Raspberry Pi GoPiGo Robot EKF SLAM Implementation

Gopal Menon
Computer Science Department
Utah State University
Logan, Utah 84322
Email: gopal.menon@aggiemail.usu.edu

Abstract*—Robot motion is stochastic and not deterministic. A robot does not faithfully execute movement commands due to inaccuracies in its motion mechanism, approximations made in modelling its environment and incomplete modelling of the physics of its operating environment. Due to these reasons, the robot may not end up exactly at the location where it should have been if the movement command had been faithfully executed. The robot position may be thought of as a two dimensional probability distribution. This probability distribution will have a mean that will be equal to the expected robot location based on the movement command. The covariance will be the measure of how faithfully it executes commands. Due to this uncertainty in movement, with each motion command, the location uncertainty increases and after a while the robot gets lost. In some applications, robots need to be able to go into an unknown environment and explore it. This is a difficult problem due to the issue described above. If a map of the environment is available, the robot can use sensors to find its location in the environment. Conversely, if the robot knows where it is, it can generate an environment map. This is a sort of chicken and end problem. Simultaneous Localization and Mapping (SLAM) solves this problem. An Extended Kalman Filter (EKF) can be used to reduce robot location uncertainty. This is done by combining the predicted robot location based on the equation of robot motion with the measured location based on sensor readings. The two robot location estimates are Gaussian distributions. These two distributions are combined and the resulting distribution will have a mean that is closer to the location estimate with the lower uncertainty and the uncertainty of resulting location will be lower than either of the two distributions (see VII-A). This document describes the implementation of EKF SLAM using a robot.

I. Introduction

A. The robot platform

The robot used for this implementation is based on a Raspberry Pi and is manufactured and sold by Dexter Industries [1]. The robot is equipped with an ultrasonic sensor that is capable of measuring distances accurately to 250 cm. The sensor is mounted on a servo mechanism that can point the sensor in any direction ahead of the robot. The robot moves through use of two wheels that can be independently controlled by motors. Each wheel has an encoder using which the robot can issue commands for a wheel to move a fixed number of steps. The robot also has a third castor wheel that can move in any direction, but is not powered. The robot will issue commands to move and it can sense its surroundings using the ultrasonic sensor.

B. EKF SLAM

A Kalman filter is a Gaussian filter algorithm that can use two Gaussian distributions for a computed value and merge them together to form a distribution that has lesser variance than either of the two distributions. A Kalman filter however only works in the case of systems where the mechanics are linear. This is where the Extended Kalman Filter (EKF) comes in. An EKF uses an approximation to find the resultant distribution after a prior distribution is transformed by a non-linear function. The robot movement and landmark sensing is governed by trigonometric functions that are based on the angle of the robot orientation. These functions are non-linear since they are trigonometric.

The robot will compute its expected position based on its prior position and the movement command. This is the prediction step. It will also make independent estimates of its position using the bearing and range to known landmarks returned by its ultrasonic sensor. These two measurements will be combined through use of an EKF in such a way that the position uncertainty is reduced. The Kalman Gain or the reduction in the robot position uncertainty is inversely proportional to the sensor uncertainty. The better the sensor, the higher will be the Kalman Gain and the less uncertainty there will be in the robot position [4].

II. Problem Statement

A. Location Uncertainty

As described above, robot motion is stochastic and not deterministic. Due to uncertainty in movement, with each motion command, the location uncertainty increases and after a while the robot gets lost.

Figure 1 shows the location uncertainty problem. Before the robot starts moving, it knows exactly where it is. So it's location belief is a single point. After each movement, its location uncertainty increases. This is shown by the location probability distribution that changes from a point to a larger area with the first step and the uncertainty, shown by an ellipse shaped point distribution, increasing in size with each step. In the final position of the robot, the uncertainty is the largest as it accumulates the added uncertainty with each

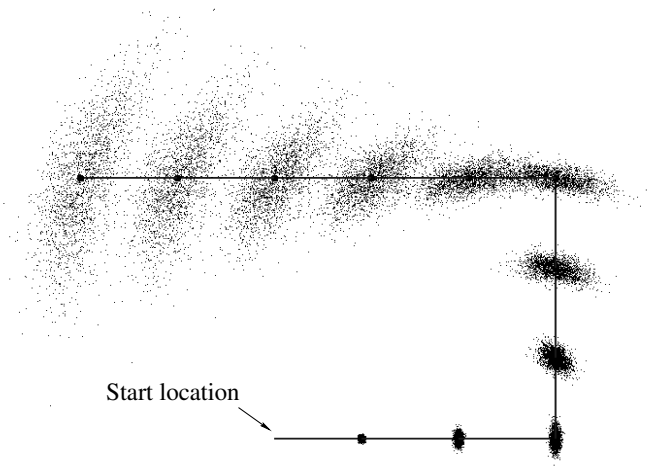

Fig. 1. Sampling approximation of the position belief for a non-sensing robot [3]. The solid line displays the actions, and the samples represent the robots belief at different points in time.

step.

This increasing uncertainty is a problem that eventually results in a lost robot that does not know where it is.

III. Approach/Methods

The following sections provide some background material that is required for understanding the solution to the robot localization problem.

A. The Bayes Filter

The Bayes Filter is an algorithm used for calculating beliefs [3]. In our case it is the belief of the robot position. This algorithm is used to compute the belief probability distribution bel from measurement and control data. The measurement data is obtained from sensors and the control data is obtained from the robot motion command or telemetry readings.

