Using Extended Kalman Filters for SLAM

Amy Phung & Kawin Nikomborirak

December 2018

1 Introduction

Simultaneous localization and mapping, or SLAM, is the process of estimating the pose of a robot and mapping the environment at the same time. This is a difficult problem to solve, since a map is needed for localization, but a good pose estimate is needed for mapping. The uses of SLAM are abundant, especially in robotics applications where robots are operating in unknown conditions. Some examples include robot vacuum cleaners that need to be able to navigate around obstacles without knowing floor plans or the location of furniture beforehand, exploratory submarine robots that are used to discover unknown regions of the ocean, or in space rovers for terrain mapping. [1]

This problem's difficulty is compounded by the fact that it has to be solved with oftentimes incomplete and erroneous data, considering that good sensors capable of accurately sensing everything are often expensive. In addition to this most robots have computational power limits, which prevents robots from using every piece of data available to it to solve this problem.

In this technical report, we will explain a solution to SLAM which uses a probability-based algorithm - the Extended Kalman Filter (EKF) - to localize a robot with using only lidar data. This paper assumes prerequisite knowledge which is briefly covered in the Mathematical Background section. While these concepts are explained briefly, this paper will not go in detail into derivations for each of the mathematical concepts. Specifics on the theory behind Kalman filters is covered in the Quantitative Calculation section, which includes other important concepts including Bayesian statistics and Gaussians. Details of the implementation are discussed in the Implementation section, and a discussion of the results and conclusions are included at the end.

2 Mathematical Background & Kalman Filter Theory

Note: Mathematical explanation, equations, and figures are highly derivative of the notebook available at `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python`.

[1] Slideshow Introduction to SLAM `http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/12-slam.pdf`

The bulk of our quantitative analysis is in our Extended Kalman Filter implementation, or EKF for short. For starters, a Kalman filter is an algorithm that uses a series of measurements observed over time that incorporates noise and predictions and produces estimates of unknown variables that tend to be more accurate than any single measurement alone. [2]

We focus on the Extended Kalman Filter (EKF), which is the nonlinear version of the Kalman filter that linearizes about an estimate of the current mean and covariance. [3] In the following subsections, we go into further detail on the intuition and concepts embedded within the EKF.

2.1 Building Intuition: The g-h Filter

Before we dive into the EKF, we first take a look at the g-h filter. The EKF is a form of a g-h filter, but is more complex in the ways it derives values, which makes it more robust and capable of handling more difficult problems. The main purpose of introducing the g-h filter is to provide a strong intuitive understanding of how the EKF works without getting too deep in the complex math required to deal with the nonlinear and multidimensional nature of problems solved with the EKF. In this section, we will walk through the operating principles of these filters, which will provide us with a intuitive road map we can later rely on when the math becomes more complex. [3]

Starting with a very simple 1-dimensional case, suppose you have a NEATO [4] sitting somewhere along a line and you want to know at what position the NEATO is at along the line. To find this out, you have a sensor that tells you what the NEATO's position is, but it's pretty inaccurate. If you have two measurements, (A) and (B), and the first measurement (A) reads 90 meters and (B) reads 110 meters, you'd intuitively expect the correct location to be between those two numbers. If you trusted the two measurements equally and had to guess the true distance, you'd probably guess the it to be around 100 meters. [3]

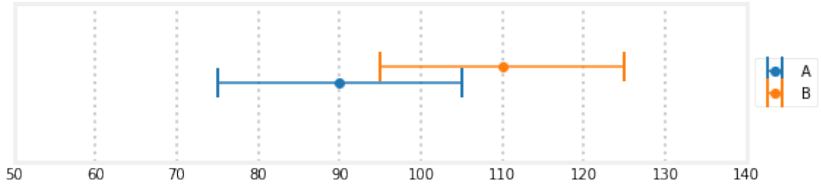

Figure 2.1: Error bar chart showing 90m and 110m with an assumed error of ± 15 m [3]

What if we measured the location of the NEATO multiple times? Assuming the NEATO doesn't run away during that time, you'd expect the true location to be fairly close to the average of all your measurements.

And if the NEATO does run away, you can also infer that from the measurements even with the noise.

[2] General Kalman Filter Introduction `https://en.wikipedia.org/wiki/Kalman_filter`
[3] Extended Kalman Filter Introduction `https://en.wikipedia.org/wiki/Extended_Kalman_filter`
[4] a robot vacuum platform with a differential drive and a 360 degree lidar

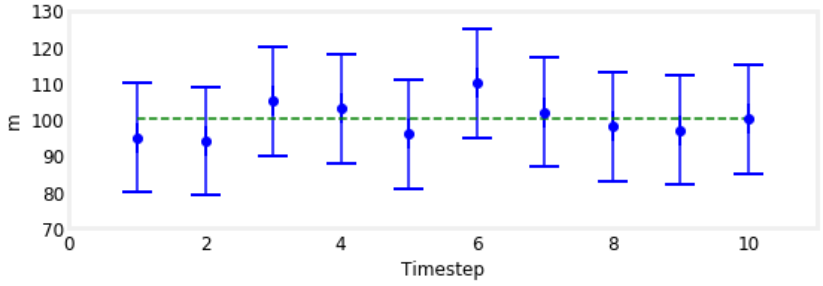

Figure 2.2: Error bar chart of multiple measurements with an assumed error of ± 15 m and an average of 100 m [3]

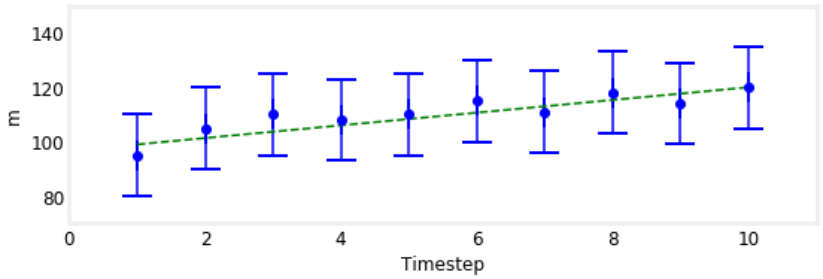

Figure 2.3: Error bar chart of multiple measurements with an assumed error of ± 15 m and an average increase of 2m/s [3]

From these measurements, we could make a prediction of where we expect the NEATO to be next. But our prediction might be wrong. But this should not come too much as a surprise. If our prediction was always exactly the same as the measurement, it would not add any information to the filter - there would be no reason to ever measure anything since the prediction is always perfect. [3]

So what do we do? If we only use the measurement then the prediction will not affect the result. If we only use the prediction then the measurement will be ignored. The optimal solution lies within a **blend of the prediction and the measurement**, which is what the Kalman filter attempts to do. [3]

In practice, what does "blending the prediction and the measurement" mean? Should the estimate be half way between the measurement and the prediction? Maybe, but oftentimes we may trust our prediction more if we have a really good model of the system, and other times we may trust our measurement more if we have a really good sensor. In these cases, the best estimate lies somewhere else. [3]

