology to map dig-sites and large areas of land, identifying things that couldn't be seen from the ground. The National Oceanic and Atmospheric Administration in America has used it to map shorelines and the surface of the Earth, and NASA utilized the technology in 1971 when Apollo 15 astronauts mapped the surface of the moon using a laser altimeter.

How Do They Work?

The technique employs ultraviolet (UV), visible or near infrared (IR) light to image objects and map their physical features. Several measurements are taken in quick succession to yield a complex map of the surface at high resolution.

LIDAR measures the distance to a target using active sensors which emit an energy source for illumination, instead of relying on sunlight. It fires rapid pulses of laser light at a surface – anything up to 150,000 pulses a second – usually IR to map land, or water-penetrating green light to measure the seafloor or riverbed.

When the light hits the target object, it is reflected back to a sensor which measures the time taken for the pulse to bounce back from the target. The distance to the object is deduced by using the speed of light to calculate the distance traveled accurately. The result is precise three-dimensional information about the target object and its surface characteristics.

Applications

LIDAR was first used in vehicles in the early 2000s, mostly in the Grand DARPA Challenge – it is only in the last five years, or so that progress has been made concerning self-driving cars. Google and Uber are just a two of many names developing self-driving vehicles; their cars feature a bulky box on top of the roof which spins continuously giving 360 visibility and precise, in-depth information about the exact distance to an object to an accuracy of 2cm. This box is the LIDAR system; it consists of a laser, scanner and optics and a specialized GPS receiver, especially important if the system is moving.

In the case of self-driving cars, LIDAR is used to generate huge 3D maps – which was its intended original use - that the car can then navigate through. It is also used – particularly by Google – to detect pedestrians and cyclists, traffic signs and other nearby obstacles.

Of course, such autonomous technology isn't without its pitfalls – take the recent fatality in Arizona where the technology failed to pick up a pedestrian crossing the road, for example. However, as LIDAR becomes more sophisticated, it will be increasingly capable of detecting and tracking objects. Improvements will mean higher resolution imagery will be possible and it will be able to operate at longer ranges so that the technology is capable of differentiating between someone walking, or someone on a bike, their speed, and direction.

SLAM using LiDAR

Light detection and ranging (LiDAR) is a method that primarily uses a laser sensor (or distance sensor). Compared to cameras, ToF, and other sensors, lasers are significantly more precise, and are used for applications with high-speed moving vehicles such as self-driving cars and drones.

The output values from laser sensors are generally 2D (x, y) or 3D (x, y, z) point cloud data. The laser sensor point cloud provides high-precision distance measurements, and works very effectively for map construction with SLAM. Generally, movement is estimated sequentially by matching the point clouds. The calculated movement (traveled distance) is used for localizing the vehicle. For lidar point cloud matching, the following algorithms are used

- iterative closest point (ICP)

- normal distributions transform (NDT)

2D or 3D point cloud maps can be represented as a grid map or voxel map.

On the other hand, point clouds are not as finely detailed as images in terms of density and do not always provide sufficient features for matching. For example, in places where there are few obstacles, it is difficult to align the point clouds and this may result in losing track of the vehicle location. In addition, point cloud matching generally requires high processing power, so it is necessary to optimize the processes to improve speed. Due to these challenges, localization for autonomous vehicles may involve fusing other measurement results such as wheel odometry, global navigation satellite system (GNSS), and IMU data. For applications such as warehouse robots, 2D lidar SLAM is commonly used, whereas SLAM using 3-D lidar point clouds can be used for UAVs and automated parking.