The robot will keep track of its current location as it moves. When it issues a movement command, it can compute its expected location after the move command is executed. This is the control data. It can use its sensors and obtain the range and bearing to known landmarks. Using this, it can compute its expected location. This is the measurement data.

Bayes Rule says that

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{1}$$

Since the denominator in Bayes Rule does not depend on x, the factor $\frac{1}{p(y)}$ can be written as η. So Bayes Rule in equation 1 can be written as

$$p(x|y) = \eta p(y|x)p(x) \tag{2}$$

Bayes Rule 1 can also be written as

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)} \tag{3}$$

as long as $p(y|z) > 0$.

Applying 3 on the robot position x_t at time t for measurements $z_{1:t}$ and commands $u_{1:t}$ gives us

$$\begin{aligned} p(x_t|z_{1:t}, u_{1:t}) &= \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) \end{aligned} \tag{4}$$

Since the measurement z_t depends on position x_t and any past measurement $z_{1:t}$ or command $u_{1:t}$ does not provide any additional information on the measurement, we can say that

$$p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{5}$$

This allows us to simplify 4 as

$$\begin{aligned} p(x_t|z_{1:t}, u_{1:t}) &= \eta p(z_t|x_t)p(x_t|z_{1:t-1}, u_{1:t}) \\ &= \eta p(z_t|x_t)\overline{bel}(x_t) \end{aligned} \tag{6}$$

where

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{7}$$

Using the theorem of total probability

$$p(x) = \int p(x|y)p(y)dy \tag{8}$$

equation 7 can be written as

$$\overline{bel}(x_t) = \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})p(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1} \tag{9}$$

If we know the state x_{t-1}, past measurements and controls have no bearing on the state x_t. So

$$p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{10}$$

We can see that command u_t can be omitted from the set of conditioning variables in $p(x_{t-1}|z_{1:t-1}, u_{1:t})$. This gives us the recursive update equation 7 as

$$\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)p(x_{t-1}|z_{1:t-1}, u_{1:t-1})dx_{t-1} \tag{11}$$

This reduces to

$$\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)bel(x_{t-1})dx_{t-1} \tag{12}$$

B. The Bayes Filter Algorithm

Based on the above mathematical derivation for a Bayes Filter, we can write down the Bayes Filter Algorithm as follows [3]

Algorithm 1 Bayes Filter Algorithm

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

The belief $bel(x_t)$ at time t is computed from the belief $bel(x_{t-1})$ at time $t-1$. The input to the algorithm is the belief bel at time $t-1$, along with the most recent control value u_t and the most recent measurement z_t. Its output is the belief $bel(x_t)$ at time t.

The Bayes Filter Algorithm consist of two essentials steps. In line 3, which is the prediction step, it calculates its belief over state x_t based on the prior belief over state x_{t-1} and the control u_t. The belief $\overline{bel}(x_t)$ that the robot assigns to state x_t is obtained by the integral (or in other words the sum) of the product of two distributions: the prior assigned to x_{t-1}, and the probability that control u_t induces a transition from x_{t-1} to x_t.

The second step of the Bayes Filter is called the measurement update. In line 4, the Bayes Filter algorithm multiplies the belief $\overline{bel}(x_t)$ by the probability that the measurement z_t may have been observed. It does this for every hypothetical posterior state x_t.

C. The Kalman Filter

The Kalman Filter is a technique for implementing a Bayes Filter [3]. This is used for filtering and prediction in linear Gaussian systems.

A probability distribution of a one-dimensional normal distribution with mean μ and variance σ^2 is given by the following Gaussian function

$$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{13}$$

The Normal distribution 13 is for a scalar x. In the case of a robot, the robot location on a flat surface is given by $\begin{bmatrix} x & y & \theta \end{bmatrix}^T$ where x is the x coordinate, y is the y coordinate and θ is the robot orientation. This vector is called the robot pose. A Normal distribution over such a vector is called *multivariate*. These distributions have the following form

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{14}$$

where μ is the mean vector that has the same dimensionality as the state x. Σ is the covariance matrix and its dimension is the dimensionality of the state x squared. Equation 14 is a generalization of equation 13. Both definitions are equivalent if x is a scalar and value and $\Sigma = \sigma^2$.

For a Kalman filter, in posteriors are Gaussian if the following three properties hold

1) The state transition probability $p(x_t|x_{t-1}, u_t)$ must be a linear function of its arguments with added Gaussian noise. This is expressed by the following equation

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{15}$$

where A_t and B_t are matrices. A_t has the dimension equal to the square of the state vector, B_t has the dimension of $n \times m$, where n is the dimension of the state vector and m is the dimension of the control variable u_t. The random variable ε_t models the uncertainty introduced by the state transition and depends on the robot mechanics. Its mean is zero and covariance will be denoted by R_t. The state transition probability distribution can be obtained by plugging in 15 into 14. The mean of the posterior state is given by $A_t x_{t-1} + B_t u_t$ and the covariance by R_t.

2) The measurement probability $p(z_t|x_t)$ must also be linear in its arguments with added Gaussian noise

$$z_t = C_t x_t + \delta_t \tag{16}$$

Here C_t is a matrix of size $k \times n$, where k is the dimension of the measurement vector z_t. The vector δ_t describes the measurement noise. The distribution of δ_t is a multivariate Gaussian with zero mean and covariance Q_t.