As an example, let's say that we trust our prediction a bit more than our estimate, but not a whole lot more. We can choose a number to represent this, such as $\frac{4}{10}$. In this case, our estimate will be four tenths the measurement and the rest will be from the prediction. In other words, we are expressing a belief here, a belief that the prediction is somewhat more

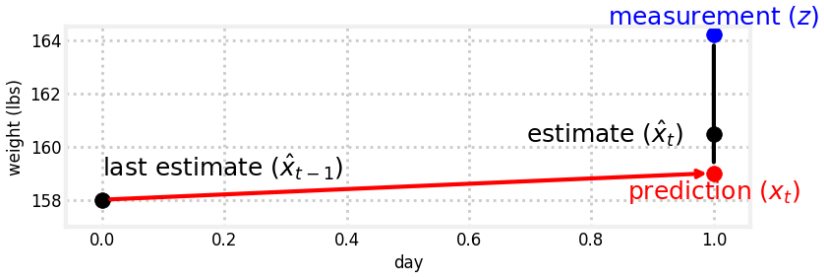

Figure 2.4: Previous estimate informs prediction, new estimate is partway between prediction and measurement [3]

likely to be correct than the measurement. We can compute that as

$$\text{estimate} = \text{prediction} + \frac{4}{10}(\text{measurement} - \text{prediction}) \tag{1}$$

The difference between the measurement and the prediction is called the residual, which is depicted by the black vertical line in the plot above. [3]

To make this example more concrete and easier to grasp, let's apply this thinking to a physical example with more numbers. Assume we're trying to estimate the location of a NEATO that starts at 60 m and seems to be moving forward at a velocity of around 1 m/s, and our trust in the measurement versus the prediction is around $\frac{4}{10}$. [3]

In the equation above, we know what our measurement is, but what is our prediction? Since the previous location was 60 m and we expect the NEATO to be moving forward at 1 m/s, our prediction would be 61 m. Assuming we got a measurement of 62 m, plugging these numbers into the above equation yields: [3]

$$(60 + 1) + \frac{4}{10}(62 - 61) = 61.4 \tag{2}$$

What if our next measurement was 67 m? This would imply a change of 5.6 m $(67 - 61.4)$, but it seems rather implausible that our NEATO could change it's velocity from 1 m/s to 5.6 m/s instantaneously. We don't want to discard our measurement, but at the same time the accuracy of the measurement seems unlikely. Once again, we have two numbers, and we want to combine them somehow (this sounds like our previous problem!). Let's use the idea of picking a value part way between the two again - since our measurements are a lot noisier than we expect, let's say that our trust in the implied velocity versus the expected velocity is around $\frac{1}{3}$. The equation for this is identical as the one for the position estimate except we have to incorporate time because this is a rate (m/s) [3]

$$\text{new velocity} = \text{old velocity} + \frac{1}{3}\frac{\text{measurement} - \text{predicted velocity}}{1\text{second}} \tag{3}$$

In summary, the way we updated our estimates can be expressed as an algorithm: [3]

Initialization

1. Initialize the state of the filter
2. Initialize our belief in the state

Predict

1. Use system behavior to predict state at the next time step
2. Adjust belief to account for the uncertainty in prediction

Update

1. Get a measurement and associated belief about its accuracy
2. Compute residual between estimated state and measurement
3. New estimate is somewhere on the residual line

In this example, the way we chose the scaling factors $\frac{1}{4}$ and $\frac{1}{3}$ are somewhat arbitrary, but this initialize-predict-update process and the idea of taking a measurement and a prediction and scaling both by some factor to update an estimate is the same for all Bayesian filters, which include both this simple example and Extended Kalman Filters. The main difference between the Kalman filter and this simple example lies in the complexity of the system and how these scaling factors are chosen, which will be explained more thoroughly in later sections. [3]

2.2 Bayes' Theorem: Discrete Version

The Kalman filter belongs to a famaly of filters called Bayesian filters (filters that estimate a the probability density function for a state by incorporating measurements and a mathematical process model). There's a lot of ground to cover in Bayesian statistics, but in this section, we will focus on building intuition for why Bayes' Theorem is so useful for updating our predictions. [3]

2.2.1 NEATO Position: Noiseless Example

Let's begin with a simple example. Taking from the previous example, let's say that we have a NEATO sitting on a line, but this time there are markers at some of the locations. Using a lidar, we can determine whether there is a marker at the NEATO's current location. How do we use this data to determine where the NEATO is? [3]

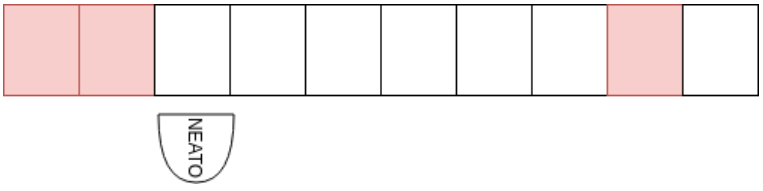

Figure 2.5: Visualizing the NEATO in an environment with 10 positions and red markers

To keep this problem simple, we will assume there are only 10 positions along this line, which we will number 0 to 9.

When we first start listening to the sensor we don't have any idea about the location of the NEATO, so the probability it is in any given position is 1/10. In Bayesian statistics, this is called the prior. [3]

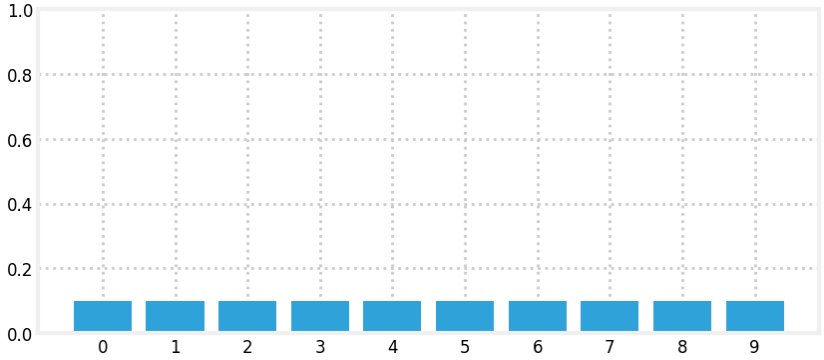

Figure 2.6: Visualizing prior estimate of NEATO location - NEATO is equally likely to be at any location [3]

We start up the lidar and in the first reading we detect a marker. From this we can conclude that the NEATO is in front of a marker, but which one? We don't know which marker the NEATO is in front of, and since it's equally likely the NEATO is in front of any given marker, we can assign a probability of 1/3 to each marker. [3]

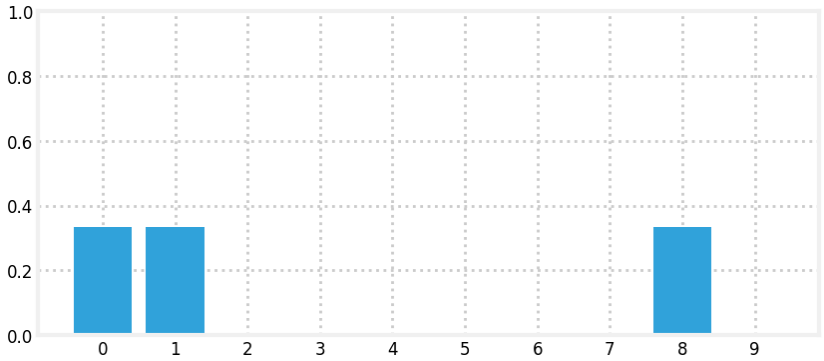

Figure 2.7: Visualizing posterior estimate of NEATO location - NEATO in front of a marker, and is equally likely to be at any marker [3]