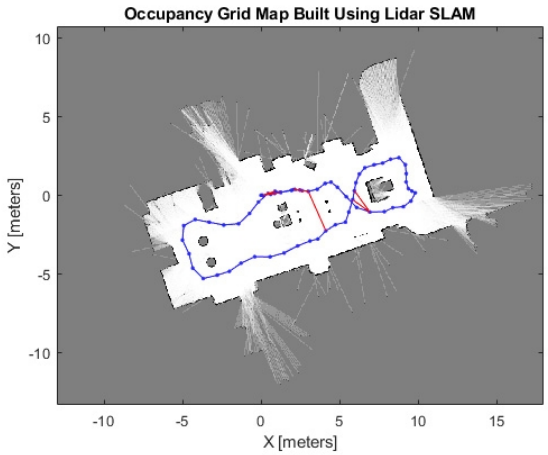


Figure 15: SLAM with 2D LiDAR

5 Code And Implementation

```
#Importing All Libraries

import pickle  #For Data Handling
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from rotations import angle_normalize, rpy_jacobian_axis_angle,
skew_symmetric, Quaternion #For Quaternions and Euler angles



#Load the Data

with open('data/pt1_data.pkl', 'rb') as file:
    data = pickle.load(file)



# gt: Data object containing ground truth, with the following fields:
# imu_a.data : IMU accelerations in Vehicle Frame
# imu_w.data : IMU angular velocities in Vehicle Frame
(Used for quaternion calculations)
# gnss.data : GNSS data
# lidar.data : LIDAR data
# imu_a.t, imu_w.t, gnss.t, lidar.t : Respective timestamps

#These are the StampedData objects
gt = data['gt']
imu_a = data['imu_f']
imu_w = data['imu_w']
gnss = data['gnss']
lidar = data['lidar']

#These are the total number of observations obtained from each
sensor in the total timespan of trajectory
IMU_time = 8734
```

```
LIDAR_time = 415
GNSS_time = 45

#we are using the following actual data and timestamps, obtained
from the StampedData object
imu_a.data = imu_a.data[0:IMU_time, :]
imu_a.t = imu_a.t[0:IMU_time]
imu_w.data = imu_w.data[0:IMU_time, :]
imu_w.t = imu_w.t[0:IMU_time]
gnss.data = gnss.data[0:GNSS_time, :]
gnss.t = gnss.t[0:GNSS_time]
lidar.data = lidar.data[0:LIDAR_time, :]
lidar.t = lidar.t[0:LIDAR_time]



#this cell plots the ground truth tajectory

gt_fig = plt.figure()
ax = gt_fig.add_subplot(111, projection='3d')
ax.plot(gt.p[:,0], gt.p[:,1], gt.p[:,2])
ax.set_xlabel('x [m]')
ax.set_ylabel('y [m]')
ax.set_zlabel('z [m]')
ax.set_title('Ground Truth trajectory')
ax.set_zlim(-1, 5)
plt.show()



# Transform the LIDAR data from the LIDAR frame to the vehicle
(IMU) frame.

#Rotation Matrix
C_li = np.array([
      [ 0.9975 , -0.04742,  0.05235],
      [ 0.04992,  0.99763, -0.04742],
```

```
    [-0.04998,  0.04992,  0.9975 ] ])

#Translation Matrix
t_i_li = np.array([0.5, 0.1, 0.5])

lidar.data = (C_li @ lidar.data.T).T + t_i_li



# Defining variances for different sensors
# This will be used to construct the Covariance matrices

var_imu_a = 0.15
var_imu_w = 0.15
var_gnss  = 0.1
var_lidar = 2



#The Jacobians 'L' and 'H' and gravity 'g'

g = np.array([0, 0, -9.81])  # gravity

L = np.zeros([9, 6])
L[3:, :] = np.eye(6)  # motion model noise jacobian

H = np.zeros([3, 9])
H[:, :3] = np.eye(3)  # measurement model jacobian



#There are total 10 states for a Self Driving Car
# 3 for Positions in each direction (px, py and pz)
# 3 for Velocities in each direction (vx, vy and vz)
# 4 for Quaternions

#Creating an array of zeros for each state
# position estimates
```

```
p_est = np.zeros([imu_a.data.shape[0], 3])

# velocity estimates
v_est = np.zeros([imu_a.data.shape[0], 3])

# orientation estimates as quaternions
q_est = np.zeros([imu_a.data.shape[0], 4])

# covariance matrices at each timestep
p_cov = np.zeros([imu_a.data.shape[0], 9, 9])

# Set initial values.
p_est[0] = gt.p[0]
v_est[0] = gt.v[0]
q_est[0] = Quaternion(euler=gt.r[0]).to_numpy()
p_cov[0] = np.zeros(9)  # covariance of estimate



# Create a function to evaluate Measurement update for LIDAR
and/or GNSS data

def measurement_update(sensor_var, p_cov_check, z_k, p_check,
v_check, q_check):
    # 1) Transform Sensor Variance into Matrix form