3) Finally the initial belief $bel(x_0)$ must be normally distributed. We can denote the mean of this belief by mean μ_0 and covariance Σ_0.

The Kalman filter algorithm is given below [3]

Algorithm 2 Kalman Filter Algorithm

1: **procedure** KALMAN FILTER ALGORITHM($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2: $\overline{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3: $\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4: $K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$
5: $\mu_t = \overline{\mu}_t + K_t(z_t - C_t \overline{\mu}_t)$
6: $\Sigma_t = (I - K_t C_t)\overline{\Sigma}_t$
7: return μ_t, Σ_t
8: **end procedure**

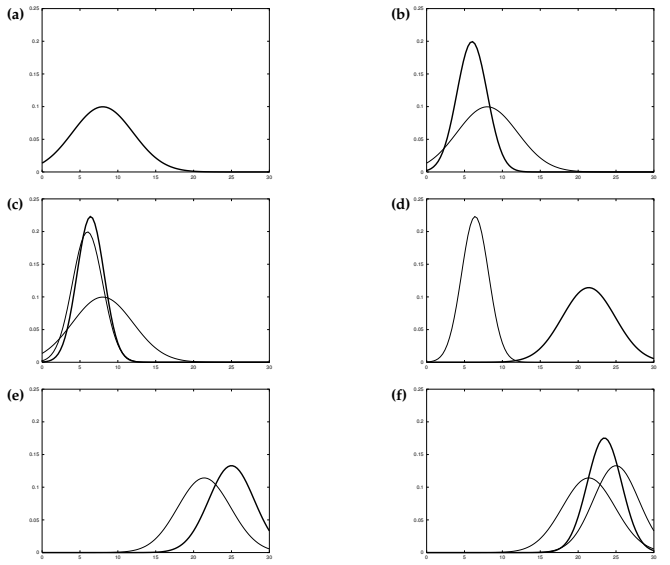

Fig. 2. Illustration of Kalman filters [3] (x-axis=location, y-axis=probability of being in that location): (a) initial belief, (b) a measurement (in bold) with the associated uncertainty, (c) belief after integrating the measurement into the belief using the Kalman filter algorithm, (d) belief after motion to the right (which introduces uncertainty), (e) a new measurement with associated uncertainty, and (f) the resulting belief.

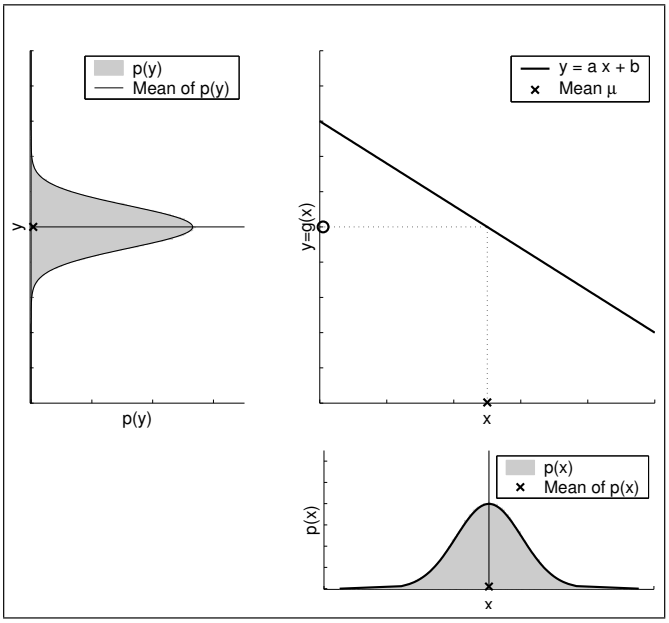

Fig. 3. Linear transformation of a Gaussian random variable [3]. The lower right plot show the density of the original random variable, X. This random variable is passed through the function displayed in the upper right graph (the transformation of the mean is indicated by the dotted line). The density of the resulting random variable Y is plotted in the upper left graph.

The algorithm represents the belief $bel(x_t)$ at time t by the mean μ_t and covariance Σ_t. The input is the belief at time $t-1$, the covariance at time $t-1$, the control u_t and measurement z_t. The output is the belief at time t.

In lines 2 and 3, the predicted belief $\overline{\mu}$ and $\overline{\Sigma}$ is calculated representing the belief $\overline{bel}(x_t)$ one time step later, but before incorporating the measurement z_t. This belief is obtained by incorporating the control u_t. The mean is updated using the deterministic state transition function 15 with the mean μ_{t-1} substituted for the state x_{t-1}. The update of the covariance considers the fact that states depend on the previous state through the linear matrix A_t. The variable K_t computed in line 4 is called *Kalman gain*. This is the reduction is uncertainty that is obtained by combining the prediction and correction steps. The belief $\overline{bel}(x_t)$ is transformed into the corrected belief $bel(x_t)$ in lines 5 and 6 by incorporating the measurement z_t. Line 5 manipulates the mean by adjusting it in proportion to the Kalman gain K_t and the deviation of the actual measurement z_t and the measurement predicted as per the measurement probability 16. Finally a new covariance of the posterior belief is calculated in line 6, adjusting for the information gain resulting from the measurement.