Suppose now we know that the NEATO moved right and we detect a marker again. Now our data looks like this

- marker
- move right

- marker

In our layout there is only one place you could get this sequence, and that is at the left end. Therefore we can confidently state that the NEATO is in front of the second marker. [3]

Although this example is simple, it possesses the key takeaway of working with Bayesian filters - even though the first sensor reading gave us low probabilities for the NEATO's location, after a position update and another sensor reading we know more about its location. For a more complex example (ex: a line with 100 markers distributed over 1000 locations), it would take more data to determine the NEATO's location, but after several data points it still should be possible. [3]

2.2.2 NEATO Position: Noisy Sensor Example

In the previous example, we assumed that the sensor readings were always correct, but what if there was noise in the sensor readings? Put another way, there's now a probability that our sensor readings may be incorrect. How can we reach a conclusion if we are always unsure?

Once again, we can use probabilities. We previously assigned a probabilistic belief to the location of the NEATO with sensor data; now we have to incorporate the additional uncertainty caused by the sensor noise. [3]

Let's say that after testing our lidar that it is 3 times more likely to identify markers correctly than incorrectly. In Bayesian statistics, this is called the likelihood. We should scale the probability distribution by 3 where there is a door. We get [3]

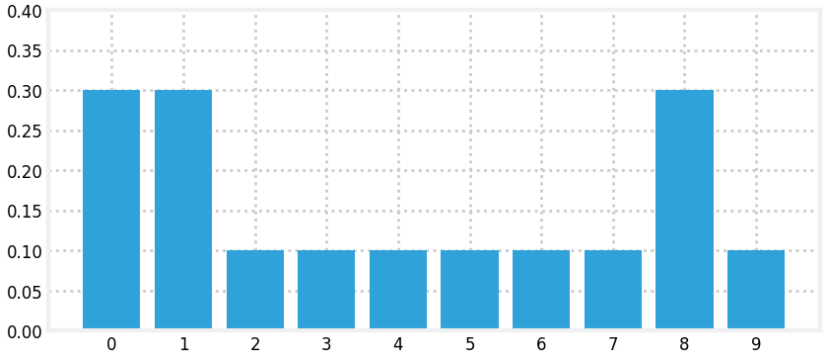

Figure 2.8: Unnormalized probability distribution incorporating uncertainty in measurement [3]

This is not probability distribution because it does not sum to 1, but that can easily be fixed by normalizing the data so that it does. We can do this by dividing each element by the sum of the elements in the list. From this we get

As we expect, the sum of the output is now 1, and the probability of the NEATO being in a location with a marker vs without is still three times larger.

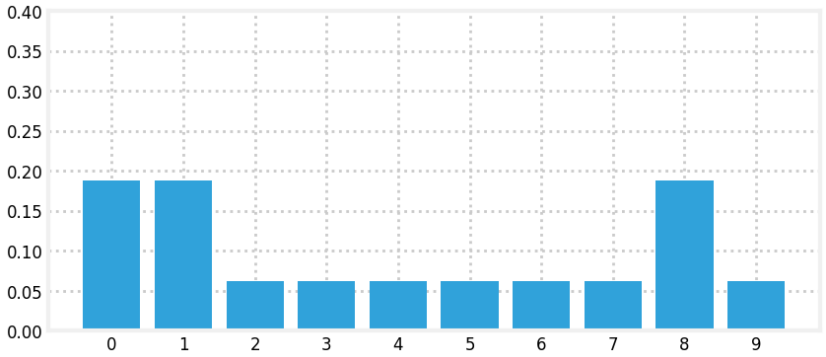

Figure 2.9: Normalized probability distribution incorporating uncertainty in measurement [3]

This result is called the posterior probability distribution - the probability distribution after incorporating the measurement information. [3]

2.2.3 Bayes' Theorem Formalization

The Bayes' Theorem can be written as: [3]

$$\texttt{posterior} = \frac{\texttt{likelihood} \times \texttt{prior}}{\texttt{normalization}} \tag{4}$$

In this example, the prior probability distribution was the assumption that all positions were equally likely. The likelihood (how likely a measurement is given the data), was that the measurement was 3 times more likely to be correct than incorrect, and the normalization made the distribution sum up to 1. [3]

We can see that even with noisy sensor data, implementing a Bayesian filter allowed us to predict with reasonable confidence the location of the NEATO. [3]

2.3 Gaussians

In the previous section, the Bayesian filter we implemented was discrete - the NEATO could only be at either position 2 or 3, not 2.34 or something in-between. If we wanted to make this continuous by increasing the number of possible positions, this would quickly get computationally unfeasible. To implement a continuous version of the Kalman filter that's still computationally feasible, we look towards Gaussians. [3]

There's a lot to be said about Gaussians, but in brief, they're used in Kalman filters because

- Gaussians only need two numbers to be fully defined, the mean μ and the variance σ^2
- Most probability distributions can be represented decently well with Gaussians

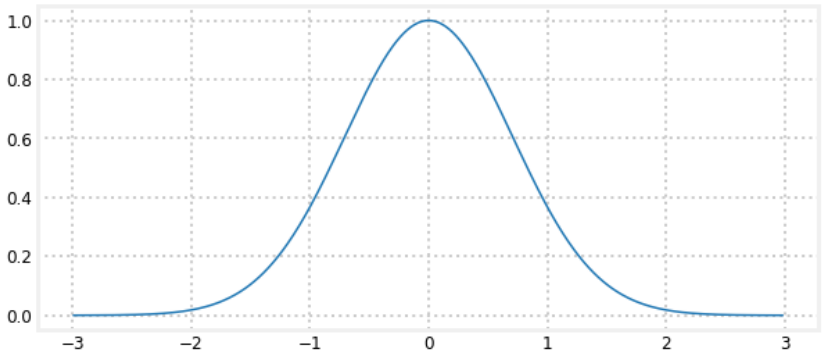

Figure 2.10: Gaussians have a familiar bell curve shape. Notice the continuity of data points! [3]

- Gaussians aren't linear functions, yet multiplying and adding Gaussians results in a Gaussian (though is sometimes unnormalized). This makes them much more computationally feasible than most nonlinear system

In the previous section, we represented probability distributions as discrete "bars" of probability at any given location. Using Gaussians, we can replace these discrete probabilities with a Gaussian function that provides the probability of any given point. This is illustrated in the following figure: [3]