    (Covariance matrix)
    R_cov=np.eye(3)*sensor_var

    # 2) Compute Kalman Gain
    S=H.dot((p_cov_check).dot(H.T)) + R_cov
    K=p_cov_check.dot((H.T).dot(np.linalg.inv(S)))

    # 3) Compute innovation and multiply kalman gain
    I = z_k-p_check
    X=K.dot(I)

    # 4) Correct predicted state
    p_hat = p_check + X[0:3]
```

```
    v_hat = v_check + X[3:6]

    # Corrected state for Quaternions
    q_hat = Quaternion(euler=X[6:]).quat_mult_left(q_check)

    # 5) Compute corrected covariance
    p_cov_hat=(np.eye(9)-K.dot(H)).dot(p_cov_check)

    return p_hat, v_hat,q_hat, p_cov_hat



# Motion Model loop to predict using IMU values

for k in range(1, imu_a.data.shape[0]):  # start at 1 b/c we have
initial prediction from gt

    # 1) Calculate "delta_t"
    delta_t=imu_a.t[k]-imu_a.t[k-1]

    # 2) Update state with IMU inputs ("p_k, v_k")
    #C_ns is Euler estimation for Quaternions
    # In position update, use matrix multiplication of (C_ns and
    imu_a.data[number]) to linearize the model
    # Instead of just using imu_a.data[number] (Use this in
    velocity)

    C_ns = Quaternion(*q_est[k-1]).to_mat()

    p_k= p_est[k-1]+v_est[k-1]*delta_t+0.5*(C_ns.dot((imu_a.data[k-
    1])+g))*(delta_t**2)

    v_k= v_est[k-1]+(C_ns.dot((imu_a.data[k-1])+g))*delta_t

    # Quaternion States update
    q_k = Quaternion(axis_angle=imu_w.data[k-1]*delta_t).quat_mult_
    right(q_est[k-1])
```

```
# Linearize the motion model and compute Jacobians
Fk = np.eye(9)
Fk[0:3, 3:6] = np.eye(3)*delta_t
Fk[3:6, 3:6] = np.eye(3)
Fk[3:6, 6:9] =
-(C_ns@skew_symmetric(imu_a.data[k-1].reshape((3,1))))*delta_t

# 3) Propagate uncertainty
Q = np.eye(6)
Q[0:3, 0:3] = var_imu_a * Q[0:3, 0:3]
Q[3:6, 3:6] = var_imu_w * Q[3:6, 3:6]
Q = np.dot(Q,Q.T)*(delta_t ** 2) #Integration acceleration to
obstain Position0

# 4) Calculate Covariance Matrix (A diagonal matrix consisting
of multiple variances)
p_cov_check=Fk.dot((p_cov[k - 1]).dot(Fk.T)) +
L.dot((Q).dot(L.T))

# 5) Check if GNSS and LIDAR measurements are avaiable
for i in range(len(gnss.t)):
    if abs(gnss.t[i] - imu_a.t[k]) <=0.01:
        p_k, v_k,q_k, p_cov_check =
        measurement_update(var_gnss, p_cov_check, gnss.data[i],
        p_k, v_k, q_k)
for i in range(len(lidar.t)):
    if abs(lidar.t[i] - imu_a.t[k]) ==0:
        p_k, v_k,q_k, p_cov_check =
        measurement_update(var_lidar, p_cov_check,
        lidar.data[i], p_k, v_k, q_k)

#updating
p_est[k,:] = p_k
v_est[k,:] = v_k
q_est[k,:] = q_k
p_cov[k,:,:] = p_cov_check
```

```
# Visualize your trajectory w.r.t. the ground truth trajectory
and calculate the mean squared error

est_traj_fig = plt.figure()
ax = est_traj_fig.add_subplot(111, projection='3d')
ax.plot(gt.p[:,0], gt.p[:,1], gt.p[:,2], label='Ground Truth')
ax.plot(p_est[:,0], p_est[:,1], p_est[:,2], label='Estimated')
ax.set_xlabel('Easting [m]')
ax.set_ylabel('Northing [m]')
ax.set_zlabel('Up [m]')
ax.set_title('Ground Truth and Estimated Trajectory')
ax.set_xlim(0, 200)
ax.set_ylim(0, 200)
ax.set_zlim(-100, 100)
ax.set_xticks([0, 50, 100, 150, 200])
ax.set_yticks([0, 50, 100, 150, 200])
ax.set_zticks([-2, -1, 0, 1, 2])
ax.legend(loc=(0.62,0.77))
ax.view_init(elev=40, azim=-50)
plt.show()

MSE = np.sum(np.square(np.subtract(gt.p, p_est)),
axis=None)/IMU_time
print("The Mean Squared Error is :{0} ".format(MSE))
```