Figure 2 illustrates the Kalman filter for a simplistic one-dimensional localization scenario. Suppose the robot moves in the horizontal axis in each diagram in figure 2 with the prior over the robot localization given by the normal distribution shown in figure 2a. The robot queries its sensors on its location and these return a measurement that is centered at the peak of the bold Gaussian distribution in figure 2b. The peak corresponds to the value returned by the sensors and the its width or variance corresponds to the uncertainty in measurement. Combining the prior with the measurement in lines 4 through 6 of the Kalman filter algorithm, yields the bold Gaussian in figure 2c. The belief's mean lies between the two original means, and its uncertainty width is smaller than both contributing Gaussians. The is the result of the information gain from using the Kalman filter.

Now lets assume that the robot moves towards the right. Its uncertainty grows due to the fact that the state transition is stochastic. Lines 2 and 3 of the Kalman filter algorithm provide us with the Gaussian shown in bold in figure 2d. This Gaussian is shifted by the amount the robot moved and is also wider because of the uncertainty introduced by the movement. The robot does a second sensor measurement shown by the bold Gaussian in figure 2e, which leads to the posterior shown in bold in figure 2f.

As this example illustrates, the Kalman filter alternates a measurement update step, in which sensor data is integrated into the present belief, with a prediction step, which modifies the belief. The prediction step increases and the update step decreases the uncertainty in the robot's belief.

D. The Extended Kalman Filter

For a Kalman filter to work, all the transformations need to be linear. Figure 3 illustrates the linear transform of a one-dimensional Gaussian random variable. The graph on

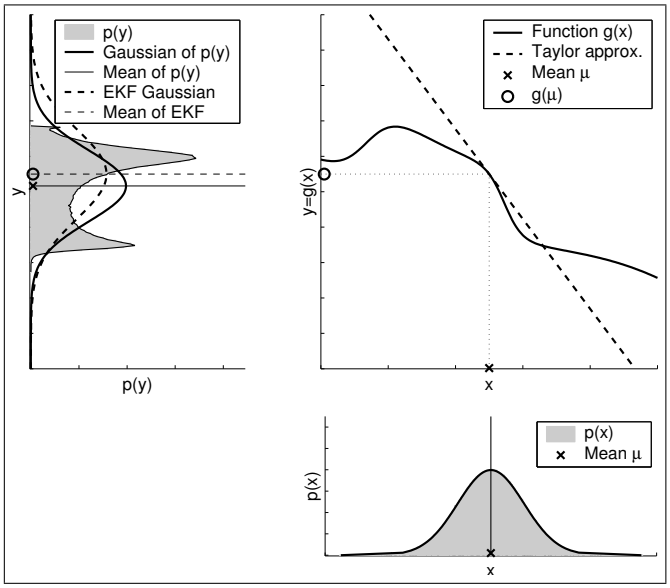

Fig. 4. Illustration of linearization applied by the EKF [3]. Instead of passing the Gaussian through the nonlinear function g, it is passed through a linear approximation of g. The linear function is tangent to g at the mean of the original Gaussian. The resulting Gaussian is shown as the dashed line in the upper left graph. The linearization incurs an approximation error, as indicated by the mismatch between the linearized Gaussian (dashed) and the Gaussian computed from the highly accurate Monte-Carlo estimate (solid).

the lower right shows the density of this random variable having mean μ and variance σ^2. The random variable is passed through a linear function $y = ax + b$ shown in the upper right graph. The resulting random variable will have Gaussian distribution with mean $a\mu + b$ and variance $a^2\sigma^2$. This resulting Gaussian is shown in the upper left graph of figure 3.

Unfortunately state transitions and measurements are rarely linear in practice [3]. This renders Kalman filters inapplicable to most robotics problems. The Extended Kalman filter, or EKF, relaxes one of the assumptions. Here the assumption is that state transition probabilities and landmark measurement probabilities are governed by nonlinear functions g and h respectively:

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \tag{17}$$

$$z_t = h(x_t) + \delta_t \tag{18}$$

Here x_t is the robot location at time t, u_t is the robot command that causes it to reach position x_t, z_t is the landmark measurement at time t, ϵ_t is the robot motion uncertainty that depends of the mechanical properties of the robot and δ_t is the sensor measurement uncertainty.

Figure 4 illustrates the impact of a nonlinear transformation on a Gaussian random variable. The graph on the upper right plots the nonlinear function g. As you can see, the transformed function is not a Gaussian. The idea behind the EKF is linearization. The nonlinear function is approximated

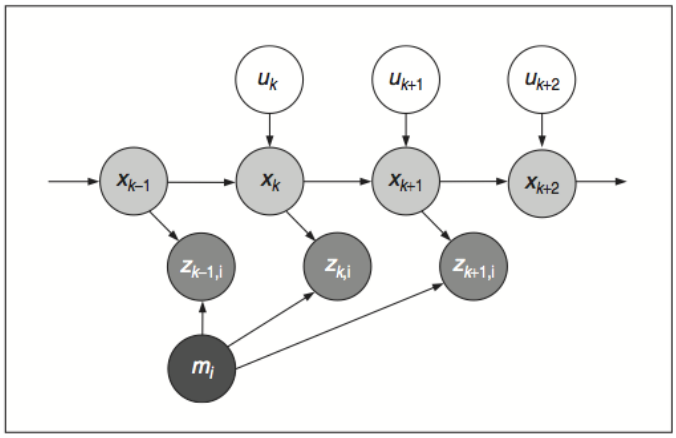

Fig. 5. A graphical model of the SLAM algorithm [3].

at mean of the Gaussian. Projecting this Gaussian through this linear approximation results in a Gaussian density as represented by the dotted line in the upper left of figure 4.

