Simultaneous Location and Mapping

TJ Bollerman

April 30, 2015

Contents

1 Introduction 1

1.1 What is Simultaneous location and mapping . . . 1

1.2 The SLAM Process . . . 2

1.3 Probabilistic Formulation . . . 2

2 SLAM Sub-problems and Solutions 2

2.1 Landmarks . . . 2

2.2 Data Association . . . 4

2.3 The Extended Kalman Filter . . . 4

3 My Simulation 6

3.1 Model . . . 6

3.2 Snapshots . . . 8

4 State-of-the-Art 8

5 Appendix 9

5.1 Code . . . 9

1 Introduction

1.1 What is Simultaneous location and mapping

Simultaneous location and mapping (SLAM) deals with the problem of estimating the state of a robot, while concurrently building map of the unknown environment in real time. SLAM is a fundamental problem in robotics; in fact Hugh Durrant-Whyte called SLAM the holy grail of robotics [3] and it is easy to see why. The SLAM problem and solution is ubiquitous in robots, implementations are seen in planetary rovers, self-driving cars, UAVs, autonomous vacuum cleaners and many other systems; even you do it every day. Any time a robot enters into an uncertain area, it must solve the SLAM problem. Even when an area is known, environments are dynamic by nature and simple localization is often not robust enough and SLAM algorithms are used instead. SLAM is really more of a concept than an algorithm as there are many sub-problems that can be solved using many different algorithms. However, theoretically, SLAM is solved problem but it is still a very active research topic including loop closer, large scale SLAM, 3-D SLAM, and various data association and other algorithm improvements and generalizations. In this report, a simple SLAM simulation is written in python using the Extended Kalman Filter.

1.2 The SLAM Process

The robot starts in an initial position in an unknown environment. The robot moves according to control input, the system dynamics, and a zero mean Gaussian process noise. As the robot moves, the uncertainty in its position grows quickly due to the process noise. However, the robot can gain information on its current state by measuring the angle and distance to various landmarks and comparing the measurements to what we expected to measure given the history of the previous measurements. Landmarks are elements of environment that the robot can measure effectively. While the robot moves it measures the angle and distance to each landmark with a zero mean Gaussian error. The difference between what the robot measures and you would expect to measure given the controls is called the innovation. The expected state is then corrected by combining the innovation and the expected state due to the control input according to the Kalman gain. Using the Extended Kalman Filter, it can be shown that the individual landmark and robot position variances converge toward a lower bound determined by the initial uncertainties in robot position and observation [3]. The Extended Kalman Filter process will be covered in section 2.3.

1.3 Probabilistic Formulation

The formulation of SLAM is inherently probabilistic. In fact, there are two formulations of the SLAM problem; the full and online SLAM. Let x_t be the robot location and bearing at time t, m be the map, and $z_{1:t}$, $u_{t:1}$ be the history of measurements and control from time 1 to t respectively.

The solution to the full SLAM formulation requires finding the probability distribution of

$$P(x_{1:t}, m | z_{1:t}, u_{1:t}) \tag{1}$$

for all times t. The full SLAM problem cannot be solved in real time and is infeasible in many applications. The more common online SLAM requires finding the probability distribution of

$$P(x_t, m | z_{1:t}, y_{1:t}) \tag{2}$$

for all times t. The online SLAM is derived from the full SLAM by integrating out past robot locations and bearings. The online problem is much more common due to the fact it can be solved in real time using many different algorithms. [5]

2 SLAM Sub-problems and Solutions

2.1 Landmarks

Landmarks play an integral role in the solution of SLAM. The robot uses the repeated measurements of its bearing and distance to landmarks to correct and decrease the uncertainty in its position. It is thus clear that in order to be effective landmarks should be re-observable, easily distinguishable, plentiful, and stationary. [4]

The LIDAR is the most commonly used range measurement device used today. LIDARs are very precise, efficient and the output does not require much computation to process. However, LIDARs are expensive and cannot be used underwater or measure distances to glass. [4]

Using computer vision for SLAM is a very active research topic and considerable advancements have been made. Using vision allows the robot to gain a lot more information of the environment and mimics the way humans approach the SLAM problem and thus it is far better for human-robot team applications. Vision is computationally expensive, but as computational power of processors increases, this is becoming less of a problem. [4]

In the past SONARs were widely used as a range measurement device in robotic. SONARs beams can be as large as 30 degrees providing considerable challenges in accurately measuring the bearing. SONARs are also much nosier than LIDAR measurements and often give bad readings.