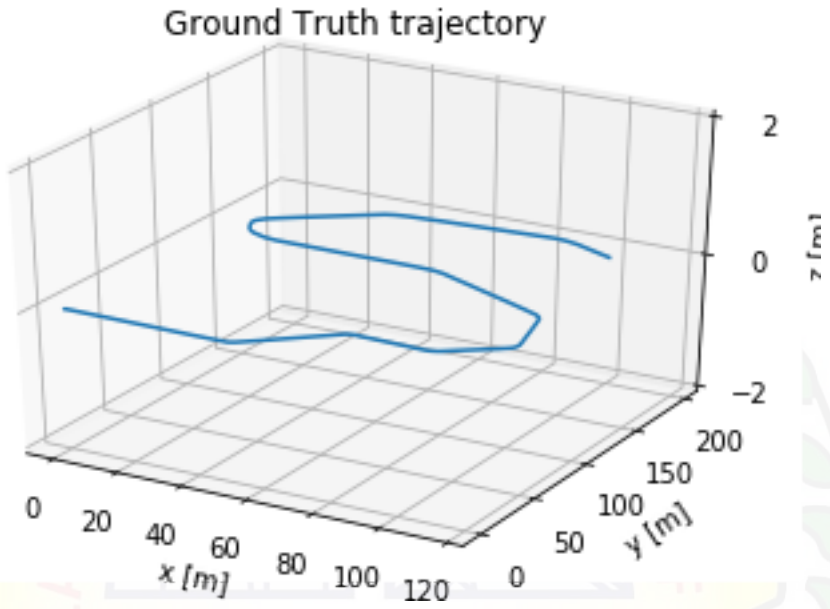


Figure 16: Ground Truth Trajectory

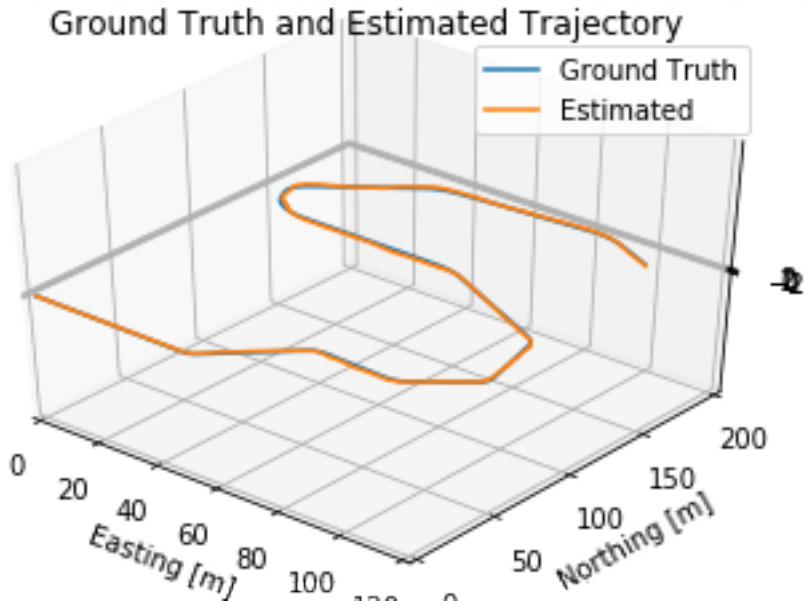


Figure 17: The Mean Squared Error is :3.1523085368545334

6 Timeline

18 August 2020-
We started this project.We Started off with Basic Localisation theory from a course on Artificial Intelligence for Robotics is given in the link below which includes courses for locaization as well as kalman filter problems.
https : *//classroom.udacity.com/courses/cs*373

25 August 2020-
With all the things learnt,we started implementing Kalman filters in 1D using python. The input data was taken from Accelerometer and GPS and successfully used in predict and update function of the kalman filter.

1 september 2020-
we started documenting all that we learnt along all these days and listing all the problems we faced.Initially we individually made seperate documentations and then we combined them in the final stage.

5 september 2020-
DRISHTI conducted an update meet for all the teams to give everyone updates about their project.

6 September 2020-
we started learning about extended kalman filters and error state kalman filters and we also started research for the same.

7 september 2020-
We started implementing extended kalman filters on a dataset from an autonomus bot.

12 september 2020-
We implemented the motion model for the extended kalman filter.

17 september 2020-
We implemented the measurement model for the Extended Kalman Filter, completed our final code of extended kalman filter and calculated the MSE of our Extended kalman filter estimation.