Robot Localization and Bayesian Filtering: CS460

Joshua Chung (jyc70)

December 14, 2021

1 Bayes Filter

There three initial parameters we need to set up before figuring out the probability.

x_t: State of the floor at x before vacuuming - binary: clean or dirty
x_{t+1}: State of the floor at x after vacuuming - binary: clean or dirty
z_{t+1}: Measurement of the floor after vacuuming - binary: clean or dirty

The only other parameter we have to consider is the control u, which will always be vacuum. From the problem, we know 8 different probabilities.

$$P(x_{t+1} = \text{dirty} \mid x_t = \text{dirty}, u = \text{vacuum}) = .7$$
$$P(x_{t+1} = \text{clean} \mid x_t = \text{dirty}, u = \text{vacuum}) = .3$$
$$P(z_{t+1} = \text{clean} \mid x_{t+1} = \text{dirty}) = .3$$
$$P(z_{t+1} = \text{dirty} \mid x_{t+1} = \text{dirty}) = .7$$
$$P(z_{t+1} = \text{clean} \mid x_{t+1} = \text{clean}) = .9$$
$$P(z_{t+1} = \text{dirty} \mid x_{t+1} = \text{dirty}) = .1$$
$$P(x_{t+1} = \text{clean} \mid x_t = \text{clean}, u = \text{vacuum}) = 1$$
$$P(x_{t+1} = \text{dirty} \mid x_t = \text{clean}, u = \text{vacuum}) = 0$$

With these in ind, we can start calculating $P(x_{t+1} = \text{clean} \mid z_t = \text{clean}, u = \text{vacuum})$.

Given that the $P(x_t = \text{clean}) = c$, we know that $P(x_t = \text{dirty}) = 1\text{-}c$. Let us assume that $t = 0$ and that the robot is just starting to vacuum its environment. Then,

$P(x_1 = \text{clean} \mid z_1 = \text{clean}, u = \text{vacuum}) = \eta P(z_1 = \text{clean} \mid x_1 = \text{clean}) \sum P(x_1 = \text{clean} \mid x_0, u = \text{vacuum}) P(x_0)$

$= \eta P(z_1 = \text{clean} \mid x_1 = \text{clean})(P(x_1 = \text{clean} \mid x_0 = \text{clean}, u = \text{vacuum}) P(x_0 = \text{clean}) + P(x_1 = \text{clean} \mid x_0 = \text{dirty}, u = \text{vacuum}) P(x_0 = \text{dirty}))$

$= \eta(.9)(1 * c + .7 * (1\text{-}c)) = \eta(.27c + .63)$

To normalize the probability value, we must need to normalize the belief as well. This means that: $P(x_1 = \text{clean} \mid z_1 = \text{clean}, u = \text{vacuum}) + P(x_1 = \text{dirty} \mid z_1 = \text{clean}, u = \text{vacuum}) = 1$

$P(x_1 = \text{dirty} \mid z_1 = \text{clean}, u = \text{vacuum}) = \eta P(z_1 = \text{clean} \mid x_1 = \text{dirty}) \sum P(x_1 = \text{dirty} \mid x_0, u = \text{vacuum}) P(x_0)$

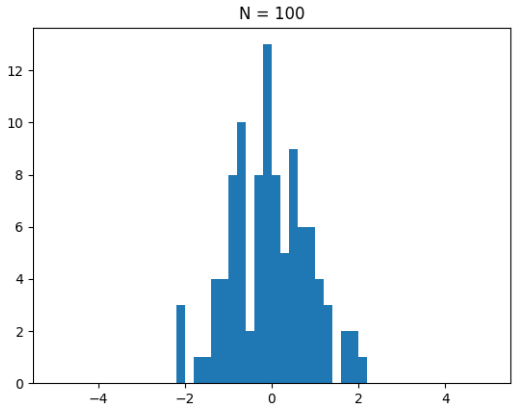
$= \eta P(z_1 = \text{clean} \mid x_1 = \text{dirty})(P(x_1 = \text{dirty} \mid x_0 = \text{clean}, u = \text{vacuum}) P(x_0 = \text{clean}) + P(x_1 = \text{dirty} \mid x_0 = \text{dirty}, u = \text{vacuum}) P(x_0 = \text{dirty}))$