After the range measurement device is selected, one must choose a way to extract the landmark location from the raw data. One commonly used method of landmark extraction is the ‘spike method’. As the LIDAR scans an area, a landmark may be found by considering the derivative of the range as a function of the angle.

A sharp decline in the range follow by a sharp increase in range implies a landmark is at that location. Figure 1 from Claus Brenner's SLAM lectures [2] shows an example of the spike method. It is important to note that often the LIDAR or other ranging instrument will give false readings, thus it is often necessary to define a cutoff value such that any readings with a slope greater than the cutoff value is ignored. This too can be seen in figure 1. The spike method is reliant on there being many well defined discrete landmarks instead of smooth surfaces.

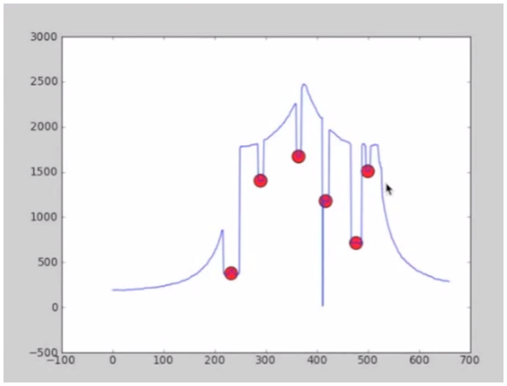

Figure 1: The spike method is demonstrated.

RANSAC stands for Random Sampling Consensus. RANSAC is used to extract straight lines from the raw data. RANSAC is more robust than the spike technique and is better suited for practical SLAM applications where well-defined landmarks are rare or non-existent and instead walls and corners are plentiful. RANSAC samples the range measurements and uses a line of best fit to extract lines from those measurements. From that line, it then breaks the line into N equal parts and uses those points (and often the corners) as landmarks. Since RANSAC is a statistical landmark extraction technique it is robust to dynamic environmental factors such as people walking in a room, which makes the RANSAC method very desirable. Figure 2 depicts the RANSAC method

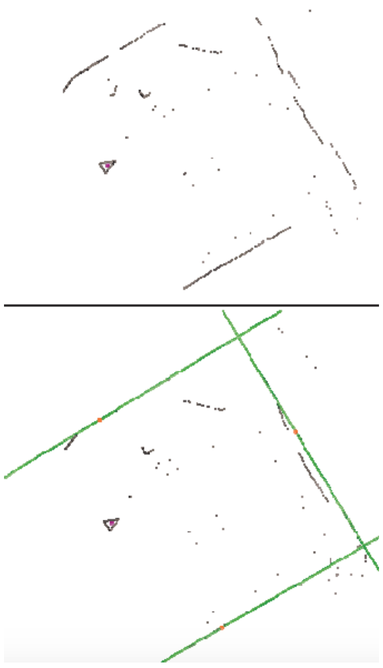
Figure 2: The RANSAC method is demonstrated. The above graph shows the raw scan data, and the bottom graph shows the lines of best fit drawn by the RANSAC algorithm.