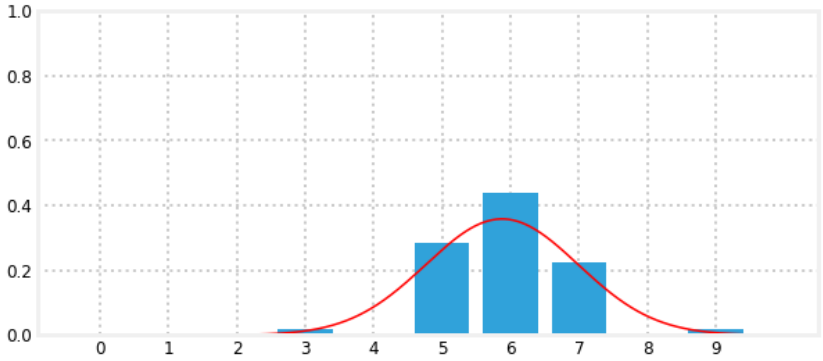

Figure 2.11: Discrete "bars" of probability represented with Gaussians [3]

If we had data from multiple sensors, we can merge the information to provide a more informative center and spread via a product, which can be computed with the equations [3]

$$\mu = \frac{\sigma_1^2\mu_2 + \sigma_2^2\mu_1}{\sigma_1^2 + \sigma_2^2} \tag{5}$$

$$\sigma = \frac{\sigma_1\sigma_2}{\sigma_1^2 + \sigma_2^2} \tag{6}$$

We can also easily add two Gaussians using the equations [3]

$$\mu = \mu_1 + \mu_2 \tag{7}$$

$$\sigma^2 = \sigma_1^2 + \sigma_2^2 \tag{8}$$

Gaussians are typically dented in mathematical form as $\mathcal{N}(\mu, \sigma^2)$,

The ability to combine Gaussians using multiplication and addition is what makes a Kalman filter so elegant and computationally feasible. In the Kalman filter, we will use these Gaussians to combine the data from a prediction and a measurement to produce a single Gaussian we can further refine with more data. [3]

For more information on Gaussians see Chapter 3 of the Kalman filters notebook [3]

2.4 One Dimensional Kalman Filters

From the previous two sections, we built intuition for how a g-h filter (a Bayesian filter) works by using a discrete example and explained how discrete probability distributions can be represented by Gaussians, which are easy to manipulate and are fully defined by only two numbers. In this section, we will combine these topics and explain how to use Gaussians to implement a Bayesian filter. That's all the Kalman filter is - a Bayesian filter that uses Gaussians. [3]

The prediction and update steps are the same for discrete and Gaussian probability distributions, but the math changes slightly due to the fact that addition and multiplication work a bit differently as we saw in the previous section. [3]

Predictions with Gaussians

Looking at the NEATO example we used previously, we can now use Gaussians to replace the histogram of probabilities and simplify our calculations. By doing this, we can replace hundreds to thousands of numbers with a single pair of numbers $x = \mathcal{N}(\mu, \sigma^2)$ [3]

We use Newton's equation of motion to compute current position based on the current velocity and previous position:

$$\bar{x}_k = x_{k-1} + v_k\Delta t \tag{9}$$

$$= x_{k-1} + f_x \tag{10}$$

where $\bar{x}_k$ is the prediction, x_{k-1} is the previous estimate, and $v_k\Delta t$ is the expected amount of movement based on the previous velocity estimate. [3]

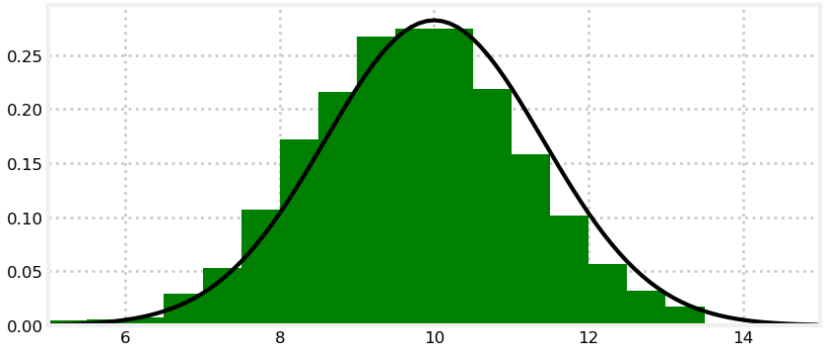

Figure 2.12: The large set of numbers in a discrete probability distribution can be represented by just two Gaussians [3]

Since we are uncertain about the current position and velocity, we need to express the uncertainty with a Gaussian. This can be done by defining x as a Gaussian. For example, if we think the NEATO is at 10 m, and the standard deviation of our uncertainty is 0.2 m, we get $x = \mathcal{N}(10, 0.2^2)$. [3]

For uncertainty in movement, we can define f_x as a Gaussian. If the NEATO's velocity is 15 m/s, the epoch (Δt) is 1 second, and the standard deviation of our uncertainty is 0.7 m/s, we get $f_x = \mathcal{N}(15, 0.7^2)$. [3]

Now, we have two separate predictions that need to be merged into one prediction. The equation to compute the prior probability distribution is [3]

$$\bar{x} = x + f_x \tag{11}$$

Here we need to find the sum of two Gaussians. Recall that the equations for adding Gaussians are [3]

$$\mu = \mu_1 + \mu_2 \tag{12}$$
$$\sigma^2 = \sigma_1^2 + \sigma_2^2 \tag{13}$$

In our example, we have: [3]

$$x = \mathcal{N}(10, 0.2^2) \tag{14}$$
$$f_x = \mathcal{N}(15, 0.7^2) \tag{15}$$

If we add these we get: [3]

$$\bar{x} = \mu_x + \mu_{f_x} = 10 + 15 = 25 \tag{16}$$
$$\bar{\sigma}^2 = \sigma_x^2 + \sigma_{f_x}^2 = 0.2^2 + 0.7^2 = 0.53 \tag{17}$$

Updates with Gaussians

Now, we need to update our probability distribution with the new measurements. Recall that this is called the posterior, which can be calculated with the equation [3]

$$\texttt{posterior} = \frac{\texttt{likelihood} \times \texttt{prior}}{\texttt{normalization}} \tag{18}$$

This can be rewritten in mathematical notation as

$$x = \|\mathcal{L}\bar{x}\| \tag{19}$$

We've just shown that we can represent the prior with a Gaussian. What about the likelihood? The likelihood is the probability of the measurement given the current state. We've learned how to represent measurements as Gaussians. For example, maybe our sensor states that the NEATO is at 23 m, with a standard deviation of 0.4 meters. Our measurement, expressed as a likelihood, is $z = \mathcal{N}(23, 0.16)$. [3]

Now, we need to multiply two Gaussians. Recall that the product of two Gaussians can be computed with the equations [3]

$$\mu = \frac{\sigma_1^2\mu_2 + \sigma_2^2\mu_1}{\sigma_1^2 + \sigma_2^2} \tag{20}$$