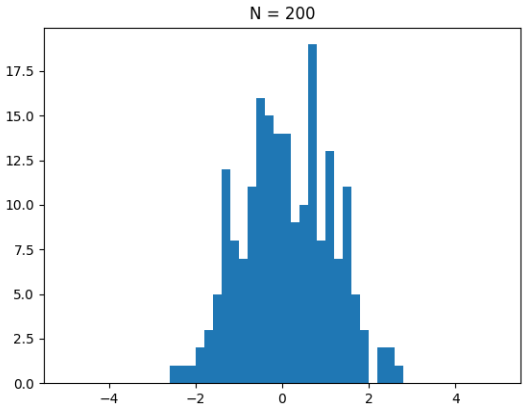
$= \eta(.3)(0 * c + .3 * (1\text{-}c)) = \eta(.09c + .09)$

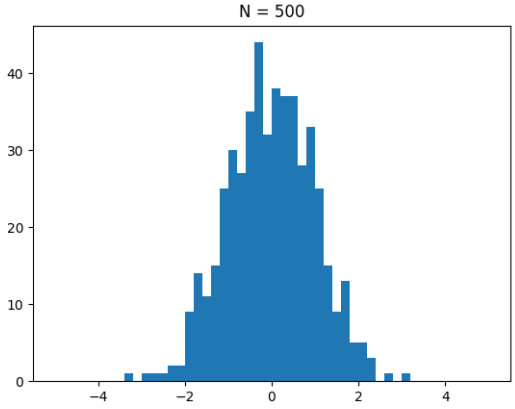
With this in mind, $\eta(.09c + .09) + \eta(.27c + .63) = 1$ and therefore $\eta = 1 / (.18c + .72)$.

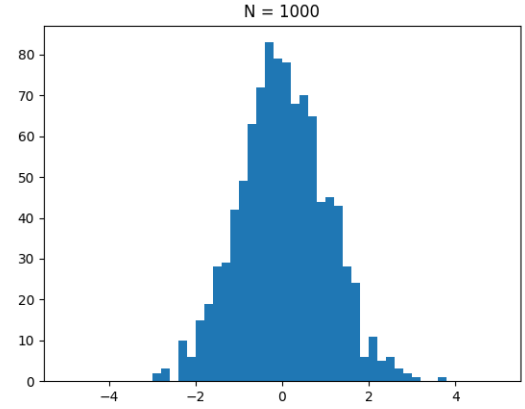
Thus, $P(x_1 = \text{clean} \mid z_1 = \text{clean}, u = \text{vacuum}) = \eta(.27c + .63)$
$=$ **(.27c + .63)/ (.18c + .72)**

2 Sampling from a Distribution





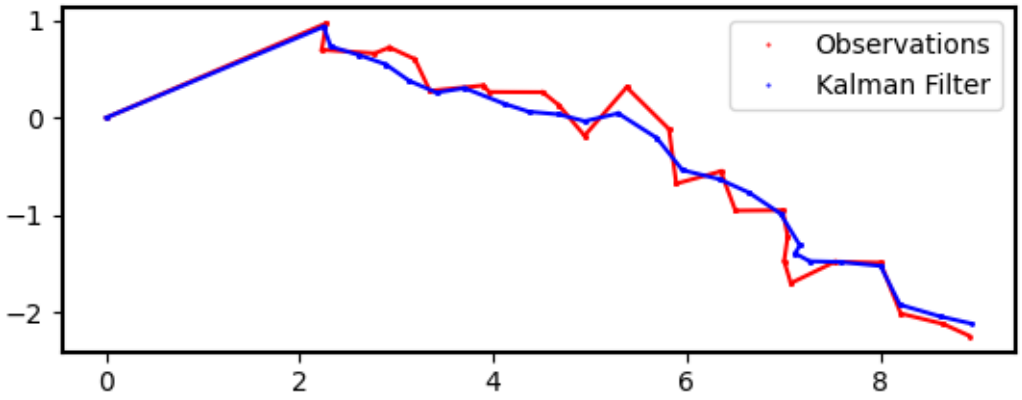




3 Kalman Filter

3.1 Equation and Visualizations

Given the control and observations, this is the robot path after applying a Kalman Filter. The red path is the observations while the blue path is the path with the Kalman Filter.