2.2 Data Association

In order to correctly use the Extended Kalman Filter it is essential to know if the landmark measured in time t is the same as which landmark in time $t-1$, or if it is being viewed for the first time. It is the responsibility of data association algorithms to correctly match the observe landmarks with the previously observed landmarks or conclude it is being viewed for the first time. Incorrectly associating landmarks can be devastating in many SLAM algorithms, especially the Extended Kalman Filter as it provides incorrect information causing the robot to make incorrect adjustments in its estimated position. [3]

There are many data association algorithms; in fact data association could be call a field of research on its own. A very simple and common method of data association is the nearest neighbor algorithm. The nearest neighbor algorithm can be summarized as follows: After extracting landmarks in time t, for each observation if the landmark is closer than λ away from where we would expect a previous landmark measured in $t-1$ would be given a control u_t then the landmarks are associated. Otherwise, we assume the new landmark has not been seen yet and the new landmark is added to the state. While the nearest neighbor approach is simple and easy to implement, it is easy to see wrong associations, or incorrectly added a non-existent landmark are possible, if not frequent [4] [2]. In practice, more complex data association algorithms are used, but they are left out of this summary for the sake of brevity.

2.3 The Extended Kalman Filter

As described in the probabilistic formulation of SLAM the goal is to describe the probability density function

$$P(x_t, m|z_{0:t}, u_{0:t}, x_0) \tag{3}$$

Starting with an estimate of the probability of the previous time period (initial position is usually assumed to be known exactly), Bayes theorem is then used to estimate the current probability distribution. The prediction step can be defined as

$$P(x_t, m|z_{0:t-1}, u_{0:t}, x_0) = \int P(x_t|x_{t-1}, u_k) \times P(x_t, m|z_{0:t-1}, u_{0:t-1}, x_0) dx_{t-1} \tag{4}$$

and the correction step is defined as

$$P(x_t, m|z_{0:t}, u_{0:t}, x_0) = \frac{P(z_t|x_t, m)P(x_t, m|z_{0:t-1}, u_{0:t})}{P(z_t|z_{0:t-1}, u_{0:t})} \tag{5}$$

Taking

$$P(x_t, m|z_{0:t-1}, u_{0:t-1}, x_0) \Longleftrightarrow x_t = g(x_{t-1}, u_t) + w_t \tag{6}$$

and

$$P(z_t|x_t, m) \Longleftrightarrow h(x_t, m) + v_t \tag{7}$$

where v_t and w_k are random Gaussian vectors with zero mean. The Bayes Filter can be casted into the more practical Extended Kalman Filter (EKF). [5] Similarly to the Bayes theorem formulation the EKF has two main steps the prediction step and the correction step. It is important to note however that landmark extraction, adding new landmarks and data association must also take place. Since the probability density function is assumed to be Gaussian, a mean and covariance matrix can uniquely define the function. As both the position and bearing of the robot as well as the map is being calculated the state is defined as

$$X_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \\ x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix} \tag{8}$$

Where x_t, y_t and θ_t are the x, y, and bearing of the robot at time t and x_j, y_j is the location of landmark j. It is important to recognize that the function g is the movement function with no error, and the function h is the measurement function with no error. The function g may be appended with zeroes (twice the number of landmarks) to conform with the size of the state.

Let $\bar{\mu}_t$ and $\bar{\Sigma}_t$ be the expected value and covariance matrix of the state X_t only considering the input u_t. The prediction step of the Extended Kalman Filter is as follows

$$\bar{\mu}_t = g(\mu_{t-1}, u_t) \tag{9}$$

$$\bar{\Sigma}_t = G_{t-1}\Sigma_{t-1}G_t^T + V_t\Sigma_{pr}V_t^T \tag{10}$$

The prediction step for $\bar{\mu}_t$ is very appealing intuitively. Since the noise is a zero mean Gaussian random vector, the best guess for the error is zero. Thus the prediction step for $\bar{\mu}_t$ is simply the movement function applied to the previous μ_{t-1}.