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \tag{21}$$

We can put this in Bayesian terms in the form [3]

$$\mathcal{N}(\mu, \sigma^2) = \|prior \cdot likelihood\| \tag{22}$$

$$= \mathcal{N}(\bar{\mu}, \bar{\sigma}^2) \cdot \mathcal{N}(\mu_z, \sigma_z^2) \tag{23}$$

$$= \mathcal{N}(\frac{\bar{\sigma}^2\mu_z + \sigma_z^2\bar{\mu}}{\bar{\sigma}^2 + \sigma_z^2}, \frac{\bar{\sigma}^2\sigma_z^2}{\bar{\sigma}^2 + \sigma_z^2}) \tag{24}$$

Kalman Gain

Recall that the form of the g-h filter in the first section involved choosing a constant to represent the amount we believed the measurement versus the prediction. In the first few examples, we chose arbitrary constants based on intuition, but now we have the information we need to calculate this constant. In the Kalman filter, this constant is called the Kalman Gain, which is given by the equation [3]

$$K = \frac{\bar{\sigma}^2}{\bar{\sigma}^2 + \sigma_z^2} \tag{25}$$

The derivation of this equation can be found in Chapter 4 under the sub-section Kalman gain in the Kalman filters notebook. [3]

Our process for this predict-update cycle of the Kalman filter can be summarized as:

Initialization

1. Initialize the state of the filter
2. Initialize our belief in the state

Predict

1. Use system behavior to predict state at the next time step
2. Adjust belief to account for the uncertainty in prediction

Update

1. Get a measurement and associated belief about its accuracy
2. Compute residual between estimated state and measurement
3. Compute scaling factor based on whether the measurement or prediction is more accurate
4. Set state between the prediction and measurement based on scaling factor
5. Update belief in the state based on how certain we are in the measurement

2.5 Multivariate Gaussians

Up to this point, our examples have been one dimensional. To extend the Kalman filter to higher dimensions, the mean and the variance are no longer scalars. The new mean is represented by a vector containing the mean for the data along each dimension and the new variance is represented by a covariance matrix. [3]

Consider we have a two dimensional vector for the x and y coordinates of a NEATO. The mean and covariance would be [3]

$$\boldsymbol{\mu} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{26}$$

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{\text{xx}} & \Sigma_{\text{xy}} \\ \Sigma_{\text{yx}} & \Sigma_{\text{yy}} \end{bmatrix} \tag{27}$$

If the off-diagonal elements were nonzero, there would be a correlation between the two state elements. This causes the ellipse to thin along a diagonal due to this linear correlation. [3]

As you can see, the distribution has a bell curve along multiple dimensions. [3]

To combine multidimensional Gaussians, we use a new set of equations [3]

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}_2(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_2 \tag{28}$$

$$\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\Sigma}_2 \tag{29}$$

In context, this means that we can take more than one measurement and combine all the information we have into one Gaussian probability distribution.

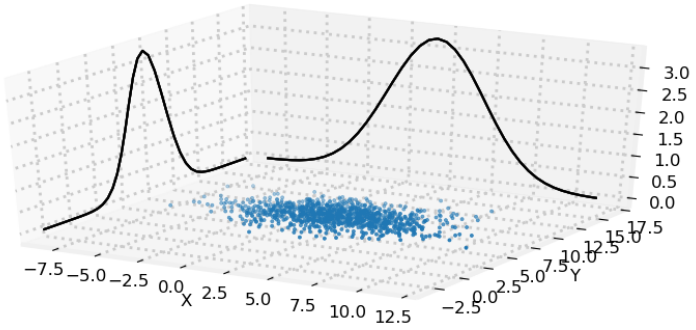

Figure 2.13: The Gaussian shows the variance and mean in two dimensions, hence the two bell curves. Notice that the data in the X-Y plane is clustered in the shape of an ellipse

2.6 Extended Kalman Filters

The Extended Kalman filter (EKF) provides an extension to Kalman filters so that the same concepts can be applied to nonlinear systems. The original Kalman filter is almost identical to the EKF, so we cover the EKF in this section interjecting when the two are different. The EKF handles nonlinear systems by linearizing the system at the point of the current estimate with Jacobians, whereas the original Kalman filter only handles linear systems. Problems can be nonlinear in two ways - either in the process model or in the measurements. For example, rotation is a nonlinear motion model, and converting polar coordinates to Cartesian coordinates is also a nonlinear operation. [3]

The prediction step is similar to a uni-variate Kalman filter. Instead of prediction function, though, we use a matrix $\boldsymbol{F}$. In the case of an EKF where the prediction function is nonlinear, $\boldsymbol{F}$ is still used to compute confidences, but a function f is used to compute the prediction and F is the jacobian of f. Since mapping is required to convert, for example, a state space denoting the coordinates of a landmark in global coordinates and a measurement space denoting the relative coordinates of the landmarks, we use a matrix $\boldsymbol{H}$ to map from a state to a measurement $\boldsymbol{z}$. Similar to prediction, in nonlinear systems a function h does the mapping and the jacobian H is used to transform confidences. The residual y is still calculated as the difference between the measurement and the predicted measurement. [3]

$$\dot{\mathbf{x}} = f(\boldsymbol{x}) \tag{30}$$

$$\hat{\boldsymbol{z}} = h(\hat{\boldsymbol{x}}) \tag{31}$$

$$\boldsymbol{y} = \boldsymbol{z} - \hat{\boldsymbol{z}} \tag{32}$$

where $\mathbf{F}$ is the fundamental matrix. This form of the equations allow us to compute the state at step k given a measurement at step k and the state estimate at step $k-1$. [3]

Updating the covariance involves finding 3 uncertainties. The first is the uncertainty $\hat{\boldsymbol{\Sigma}}$ in our prediction, the uncertainty $\boldsymbol{S}$ in our predicted measurement, and $\boldsymbol{\Sigma}_t$ for our new state

uncertainty. The predicted state uncertainty is found by applying $\boldsymbol{F}$ to the original state uncertainty and adding the process noise. The predicted measurement uncertainty is found by applying $\boldsymbol{H}$ to the predicted state uncertainty and adding the measurement uncertainty. [3]

$$\hat{\boldsymbol{\Sigma}} = \boldsymbol{F}\boldsymbol{\Sigma}\boldsymbol{F}^T + \boldsymbol{Q} \tag{33}$$

$$\boldsymbol{S} = \boldsymbol{H}\hat{\boldsymbol{\Sigma}}\boldsymbol{H}^T + \boldsymbol{R} \tag{34}$$

$$\tag{35}$$

The difference between an EKF and a regular Kalman filter is that sometimes F and H must take the place of a nonlinear function f or h respectively. In this case, F or H is the jacobian of the functions f or h and the filter is now an EKF. To process, we need the Kalman gain K, the multivariate version of the one discussed in §2.4. Now, the equation for K is [3]

$$\hat{K} = \hat{\boldsymbol{\Sigma}}\boldsymbol{H}^T\boldsymbol{S}^{-1} \tag{36}$$

The confidence is difficult to unpack, but try to think of it as your predicted state confidence remapped to the measurement model and normalized by $\boldsymbol{S}$. With $\boldsymbol{K}$, it is now possible to find the new state estimate $\boldsymbol{x}_t$ and confidence $\boldsymbol{\Sigma}_t$. [3]

$$\boldsymbol{\Sigma}_t = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\hat{\boldsymbol{\Sigma}} \tag{37}$$

$$\boldsymbol{x}_t = \hat{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{y} \tag{38}$$

We can iteratively restart this process with the new state and confidence to track a robot in realtime. [3]

3 Quantitative Calculation

3.1 Motion Model

In many robotic platforms, the robot is expected to often maintain a constant velocity and constant angular acceleration. Suppose that we model the robot with a 5-dimensional vector $\boldsymbol{x}$.