The Extended Kalman filter algorithm is given below [3]

Algorithm 3 Extended Kalman Filter Algorithm

1: **procedure** EXTENDED KALMAN FILTER ALGORITHM($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2: $\overline{\mu}_t = g(u_t, \mu_{t-1})$
3: $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4: $K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
5: $\mu_t = \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t))$
6: $\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$
7: return μ_t, Σ_t
8: **end procedure**

As can be seen, the EKF algorithm is similar to the Kalman filter algorithm in may ways. EKFs use Jacobians G_t and H_t instead of the corresponding linear system matrices A_t, B_t and C_t in Kalman filters. The Jacobian G_t corresponds to the matrices A_t and B_t, and the Jacobian H_t corresponds to C_t. The Jacobians are actually the linear approximations of the non-linear functions at the mean of the random variable that is being transformed.

E. EKF SLAM Explanation

A graphical model of the SLAM algorithm is shown in figure 5. The estimated mean of the robot pose (X coordinate, Y coordinate and robot orientation) is depicted by the circles containing the x with subscript. The circles with u and a subscript stand for the motion command at the location. The circle with m stands for a landmark and the circles with z and subscript stand for sensor measurements of the landmarks taken at the robot pose that the circle is attached to.

When the robot is at pose x_k it takes measurement $z_{k,i}$ of landmark m_i. It then issues a move command u_k and

moves to pose x_{k+1}. At x_{k+1} the robot computes a revised estimate of the mean of its pose by taking into account its previous pose estimate x_k and the motion command u_k. This is represented in line 2 of algorithm 3 for the Extended Kalman Filter, where g represents the non-linear transformation function for the mean value of the robot pose.

Since location uncertainty increases with each move, the robot computes its new location uncertainty by using the Jacobian of the G of the move transformation function g and the covariance matrix that represents its movement uncertainty. This movement uncertainty represents how much the robot actual location differs from its expected location after a move. It is a measure of the precision of the robot movement mechanism. This step is shown in line 3 of the EKF algorithm. Here $\overline{\Sigma}_t$ is the new location uncertainty, G_t is the Jacobian of the movement transformation function at pose x_t, Σ_{t-1} is the previous location uncertainty and R_t is the covariance matrix representing the precision of the robot movement mechanism. Lines 2 and 3 represent the prediction step of the EKF algorithm.

The robot saves the location and signature of each landmark it has seen in its pose [3] and when it sees a new landmark, it adds it to the pose, which is a state vector of the form

$$\begin{aligned} y_t &= \begin{pmatrix} x_t \\ y_t \end{pmatrix} \\ &= (x \ y \ \theta \ m_{1,x} \ m_{1,y} \ s_1 \ \ldots \ m_{N,x} \ m_{N,y} \ s_N)^T \end{aligned} \tag{19}$$

Here x, y and θ denote the robot's coordinates at time t (not to be confused with the state variables x_t and y_t), $m_{i,x}$ and $m_{i,y}$ are the coordinates of the i-th landmark, for $i = 1, \ldots, N$, and s_i is its signature.

At each time t, the robot uses its sensor to find the range and bearing to known landmarks (known x and y landmark coordinates). Each such sensor reading is represented by z_t. At each robot location, it can compute the expected sensor reading using a non-linear function h, where $h(\overline{\mu}_t)$ returns the expected sensor reading at the location based on one of the known landmarks. Since the robot has already computed its expected location $\overline{\mu}_t$ and it knows the landmark coordinates, it can easily compute the expected sensor reading using trigonometry. This is the non-linear function h.

Line 4 of the EKF algorithm shows the computation of the Kalman Gain K_t. H_t represents the Jacobian of the non-linear function h and Q_t is a covariance matrix that represents the uncertainty in the sensor reading. Q_t is of the form

$$Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix} \tag{20}$$

where r denotes the range to a landmark, ϕ is the bearing to a landmark and s the landmark signature. The terms σ_r, σ_ϕ and σ_s are the corresponding covariances.

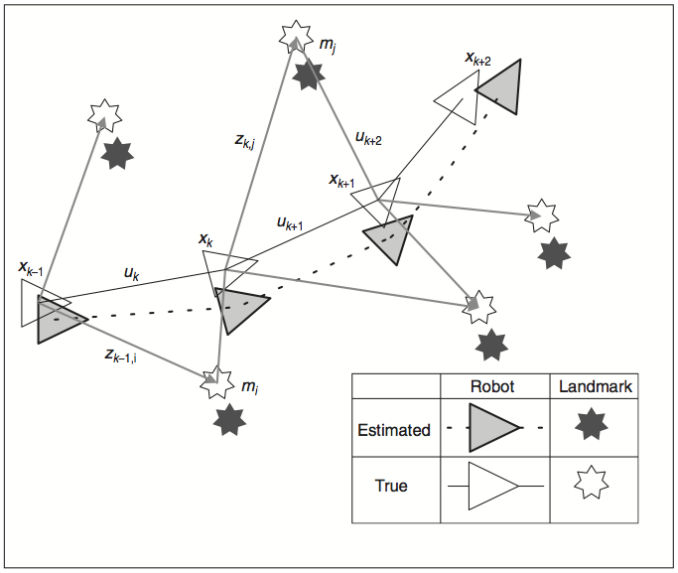

Fig. 6. The essential SLAM problem [2]. A simultaneous estimate of both robot and landmark locations is done. The true locations are never known or measured directly. Observations are made between true robot and landmark locations.