The prediction step for the covariance matrix $\bar{\Sigma}_t$ is harder to analyze intuitively but it is clear that the covariance matrix will tend to infinity if the landmark measurements are not used. The covariance estimation update as defined in equation [10] is derived by the linearization of the movement function g. The matrix G_t is the Jacobian of the movement function with respect to the state at time t and the matrix V_t is the Jacobian of the movement function with respect to the control at time t. The process noise is represented by the matrix Σ_{pr}. The process noise is assumed to be a diagonal covariance matrix as there is no correlation between the process noise in different directions. [2]

After the prediction step takes place, it is necessary for the robot to scan the environment, extract the landmarks and associate the landmarks to previous scans. Each one of these steps may be carried out using any of the algorithms described in this summary, or more advanced state-of-the-art algorithms left out for the sake of brevity. After the landmark extraction and association takes place, the robot can use this information, in the correction step of the Extended Kalman Filter, such that the covariance matrix will converge to zero (iff the initial covariance is zero) and the expected state will approach the true state as time goes to infinity. [3]

The correction step of the Extended Kalman Filter is as follows. It is important to note that the prediction step is to be follow for each observation seen in time t.

$$\mu_t = \bar{\mu}_t + K_t(z - h(\bar{\mu}_t)) \tag{11}$$

$$\Sigma = (I - K_t H_t)\bar{\Sigma}_t \tag{12}$$

where the matrix K_t, the Kalman gain is defined as

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q)^{-1} \tag{13}$$

The matrix H_t is defined as the Jacobian of the measurement function h with respect to the state at time t. The matrix Q is the covariance of the measurement error. Like the matrix Σ_{pr}, Q is assumed to be diagonal as there is no correlation between the measurement errors in different directions. As previously defined, z represents the distance and angle to a landmark measured by the robot, while the function h represents what we should expect to measure given the position of the robot. Thus the term $z - h(\bar{\mu}_t)$ may be interpreted as the difference in what we measure to what we expected to measure given only the control. This term is called the innovation. [2] [3]

The Kalman gain may be thought of as the optimal way to average the expected state given the controls and the state which the measurements imply. This interpretation can be qualitatively seen by studying the effect the matrix Q has on the Kalman gain. Using this knowledge the correction step of the expected state of the robot and map is clear. The corrected expected state μ_t is simply an optimal average of the information gained by the controls, $\bar{\mu}_t$, and the information gained by the measurements, the innovation.

The robot then moves again and the process is repeated. The Extended Kalman Filter thus provides an easy and intuitive way to solve the SLAM problem. However, we must consider the performance of the EKF. The EKF is convergent in that the covariance matrix tends towards a lower bound determined by the initial uncertainties in the robot position and uncertainties [3]. This makes sense intuitively because as the robot moves the correlation between the landmarks become very highly correlated. This correlation can be thought as a network of springs, who's spring coefficient is analogous to the correlation between the two landmark positions. Thus if the robot is initially unsure of where it is, the whole landmark structure can move according to the bound of the initial uncertainty [2]. The computation time of the Extended Kalman Filter grows quadratically with the number of landmarks. Additionally the Extended Kalman Filter is very sensitive to incorrect associations, an incorrect association can cause a large bias in the solution. As the EKF employs linearizing the state via Jacobians, it is also sensitive to very non-linear motion. [3]

In the next section, I will show my model and how I used the EKF to solve the SLAM problem.

3 My Simulation

3.1 Model

To practice my programming ability and fully understand the heart of SLAM and the EKF solution, I implement a simple simulation using python. Landmarks are implemented as discrete custom python objects which inherit sprite objects from the pyglet module. The landmark positions are an ordered list in python and thus landmark extraction and data association are trivial. Additionally, using the order list of landmark locations implies the range measurement device has 360^o view and experiences no shadow effects. For future work on this simulation I would implement a 'real' sweeping range measurement device that would require implementation of landmark extraction and data association algorithms.