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix} \tag{39}$$

To increment the position of the robot, we must add a value to x and y. Specifically, the components of the velocity are

$$\boldsymbol{v} = v \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{40}$$

The orientation is simply incremented by ωdt while velocity and angular velocity stay the same. Therefore, a motion model for this robot is [2]

$$\hat{\boldsymbol{x}} = f(\boldsymbol{x}) \tag{41}$$

$$= \begin{bmatrix} x + v\cos(\theta)dt \\ y + v\sin(\theta)dt \\ \theta + \omega dt \\ v \\ \omega \end{bmatrix} \tag{42}$$

However, this is not a linear function. In order to linearize this motion model, we obtain a linear approximation about the state of concern by taking the jacobian

$$F = \begin{bmatrix} 1 & 0 & -vdt\sin\theta & dt\cos\theta & 0 \\ 0 & 1 & vdt\cos\theta & dt\sin\theta & 0 \\ 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{43}$$

Now we can multiply this jacobian by $\boldsymbol{x}$ to estimate the next state. This is important in the Extended Kalman filter, where we must predict the next measurement using a nonlinear model.

3.2 Measurement Model

The measurements consist of r and θ of each object relative to the NEATO's coordinate frame. Since we need the x and y position of each object in the global frame to make our map, we need a way to convert from (r, θ) to (x, y) coordinates. This relationship is illustrated in the figure below:

We can describe also this relationship with the equations

$$x_i = x + r\cos(\theta + \phi) \tag{44}$$

$$y_i = y + r\sin(\theta + \phi) \tag{45}$$

where (x_i, y_i) are the x and y coordinates of the object, (x, y) are the x and y coordinates of the NEATO, r is the distance of the object from the NEATO, θ is the heading of the NEATO, and ϕ is the angle of the object relative to the NEATO.

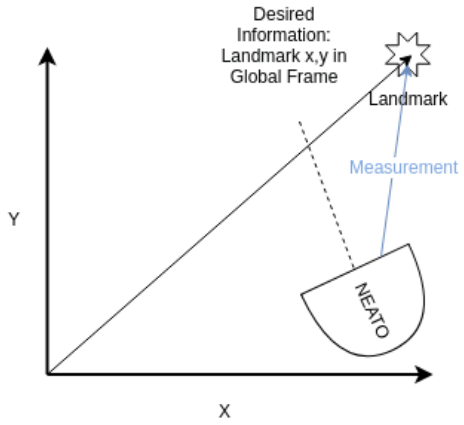

Figure 3.1: The NEATO doesn't directly measure the (x, y) coordinates of the landmark. These coordinates can be inferred from the (r, θ) measurement observed with respect to the NEATO

3.3 Putting it all together: SLAM with a Lidar

Process based on Monocular SLAM implementation. [1]

When using SLAM to navigate robots through an unknown environment, being able to map the environment while tracking the robot's movement through the map is often a desirable function. An example of a robot where this functionality is especially useful is in robot vacuums like the NEATO, which can use lidar data to map out its surroundings while keeping track of the ground it's already covered.

3.3.1 State Vector

In order to do this, we need to know the state of the NEATO at any given time. The NEATO's state x_n can be represented by the 5-dimensional vector [1]

$$\boldsymbol{x}_n = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix} \tag{46}$$

where x, y is the 2D position, θ is the heading, v the linear velocity and ω the angular velocity.

We also need to keep track of the location of the obstacles in order to create a map. Although the true location of each obstacle is fixed, since we don't know the location of the obstacles, we will continuously update our estimate of where we think each obstacle is as we collect more data. Because of this, we will also need to include the location of obstacles as part of the state vector. The overall state can be represented by the vector [1]

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_n \\ \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_N \end{bmatrix} \tag{47}$$

where $\boldsymbol{x}_n$ is the NEATO's state, $\boldsymbol{y}_1$ is the x and y position of the first obstacle, and $\boldsymbol{y}_N$ is the x and y position of the Nth obstacle. [1]

Since $\boldsymbol{x}$ is merely an estimate and we don't know for certain what the state is, we need to keep track of the uncertainty in our measurements. Recall that gaussians need two numbers to be fully defined, the mean μ and the variance σ^2 (§2.3). Also recall that we can represent the variance for a multidimensional system (§2.5) by the covariance matrix of $\boldsymbol{x}$, which is given by [1]

$$\boldsymbol{\Sigma} = \begin{bmatrix} \Sigma_{\mathrm{x}_c\mathrm{x}_c} & \Sigma_{\mathrm{x}_c\mathrm{y}_1} & \cdots & \Sigma_{\mathrm{x}_c\mathrm{y}_N} \\ \Sigma_{\mathrm{y}_1\mathrm{x}_c} & \Sigma_{\mathrm{y}_1\mathrm{y}_1} & \cdots & \Sigma_{\mathrm{y}_1\mathrm{y}_N} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{\mathrm{y}_N\mathrm{x}_c} & \Sigma_{\mathrm{y}_N\mathrm{y}_1} & \cdots & \Sigma_{\mathrm{y}_N\mathrm{y}_N} \end{bmatrix} \tag{48}$$

3.3.2 Overview

As a review, the Kalman filter can be summarized in these steps (§2.4).

Initialization

1. Initialize the state of the filter
2. Initialize our belief in the state

Predict

1. Use system behavior to predict state at the next time step
2. Adjust belief to account for the uncertainty in prediction

Update

1. Get a measurement and associated belief about its accuracy
2. Compute residual between estimated state and measurement
3. Compute scaling factor based on whether the measurement or prediction is more accurate
4. Set state between the prediction and measurement based on scaling factor
5. Update belief in the state based on how certain we are in the measurement

We will refer to these these steps in the following sections. To understand the reasoning behind each of these steps, refer to the Kalman Filter section and the Extended Kalman filter section (§2.6).

3.3.3 State Prediction Step

Objective: Use system behavior to predict state at the next time step

Recall that one of the key elements of a Kalman filter is incorporating the prior of a measurement in updating the prediction. The purpose of this prediction step is to estimate the state at current time t based on the previous estimated state. During the prediction step, the robot's state is is updated according to a motion model which we calculated earlier.

Recall that our motion model states that

$$\boldsymbol{x}_{pred} = \begin{bmatrix} x + v\cos(\theta)dt \\ y + v\sin(\theta)dt \\ \theta + \omega dt \\ v \\ \omega \end{bmatrix} \tag{49}$$

where $\boldsymbol{x}$ is the new predicted state, and x, y, θ, v, ω are the state values from the previous state. We can represent this mapping from the old state vector to the new predicted state as a function f.