In line 4, when the sensor measurement uncertainty is high, the Kalman Gain is low and vice versa. This Kalman Gain is used in line 5, to compute a better estimate of the robot expected mean location. The Kalman Gain is again used in line 6 to compute a better estimate of the uncertainty, thereby reducing the location uncertainty.

The newly computed location and uncertainty, μ_t and Σ_t are returned in line 7 of the algorithm.

Figure 6 shows EKF SLAM in operation. The robot moves through the environment and simultaneously estimates both its own location and those of the landmarks. Due to uncertainties in robot movement and sensor measurements, the true locations are never known, but an estimate consisting of a mean value and covariance is computed. The estimated mean value may be different from the true one, but all the estimates are correlated with each other. As the robot moves through the environment, and observes the same landmark multiple times, the correlations between landmarks increases. Landmarks that the robot can observe together, end up being more correlated with each other. This is shown in figure 7 where the increased correlation is shown by thicker red springs. All landmarks locations are correlated with each other and the robot, with the landmarks that can be observed at the same time being more strongly correlated and represented by thicker springs.

Figure 8 shows a robot that is sensing three obstacles shown in gray in figure 8(a). Figure 8(b) shows the likelihood of

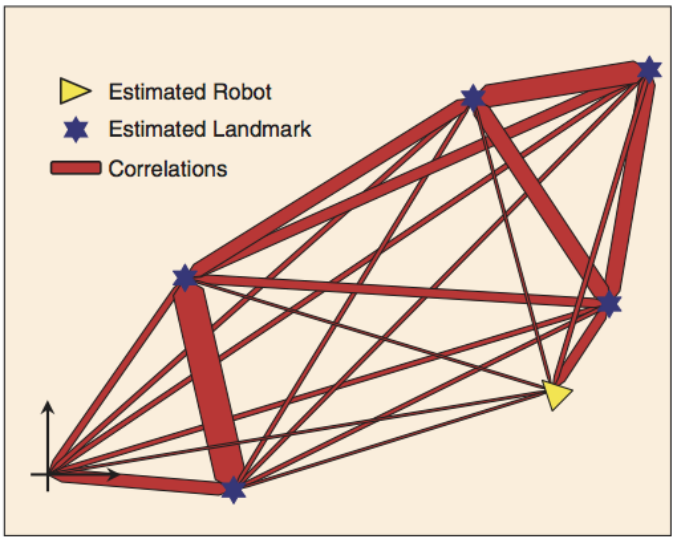

Fig. 7. Spring network analogy [2]. The landmarks are connected by springs describing correlations between landmarks. As the vehicle moves back and forth through the environment, spring stiffness or correlations increase (red links become thicker). As landmarks are observed and estimated locations are corrected, these changes are propagated through the spring network. Note, the robot itself is correlated to the map.

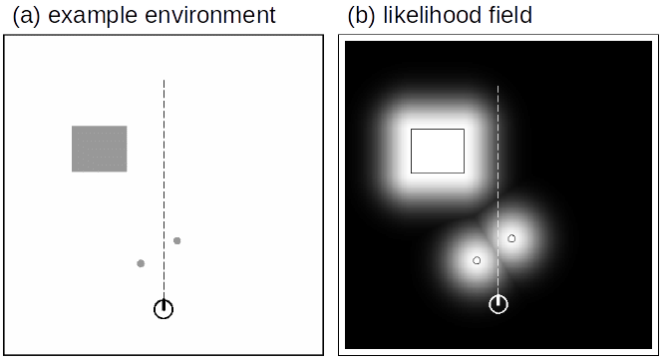

Fig. 8. (a) Example environment with three obstacles in gray. The robot is located towards the bottom of the figure and takes a measurement z_t^k as indicated by the dashed line. (b) Likelihood field for this obstacle configuration. The darker the location, the less likely is the robot to perceive an obstacle there. [3]

the robot sensing the obstacles. The darker the location, the less likely is the robot to perceive an obstruction there. The sharp definitions of the actual obstacles are turned into blurred images that represent the uncertainty in the measurement of the obstacle coordinates and the signature. The higher the Kalman Gain, the less blurred the perceived obstacles will be.

IV. Results

The robot manufacturer provides a Python library with movement control and rangefinder functions. The movement control functions provide for precision movement using encoders disks that are attached to the wheel axle. The robot wheels can be independently made to go forward or backward a specific number of encoder steps. The rangefinder can be pointed at any angle in front of the robot and will return the distance in centimeters to an obstacle that the sensor is pointing to. The python NumPy library was for matrix and array operations.

The robot can be moved forward, backwards, clockwise or anti-clockwise in place through use of Python library functions. For the movement to have a degree of precision, it is done under encoder control. Here the number of encoder steps is computed for the required forward or backward movement distance or in place rotation angle. This computed encoder target is sent to the robot encoders and the movement command issued. The program needs to wait till the target encoder value is reached.

The ultrasonic rangefinder can be made to point to a specified angle in front of the robot. It can then be queried to return the range to the obstacle in front. The bearing to the obstacle is determined by the angle that the rangefinder is made to point to.