The robot, represented by a blue triangle, has a position x_t, y_t and bearing θ_t and moves non-holonomically according to the movement function g plus some process noise w. These are defined as

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + v_t \cos(\theta_t)\Delta t \\ y_{t-1} + v_t \sin(\theta_t)\Delta t \\ \theta_{t-1} + \omega_t \Delta t \end{bmatrix} + w_t \tag{14}$$

where v_t is either zero or a constant value based on the input on the keyboard and may be interpreted as the linear velocity of the robot. Pressing the up-key signals v_t to be a non-zero constant value set by the user, while not pressing the up-key signals v_t to be zero. Similarly ω_t is the rotational velocity, which is zero if no key is pressed, or a positive non-zero constant set by the user if the right key is pressed or the same constant negated if the left key is pressed. The process noise w_t is a zero-mean Gaussian vector with variance σ_l^2 for the first two elements (linear motion noise variance) in w and variance σ_b^2 for the last element (rotational noise variance). There is assumed to be no correlation between the linear motion noise and the rotational motion noise. The user sets these variances.

The distance r and angle α from the robot to each landmark are found by using the measurement function h plus the observation noise v, defined as

$$\begin{bmatrix} r \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{(x_l - x_r)^2 - (y_l - y_r)^2} \\ \tan^{-1}\left(\frac{y_l - y_r}{x_l - x_r}\right) - \theta_r \end{bmatrix} + v_t \tag{15}$$

where x_m, y_m are the x and y position of the landmark being measured and x_r, y_r are the x and y position of the robot. The observation noise is a zero-mean Gaussian vector with variance set by the user and no correlation.

With the movement and measurement functions well defined, we can now take the Jacobians to find the necessary matrices to use the EKF as described in the previous section. The easiest matrices to find are Q and Σ_{pr} which are defined as the matrices of the observation and movement noise variance.

$$Q = \begin{bmatrix} \sigma_o^2 & 0 \\ 0 & \sigma_o^2 \end{bmatrix} \tag{16}$$

$$\Sigma_{pr} = \begin{bmatrix} \sigma_l^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix} \tag{17}$$

When taking the Jacobian of the measurement function h it is convenient to define $q = (x_l - x_r)^2 - (y_l - y_r)^2$. The matrix H is found to be

$$H = \begin{bmatrix} -\frac{\Delta x}{\sqrt{q}} & -\frac{\Delta y}{\sqrt{q}} & 0 & 0 & \dots & 0 & \frac{\Delta x}{\sqrt{q}} & \frac{\Delta y}{\sqrt{q}} & 0 & \dots & 0 \\ \frac{\Delta y}{q} & \frac{\Delta x}{q} & -1 & 0 & \dots & 0 & -\frac{\Delta y}{q} & -\frac{\Delta x}{q} & 0 & \dots & 0 \end{bmatrix} \tag{18}$$

where the 2×2 sub-matrix starts at column $2 + 2i$ for the ith landmark in the state.

The Jacobian of the movement function g with respect to the state at time t is

$$G_t = \begin{bmatrix} 1 & 0 & -v_t\sin(\theta_t)\Delta t & 0 & 0 & \dots & 0 \\ 0 & 1 & -v_t\sin(\theta_t)\Delta t & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix} \tag{19}$$

and the Jacobian of the movement function g with respect to the control at t is

$$V_t = \begin{bmatrix} v\cos(\theta_t)\Delta t & 0 \\ vsin(\theta_t)\Delta t & 0 \\ 0 & \omega_t\Delta t \end{bmatrix} \tag{20}$$

With all the SLAM matrices defined we can carry out the Extended Kalman Filter as described in the previous section. In my simulation, I create a model robot colored gray. As the user controls the real robot, the model robot shows the estimated location of the real robot as given by the EKF. Similarly, the green circles represent the estimated location of the landmarks.

3.2 Snapshots

Here are some snapshots of the simulation that show the simulation is functional.