Since the locations of the tracked obstacles are not expected to change in the global frame, the old keypoint locations are predicted to be the same as the new keypoint locations. (These estimates get refined later with measurement data) Because of this, our motion model prediction only influences the values of the robot's state, not the obstacles.

Objective: Adjust belief to account for the uncertainty in prediction

Since the prediction step only influences the NEATO state, the covariance $\Sigma_{\mathrm{x}_c\mathrm{x}_c}$ only needs to be updated for the NEATO. This is calculated through F_t –the Jacobian matrix of the function f with respect to the previous NEATO state $\boldsymbol{x}_n$ and $\boldsymbol{Q}_t$, which increments the uncertainty. $\boldsymbol{Q}_t$ is the uncertainty in our prediction. For example, if we are sure that our robot's heading will be predicted accurately within 30 degrees, the corresponding entry along the diagonal should be $\left(\frac{\pi}{6 \cdot 3}\right)^2$. Finding $\boldsymbol{Q}_t$ can be found either quantitatively as above or qualitatively in simulation.

$$\boldsymbol{F}_t = \frac{\partial f}{\partial \mathrm{x}_n} \tag{50}$$

$$\boldsymbol{\Sigma}^{pred}_{\mathrm{x}_n\mathrm{x}_n} = \boldsymbol{F}_t \boldsymbol{\Sigma}_{\mathrm{x}_n\mathrm{x}_n} \boldsymbol{F}_t^{\mathrm{T}} + \boldsymbol{Q}_t \tag{51}$$

3.3.4 Feature Prediction

Objective: Get a measurement and associated belief about its accuracy

A key part of using a Kalman Filter for SLAM is the prediction of the object measurements before actually taking any measurements. An expected observation h_i is computed for each object as a function h of the predicted new robot pose x_n and the estimates of each object

location in the global frame y_i. A corresponding Kalman Gain is also computed, which specifies the degree to which the incoming observation corrects the current state estimation (see Kalman Theory §2.4 for details).

Recall from the measurement model section (§3.2) that the relationship between the measurement and 2D location of each object is

$$x_i = r_i \cos(\theta + \phi_i) \tag{52}$$
$$y_i = r_i \sin(\theta + \phi_i) \tag{53}$$

where (x_i, y_i) are the x and y coordinates of the object r_i is the distance of the object from the NEATO, θ is the heading of the NEATO, and ϕ_i is the angle of the object relative to the NEATO. To predict the measurement given the expected (x_i, y_i), we can solve these equations for (r_i, ϕ_i) instead. The measurement function h to map from estimated x and y coordinates to predicted measurements can be described by

$$\boldsymbol{h}_i = h(\mathrm{x}_{pred}) = \begin{bmatrix} r_i \\ \phi_i \end{bmatrix} = \begin{bmatrix} \sqrt{x_i^2 + y_i^2} \\ \tan^{-1}\frac{y_i}{x_i} - \theta \end{bmatrix} \tag{54}$$

From the measurement function h the expected position (r_i, ϕ_i) for the feature $\boldsymbol{y}_i$ is now known: instead of trying to match $\boldsymbol{y}_i$ with every possible location in the lidar scan, it is now possible to search only in the area around the predicted position. In addition, the covariance matrix has information regarding the uncertainty of y_i : if the feature's position is well known, and the uncertainty is small, the area to search will be consequently small. To get a more precise formulation of the uncertainty of the expected position and subsequently about the size of its search region the innovation covariance S_t is calculated:

$$\boldsymbol{S}_t = \dot{\boldsymbol{H}} \boldsymbol{\Sigma}_t \dot{\boldsymbol{H}}^{\boldsymbol{T}} + \boldsymbol{Q} \tag{55}$$

where $\boldsymbol{\Sigma}_t$ is the predicted covariance matrix, $\boldsymbol{H}_t^{'}$ is the Jacobian of h, and $\boldsymbol{Q}$ is a matrix which modulates sensor noise.

3.3.5 Map Update

Compute residual between estimated state and measurement z-h

Now that we've predicted the state of the NEATO, the covariance matrix for the state, and the measurements, we are now ready to compute the residual. The residual is simply

$$\boldsymbol{z}_t - h(\boldsymbol{x}_{pred}) \tag{56}$$

where z_t is the actual measurement.

Objective: Compute scaling factor based on whether the measurement or prediction is more accurate

Recall that this scaling factor is called the Kalman Gain, as referenced in the Kalman Filter theory section 2.4. This can be computed with

$$\boldsymbol{K}_t = \boldsymbol{\Sigma}_{tt}^T \boldsymbol{S}^{-1} \tag{57}$$

where K_t is the Kalman Gain, Σ_t is the predicted covariance matrix, H_t^T is the transpose of the Jacobian of h and S^{-1} is the innovation covariance we computed earlier. In other words, the Kalman gain is equal to the predicted state covariance times the transpose of the measurement mode times the inverse of system noise

Objective: Set state between the prediction and measurement based on scaling factor

Our new estimate can be computed with

$$\boldsymbol{x}_t = \boldsymbol{x}_{pred} + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{h}(x_{pred})) \tag{58}$$

This is the same computation as the simple g-h filter from earlier! (Only now it's multidimensional, nonlinear, and continuous with the carefully calculated Kalman gain instead of arbitrary constants - but nonetheless still in the same form)

Objective: Update belief in the state based on how certain we are in the measurement

Our new belief can be computed with

$$\boldsymbol{\Sigma_t} = (\boldsymbol{I} - \boldsymbol{K}_{tt})\boldsymbol{\Sigma}_{pred} \tag{59}$$

The new $\boldsymbol{x}_t$ and Σ_t will be used as the old estimation in the next EKF cycle.

4 Implementation

We will implement an extended Kalman filter for SLAM given noisy measurements of the relative locations of three landmarks in Cartesian coordinates. For simplicity, these coordinates are centered at the robot when measured, but the orientation of the robot is assumed to always be aligned with the global axes. Our robot will have a constant velocity and angular velocity model.

The implementation of an extended Kalman filter can be broken up into 3 steps: initialization, prediction, and updating. Our implementation assumes the state begins at 0, as in the robot begins at the x-axis with a linear and angular velocity of 0 and oriented along the x axis. We also assume that the three landmarks do not move. Our state vector stacks the x and y coordinates, orientation, velocity, angular velocity, and the x and y coordinates of the three landmarks. This requires 11 dimensions. Our initial state assumes that everything is 0.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{60}$$

We assume that we are fairly certain of our initial position within a meter, our initial velocity within a meter per second, our orientation within π rad, and our initial angular velocity within $\frac{\pi}{6}$ rad s^{-1}. We also assume that we are fairly certain that we know the locations of the landmarks within 2 meters. By fairly certian, we mean we are within 3 standard deviations. For example, if we are sure that π is the same as 3 standard deviations, we use $\left(\frac{\pi}{3}\right)^2$ This leads to an initial state covariance.

$$\boldsymbol{\Sigma}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0.001 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0.001 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \tag{61}$$

We also need a measurement noise and process noise matrix. In our scenario, we have a variance of 0.1 meters from our landmark scanner which gives us x and y coordinates.