For the Kalman Filter update equation for the system, the one covered over recitation and lecture was used. Given that Q and R are:

$$\mathrm{Q} = \begin{Bmatrix} 10^{-4} & 2*10^{-5} \\ 2*10^{-5} & 10^{-4} \end{Bmatrix} \mathrm{R} = \begin{Bmatrix} 10^{-2} & 5*10^{-3} \\ 5*10^{-3} & 2*10^{-2} \end{Bmatrix}$$

and the rest of the matrices (A, B, P, H) initialized to an identity matrix, the **Time Update equations** are:

$$x_{next\_state} = A \cdot x_{current\_state} + B \cdot u_{current\_control}$$

to project the state ahead and

$$P_{next\_state} = A \cdot P_{current\_state} \cdot A^T + Q$$

to project the error covariance ahead.
For Measurement Update equations, we must first compute the Kalman gain.

$$K_{next\_state} = P_{next\_state} \cdot H^T \cdot (H \cdot P_{next\_state} \cdot H^T + R)^{-1}$$

With the Kalman gain, we can update the estimate with measurements.

$$x_{next\_state\_updated} = x_{next\_state} + K \cdot (z_{current\_state} - H \cdot x_{current\_state})$$

After that, we update the error covariance.

$$P_{next\_state} = (I * scaler - K \cdot H) \cdot P_{current\_state}$$

Finally, after updating the covariance, we can go onto the next control and observation with these equations. In the start, we can use the initial condition as the x_current_state.

3.2 Methods and Explanation

```
def kalmanFilterTimeUpdate(x, A, B, u, P, Q):
    xTemp = np.dot(A, x) + np.dot(B, u)
    pTemp = np.dot(A, np.dot(P, np.transpose(A))) + Q
    return xTemp, pTemp
def kalmanFilterMeasurementUpdate(P, H, R, x, z, l):
    temp = np.dot(H, np.dot(P, np.transpose(H))) + R
    inverse = inv(temp)
    K = np.dot(P, np.dot(np.transpose(H), inverse))
    xTemp = x + np.dot(K, (z - np.dot(H, x)))
    pTemp = np.dot(((np.identity(2)*l) - np.dot(K, H)), P)
    return xTemp, pTemp
```

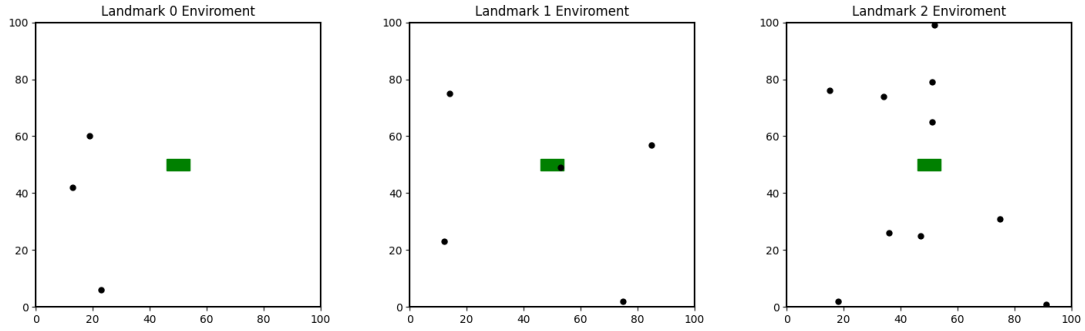
These are the Kalman time update and measurement update methods. It uses numpy's dot method for the different matrices.

```
while(going through the data file):
    u = np.array([[data[i][0]],[data[i][1]]])
    (x, P) = kalmanFilterTimeUpdate(x, A, B, u, P, Q)
    z = np.array([[data[i][2]], [data[i][3]]])
    (x, P) = kalmanFilterMeasurementUpdate(P, H, R, x, z,scaler)
    updated.append(x)
```

For using the methods, the measurement update uses the output of the time update method as its input. After each measurement update, it will save the x coordinate and use it for the time update. At the end, it will plot the predicted path.