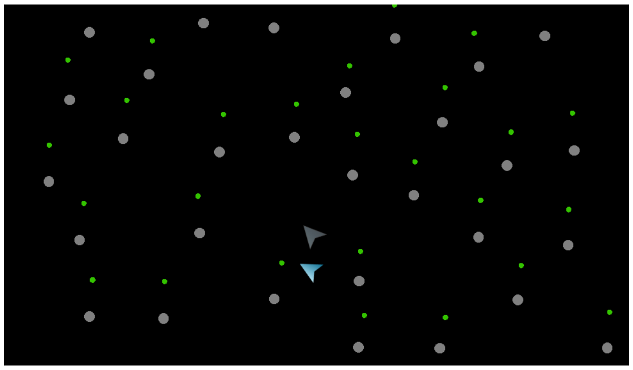

Figure 3: Here the Kalman gain is tuned to zero to show the effect of noise has on the difference between the real robot and the estimated location based only on the control. As expected, the real robot and the estimated location quickly diverge. Additionally, one can see the estimated location of the measured landmarks are relative to the estimated location of the model robot and are thus far from the actual location of the landmarks

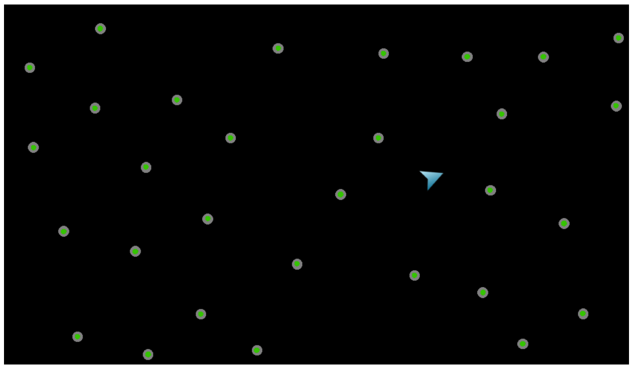

Figure 4: Turning the EKF on, the estimated robot location tracks the real robot well. The map is also well defined as the estimated location of each landmark converges to the real location. Thus the robot simultaneously builds a map and locates itself within that map. The SLAM problem has thus been solved and our simulation is successful.

4 State-of-the-Art

While SLAM is theoretically solved, it is still a very active topic of research as SLAM is naturally an applied concept and there are many real-world challenges that need to be solved. As described in this summary, there are many sub-problems in SLAM and each sub-problem has a number of active research topics in it. Unfortunately, we do not have time to go over any of these state-of-the-art algorithms in detail, instead a few of the concepts are very quickly introduced.

Computation time of the Extended Kalman filter, as previously stated, scales quadratically with the number of landmarks. One very active branch of research is speeding up the SLAM process. Sparsification is one such algorithm. Instead of representing the state and covariance matrix, it is possible to instead use the information vector and the information matrix. For large scale maps, the off diagonal terms of the normalized information matrix are near zero, allowing for use of sparse matrix functions which greatly speed computation time.[1]

Data association is extremely important in SLAM and surprisingly difficult to implement well. The simple nearest neighbor approach fails for all but the simplest of cases. Batch validation considers multiple associations simultaneously. By considering multiple associations at the same time the geometric relationship between the landmarks can be used to our advantage. A graph can be built from the possible landmarks and very effective and fast tree search algorithms can be used in order to reliably association landmarks. Another very active area of research is called 'closing the loop'. After a long journey, the robot might come back to re-observe an area. Closing the loop means to be able to recognize that the robot has been to that location before and use that information to our advantage. [1]

While we have assumed we can measure both the range and bearing to the landmarks, an active area of research is solving SLAM with very limited measuring capabilities. SLAM is still solve-able even with measurement devices that can only measure range or bearing. In fact a current area of research today is the blind robot problem. A robot is given only a contact detection device, can a robot simultaneously locate and map itself?

Maybe the most active and hopeful area of research in SLAM is scaling up. Hugh Durrant-Whyte, one of the researches who laid the foundation of SLAM, wrote " The challenge now is to demonstrate SLAM solutions to large problems where robotics can truly contribute: driving hundreds of kilometers under a forest canopy or mapping a whole city without recourse to GPS, and to demonstrate true autonomous localization and mapping of structures such as the Barrier Reef or the surface of Mars. SLAM has brought these possibilities closer." [1]

5 Appendix

5.1 Code

The most important two files of python code are included here. The first file is the main game loop followed by the model EKF robot class.