$$\boldsymbol{R} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{62}$$

The process noise matrix was found qualitatively to be

$$\boldsymbol{Q} = \begin{bmatrix} \frac{\boldsymbol{I}_5}{100} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{63}$$

With these matrices, we can move onto the prediction step. Describing our prediction $\hat{\boldsymbol{x}} = F(\boldsymbol{x})$

$$\hat{\boldsymbol{x}} = f(\boldsymbol{x}) \tag{64}$$

$$= \begin{bmatrix} x + v\cos(\theta)dt \\ y + v\sin(\theta)dt \\ \theta + \omega dt \\ v \\ \omega \\ x_1 \\ y_1 \\ \vdots \\ y_3 \end{bmatrix} \tag{65}$$

Since the prediction function is nonlinear as shown by the presence of sin and cos in the above equations, we need to linearize them about a state using the jacobian of f at $\boldsymbol{x}$

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & -v\sin(\theta)dt & \cos(\theta)dt & 0 & 0 & \dots \\ 0 & 1 & v\cos(\theta)dt & \sin(\theta)dt & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & dt & 0 & \dots \\ 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{66}$$

We now have our prediction $\hat{x} = f(x)$. To update our covariance, we use $\hat{\Sigma} = \boldsymbol{F}\boldsymbol{P}F^T + \boldsymbol{Q}$ That finishes the prediction step.

The next step is the update step. We begin by adding the measurement noise to the predicted covariance to find the system noise $\boldsymbol{S} = \boldsymbol{H}\hat{\boldsymbol{\Sigma}}\boldsymbol{H}^T + \boldsymbol{R}$. Next we find the Kalman gain $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{H}^T\boldsymbol{S}^{-1}$. In order to use the Kalman gain we must first compute the residual $\boldsymbol{y}$. Our measurement model simply translates the location of the landmarks in the state space to the robot coordinates, so

$$\boldsymbol{H} = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\ -1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{67}$$

Our residual is then $\boldsymbol{y} = \boldsymbol{z} - \boldsymbol{H}\hat{\boldsymbol{x}}$.

The updated state and covariances are then

$$\boldsymbol{x}_t = \hat{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{y} \tag{68}$$

$$\boldsymbol{\Sigma}_t = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\hat{\Sigma} \tag{69}$$

If we iteratively use this process on the new predicted state and state confidence, we can track the estimated location of the robot in realtime.

5 Results and Discussion

We generate a set of noisy sensor readings corresponding to the locations of randomly placed landmarks. The results of the simulation at 30 Hz in 5.1.

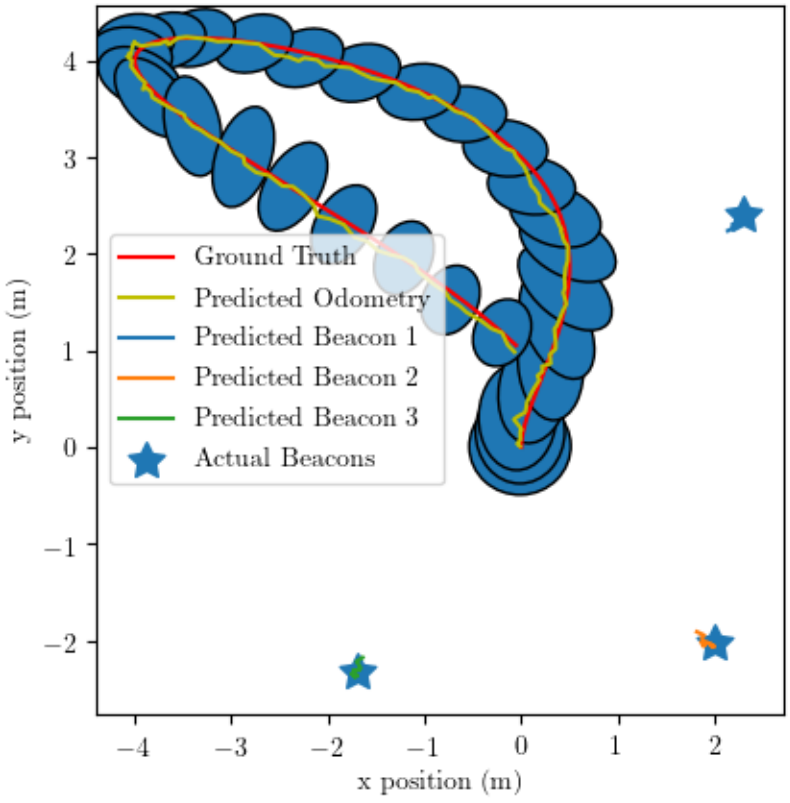

Figure 5.1: The noise-filtered predicted trajectory of the robot. The ellipses correspond to a 38.3% confidence interval, shown every 1.5 seconds.

The confidence intervals are large, often spanning half a meter from the mean. The first confidence ellipse is a circle, which makes sense as we have no notion of forward motion at the beginning and so we can be along any direction. The confidence intervals being wider across the path on straight runs shows that the model is unsure about turning. However, the means denoted by the orange line are almost identical to the ground truth, so practically our model does as expected.

In order to reduce the size of the confidence ellipses, we can add more landmarks. However, we decided on 3 because in real life finding 3 objects to use as markers is easy. We can also find a motion model that is more accurate that we can assign a higher confidence to. However, creating such a model is difficult to imagine, as most objects do not have a constant acceleration or other characteristic.

Another method of increasing confidence is giving our model a control input. More data is better no matter how unprecise, so exploring this route can give satisfactory results.

6 Conclusion

"Dogs don't move like that" – Roger Labbe [3]

When we look at noisy data, it is difficult to avoid smoothening out the data in our minds. This is because we know the system does not zig-zag like the noise does. This begs the question: How does the system move and how do we take that into account? The Kalman filter makes the motion model explict and using sound probabalistic modeling takes both measurements and predictions into account. This generalizes into more powerful ideas. For example, for SLAM, one can simply put the location of landmarks into a state space.

In the results, notice that there is time necessary to converge to the ground truth at the beginning and after the hairpin turn. This is because the nonlinearities of the system are large at those points. The EKF uses Jacobians to create an approximation of the new means and covariances. However, work has been done to intelligently generate points with which to calculate a more accurate estimation of means and covariances. This is the unscented Kalman filter. An implementation of SLAM using not an EKF, but a UKF should converge more quickly to the ground truth and more appropriately calculate nonlinearities.

7 Appendix

The code which implements this idea for a SLAM simulation is at `https://github.com/concavegit/ekf-slam-lidar`. This implementation takes into account not only the translating frame of reference, but also the rotation of the robot.

References

[1] Reid Davison AJ. Monocular slam. `https://www.doc.ic.ac.uk/~ab9515/monoslam.html`, 2007 (accessed December 2018).

[2] Thomas Hellström. Kinematics equations for differential drive and articulated steering. http://www8.cs.umu.se/kurser/5DV122/HT13/material/Hellstrom-ForwardKinematics.pdf, 2011 (accessed December 2018).

[3] Roger Labbe. Kalman and bayesian filters in python. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python, 2017 (accessed December 2018).