The robot was calibrated by giving move and rotation commands and then finding out the movement error by measuring the actual movement with a tape measure and finding the difference with the expected movement. This was repeated thirty times each for forward movement and rotation in place. These are the only two move commands that were implemented. The recorded error values for x and y coordinates and the robot orientation θ were used for computing the movement uncertainty covariance matrix R_t using the Python function $numpy.ma.cov()$.

The obstacle sensing uncertainty was measured in a similar manner by finding the difference between the range and bearing returned by the sensor to actual values that were measured using a tape measure and digital protractor. Fifteen error readings were taken in all for the rangefinder uncertainty covariance matrix Q_t.

The robot has an outer navigation control loop where it scans for obstacles and changes orientation in steps till it finds an obstacle. It then moves towards the nearest obstacle and stops when it reaches within 2.5 robot lengths. Then it turns 45° left or right based on a random selection and repeats the process.

The robot movement equations represented by function g mentioned in line 2 of the EKF algorithm are given in section VII-B. These equations are for the case of the robot moving forward by a specified amount and for the robot rotating in place by a specified angle. No other robot motion has been implemented and the robot does not move back or move in an arc. The movement equations are used for finding the predicted mean location $\overline{\mu}_t$ after a move command as described in line 2 of the EKF algorithm.

The movement Jacobian can be found by deriving the

parting derivative of the motion equation with respect to each component of the robot pose. This derivation is given in section VII-C. This Jacobian can now be used in line 3 of the EKF algorithm.

The ultrasonic rangefinder on the robot did not perform well and reported the size of an obstacle as 4.8 times of what it actually was. This was probably a combination of the low grade of the sensor and the fact that ultrasonic rangefinders are not known to be accurate. As a result of the obstacles being reported as almost five times larger than what they actually were, and because of the need to space the obstacles apart by at least this width plus the sensing uncertainty times two, the testing of the robot did not progress beyond the location prediction stage. This was due to available space limitations of where the robot could be tested. In addition to this, the process of associating new observations to previously seen landmarks became impossible with the nearest neighbor method. The reason is that the nearest neighbor association method relies on closeness of the previously observed landmarks to their new expected location based on the robot movement. Since the sensing uncertainty is so large, it made the association with known landmarks impossible. Due to this, the correction step could not be implemented.

The robot is able to move around and navigate through an obstacle course. However, as described above in the previous section, the prediction step of the EKF SLAM process was not implemented.

The project has not been a total loss though. I have a very good understanding of the EKF SLAM process and had to get re-acquainted with probability theory, conditional probability, Bayes Rule, multivariate normal distributions, covariance and covariance matrices, Kalman and Extended Kalman Filters and the EKF SLAM process as a whole. This was a non-trivial topic to grasp and needed a lot of reading and understanding.

V. Discussion

A very important concept in EKF SLAM is the use of a Kalman or Extended Kalman Filter. These filters are used in places like the Google car, rockets, missiles, tracking of objects and precise movements of robotic arms. Kalman filters can even be used for estimating the charge in a rechargeable battery.

A Kalman filter can essentially work in an environment where the data contains added noise. It can attenuate the noise and output data with less noise. As described earlier, a Kalman filter uses a model of the system to compute its predicted state along with the associated uncertainty in the state. Then it will use an independent mechanism to measure the state. Using the predicted uncertainty in the state and the expected noise in the measurement process, it will compute the Kalman Gain. The lower the predicted uncertainty in the state and the expected noise in the measurement process, the higher will be the Kalman Gain. The Kalman Gain is then used to reduce the uncertainty in the system state and apply a correction to the mean or expected system state.

VI. Conclusions and Future work

The accuracy of the rangefinder was the main obstacle encountered in this project. A better sensor can probably be used to obtain a more accurate obstacle signature and continue with the project. The Xbox Kinect sensor has a camera and rangefinder built into one sensor and promises more accuracy in measurements. A Kinect would be the next step in continuing with this project.

References

[1] GoPiGo - Dexter Industries. *Dexter Industries*. N.p., n.d. Web. 06 Nov. 2015. http://www.dexterindustries.com/gopigo/.

[2] Durrant-Whyte, H., and T. Bailey, *Simultaneous Localization and Mapping: Part I*, IEEE Robotics and Automation Magazine. 13.2 (2006): 99-110.

[3] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*, MIT press, 2005.

[4] Stachniss, Cyrill, *SLAM Course - WS13/14*, YouTube. YouTube, n.d. Web. 18 Sept. 2015. https://www.youtube.com/playlist?list=PLgnQpQtFTOGQrZ4O5QzbIHgl3b1JHimN_.

[5] Maybeck, Peter S., and George M. Siouris. "Stochastic models, estimation, and control, volume i." IEEE Transactions on Systems, Man, and Cybernetics 5.10 (1980): 282.

VII. Supplementary Materials

A. Combining two Gaussian distributions

Figure 9 shows two Gaussian distributions for the position of a robot in dotted lines with means z_1 and z_2 and standard deviations σ_{z_1} and σ_{z_2} respectively. The resulting distribution obtained by combining these two distributions is shown in the solid line with mean μ and standard deviation σ.