Main loop:

```python
# Global variable to keep track of what update number we are on (dont want to print every update)
update_number = 0

# Set up a window
sim_window = pyglet.window.Window(1010, 600)

# Set up the batch
main_batch = pyglet.graphics.Batch()
background = pyglet.graphics.OrderedGroup(0)
foreground = pyglet.graphics.OrderedGroup(1)

# Initialize the perfect model robot, scale the image down
perfect_robot = perfectrobot.Perfect_Robot(x=400, y=300, batch = main_batch)
perfect_robot.scale = .75

# Initialize the real robot, scale the image down
real_robot = realrobot.Real_Robot(x=400, y=300, batch = main_batch)
real_robot.scale = .75

# Pick which landmark set up you want, grid numer of landmarks must be a perfect square
measured_landmarks, landmarks = landmark.random_landmarks(30, main_batch, [foreground,background])
#measured_landmarks,landmarks = landmark.grid_landmarks(16, main_batch,[foreground,background])

# Tells window that the robots respond to events
sim_window.push_handlers(real_robot.key_handler)
sim_window.push_handlers(perfect_robot.key_handler)
```

```
def update(dt):


# EKF SWITCH
EKF = False


# Count the Update
global update_number
update_number += 1

real_robot.update(dt)

# Measure from real robot to landmarks
scans = util.scan(landmarks)

perfect_robot.update(dt,landmarks,scans,real_robot,EKF)
# Scan every 20th update
if update_number%10 == 0:

#Update the GUI
count = 3
for i in range(len(measured_landmarks)):
scan_x,scan_y = scans[i]


# Update where the measured landmarks are with respect to the model robot
if not EKF:
# Print the measured locations
measured_landmarks[i].update(scan_x-real_robot.x,scan_y-real_robot.y,perfect_robot.x,
    perfect_robot.y)
if EKF:
measured_landmarks[i].x = perfect_robot.state[count]
measured_landmarks[i].y = perfect_robot.state[count+1]
count += 2

# Draw the window
@sim_window.event
def on_draw():
sim_window.clear()
main_batch.draw()

if __name__ == "__main__":
#update the simulation 120 times a second
pyglet.clock.schedule_interval(update, 1/120.0)

# Tell pyglet to do its thing
pyglet.app.run()
```

The model EKF robot class:

```
#same as real robot class but with no error in movements
class Perfect_Robot(pyglet.sprite.Sprite):
def __init__(self, *args, **kwargs):
super(Perfect_Robot, self).__init__(img = resources.perfect_robot_image, *args,**kwargs)

# Set velocity and rotational speed of robot
```

```
self.velocity = 200
self.rotate_speed = 100.0

# Size of state is 3 + 2 * num of landmarks
num_landmarks = 30
size = 3 + 2 * num_landmarks

# Initialize EKF Matrices
self.cov = np.zeros((size,size))
self.G = np.eye(size)
self.V = np.zeros((size,2))
self.state_error = np.array([[.5**2,0],[0,1**2]])
self.obs_error = np.array([[5**2,0],[0,5**2]])
self.state = np.zeros((size,1))
self.update_num = 0

#Keyboard events
self.key_handler = key.KeyStateHandler()


def update(self, dt, landmarks,scans,real_robot, EKF):

delta_x= 0
delta_y = 0
delta_theta =0


################################## Prediction ###############################

# Input Controls
if self.key_handler[key.LEFT]:
delta_theta = -self.rotate_speed * dt
self.rotation += delta_theta

if self.key_handler[key.RIGHT]:
delta_theta = self.rotate_speed * dt
self.rotation += delta_theta

if self.key_handler[key.UP]:
delta_x = self.velocity * math.cos(math.radians(self.rotation%360)) * dt
delta_y = self.velocity * -math.sin(math.radians(self.rotation%360)) * dt
self.x += delta_x
#rotation is defined as clockwise, hence the negative
self.y += delta_y

if EKF:
# Build G
self.G[0][2] = -delta_y
self.G[1][2] = delta_x

# Build V
self.V[0][0] = delta_x
self.V[1][0] = delta_y
self.V[2][1] = delta_theta

# Compute Prediction Covarience
self.cov = np.dot(np.dot(self.G,self.cov),np.transpose(self.G))+
    np.dot(np.dot(self.V,self.state_error),self.V.T)
```

```
# Build State
self.state[0] = self.x
self.state[1] = self.y
self.state[2] = self.rotation
self.update_num += 1

# Add new
if self.state[3+ 2*len(scans) -1] ==0:
for i in range(3,2*len(scans)):
self.cov[i][i] = 10**5


count = 3
for scan in scans:
self.state[count] = scan[0]
self.state[count+1] = scan[1]
count += 2

############################## Correction #######################################


# Find measurements to landmarks for the prediction model and real robot
count = 3
for scan in scans:


# Find the innovation
r_measured = math.sqrt((scan[0] - real_robot.x)**2+(scan[1] - real_robot.y)**2 )
alpha_measured = (math.degrees(math.atan2(scan[1] - real_robot.y,scan[0] - real_robot.x)) +
    real_robot.rotation)%360


r_expect = math.sqrt((scan[0] - self.x)**2+(scan[1] - self.y)**2 )
alpha_expect = float((math.degrees(math.atan2(scan[1] - self.y,scan[0] - self.x)) +
    self.rotation))%360

# for substracting angles if the difference is larger than 180, take 360 - ans
if (alpha_measured - alpha_expect) > 180:
angle_error = 360-(alpha_measured - alpha_expect)
else:
angle_error = -(alpha_measured - alpha_expect)

# Define innovation
self.innovation = np.zeros((2,1))
self.innovation[0] = r_measured - r_expect
self.innovation[1] = angle_error


# Build H matrix
self.H = np.zeros((2, 3+2*len(scans)))

self.H2 = np.array(
[[-(scan[0] - real_robot.x)/r_measured,-(scan[1] - real_robot.y)/r_measured],
[(scan[1] - real_robot.y)/r_measured**2,-(scan[0] - real_robot.x)/r_measured**2]])

self.H[:,:2] = self.H2
self.H[0][2] = 0
self.H[1][2] = -1
self.H[:,count:count+2] = - self.H2
```

```
count = count + 2

# Build Kalman Gain
kal_const = np.linalg.inv(np.dot(np.dot(self.H,self.cov),self.H.T) + self.obs_error)
self.K = np.dot(np.dot(self.cov,self.H.T), kal_const)

# Update Variance
var_const = np.eye(3+2*len(scans))
self.cov = np.dot((var_const -np.dot(self.K,self.H)),self.cov)

# Correct State
self.state = self.state + np.dot(self.K,self.innovation)

self.x = self.state[0]
self.y = self.state[1]
self.rotation = self.state[2]
```

References

[1] Bailey, Tim, and Hugh Durrant-Whyte. ”Simultaneous localization and mapping (SLAM): Part II.” IEEE Robotics and Automation Magazine 13.3 (2006): 108-117.

[2] Brenner, Claus. ”SLAM Lectures.” YouTube. YouTube, 27 Aug. 2014. Web. 30 Apr. 2015.

[3] Durrant-Whyte, Hugh, and Tim Bailey. ”Simultaneous localization and mapping: part I.” Robotics and Automation Magazine, IEEE 13.2 (2006): 99-110.

[4] Riisgaard, Søren, and Morten Rufus Blas. ”SLAM for Dummies.” A Tutorial Approach to Simultaneous Localization and Mapping 22.1-127 (2003): 126.

[5] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. Cambridge, MA: MIT, 2005. Print.