The mean for the resulting distribution is given by [5]

$$\mu = (\frac{\sigma_{z_2}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2})z_1 + (\frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2})z_2 \tag{21}$$

and the variance for the resulting distribution is given by

$$\frac{1}{\sigma^2} = \frac{1}{\sigma_{z_1}^2} + \frac{1}{\sigma_{z_2}^2} \tag{22}$$

If the uncertainties σ_{z_1} and σ_{z_2} of the two distributions are equal, based on equation 21 the expected mean of the resulting distribution (the expected robot position) is just the average of the two or in other words, a position exactly between the two locations σ_{z_1} and σ_{z_2} [5].

On the other hand if σ_{z_1} were larger than σ_{z_2}, which is to say that the uncertainty involved in the measurement of z_1 is greater than that of z_2, then equation 21 dictates weighing z_2 more heavily than z_1. Which means the expected robot position will be closer to z_2. This is the scenario shown in figure 9.

As per equation 22, the uncertainty of the resulting distribution is less than σ_{z_2} even if σ_{z_1} is very large. That is to say that

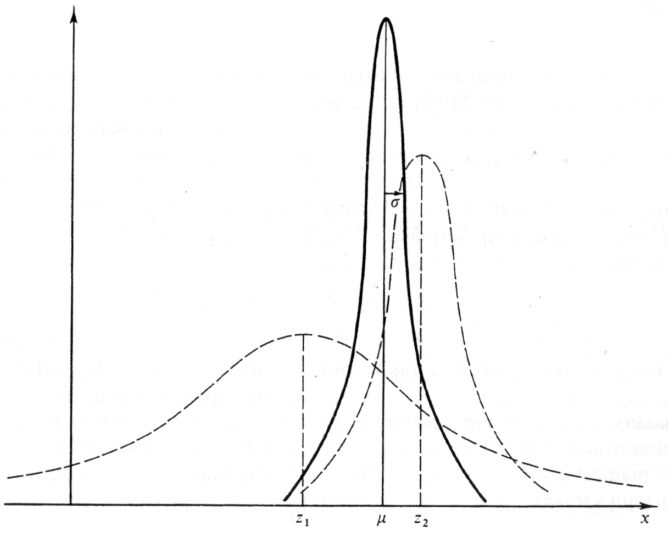

Fig. 9. Conditional density of position based on data z_1 and z_2 [5]. x-axis=position, y-axis-probability of being at the position

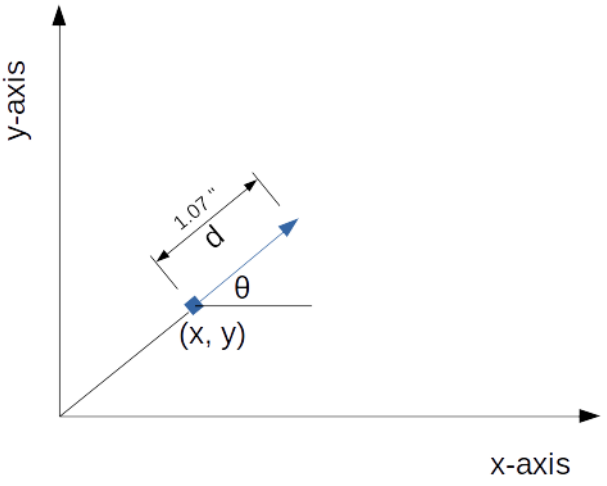

Fig. 10. Robot initial pose at coordinate (x, y) and at an orientation of θ. The robot pose is at the base of the blue arrow. The robot then moves a distance d in the direction it is pointing to.

the uncertainty of the resulting distribution is less than either of the two uncertainties. This may seem counter-intuitive, but even if very poor quality data is taken into consideration, it does provide some information and thus will increase the precision of a measurement.

B. Robot Motion Equation

The robot pose is represented by the vector $(x\ y\ \theta)^T$, where x and y are its location coordinates and θ is its orientation. The robot pose can be considered to be $(0\ 0\ 0)^T$ when it starts moving since the coordinate system does not have a specific frame of reference.

Let us consider the case when the robot starts at pose $(x\ y\ \theta)^T$ and moves forward a distance d in the direction it is pointing to. This is illustrated in figure 10. As per simple trigonometry, the equation for the new robot pose will be

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} d\cos(\theta) \\ d\sin(\theta) \\ 0 \end{pmatrix} \tag{23}$$

This is nothing but the definition of function g mentioned in line 2 of the EKF algorithm.

The robot can also rotate in place and in this case, the only thing that changes in the pose is the orientation. The function g in this case is defined by

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \theta' \end{pmatrix} \tag{24}$$

where θ' is the angle the robot is made to rotate by.

C. Robot Movement Jacobian

The movement Jacobian represented by G_t in line 3 of the EKF algorithm can be obtained by finding the partial derivative of the movement function g [3]. In the case of the move forward, the Jacobian is given by

$$\begin{aligned} G_t &= I + \frac{\partial}{\partial(x\ y\ \theta)^T} \begin{pmatrix} d\cos(\theta) \\ d\sin(\theta) \\ 0 \end{pmatrix} \\ &= I + \begin{pmatrix} 0 & 0 & -d\sin(\theta) \\ 0 & 0 & d\cos(\theta) \\ 0 & 0 & 0 \end{pmatrix} \end{aligned} \tag{25}$$

where I is the identity matrix. For the rotate in place operation, the Jacobian is given by

$$\begin{aligned} G_t &= I + \frac{\partial}{\partial(x\ y\ \theta)^T} \begin{pmatrix} 0 \\ 0 \\ \theta' \end{pmatrix} \\ &= I + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ &= I \end{aligned} \tag{26}$$