4 Non-Linear Landmark-based Localization

4.1 Set Up and Landmark

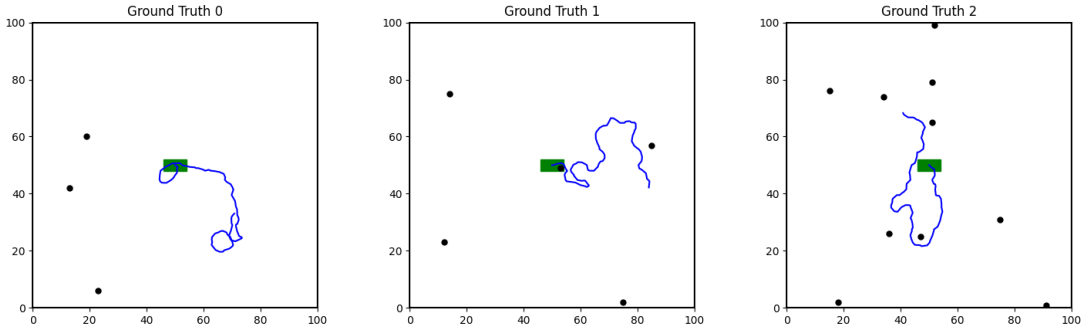


For the landmarks, there are 3 files: landmark_0.txt, landmark_1.txt, and landmark_2.txt. In these files are the land mark locations. To generate these landmarks, you will need to call:

```
def generateLandmark(N):
    file = open("landmark_"+str(N)+".txt", "w")
    temp = np.random.randint(100, size=(N,2))
    file.write(str(N)+"\n")
    for i in temp:
        msg = str(i[0])+" "+str(i[1])+"\n"
        file.write(msg)
    file.close()
```

located in landmarkGenerator.py. It will generate N landmarks given the landmark count N.

4.2 Ground Truth



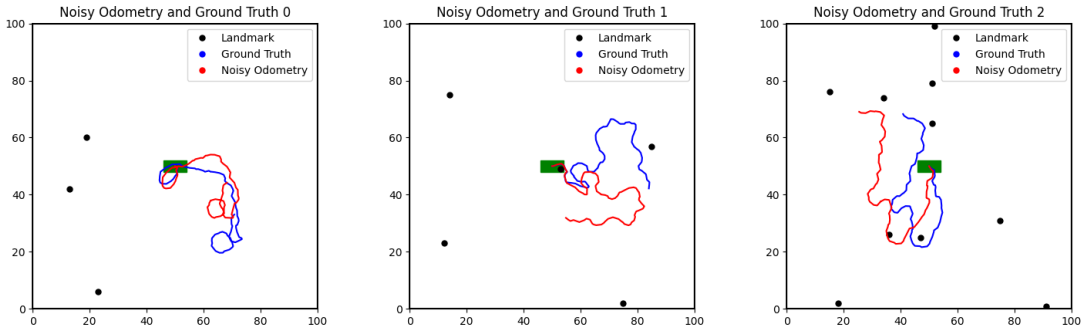
For the ground truth generation, there are multiple parameters it accepts to create a random path. The method is located in groundTruthGenerator.py.

```
def generateGroundTruth(K, init, maxRot, maxTrans, dt):
    x0 = init[0]
    y0 = init[1]
    rot0 = init[2]
```

```
temp = []
temp.append(init)
for i in range(K):
    rot = np.random.uniform(low=-maxRot, high=maxRot)
    rot0 += rot
    rot0 %= np.pi * 2
    trans = np.random.uniform(high=maxTrans)
    movement = trans * dt
    deltaX = np.cos(rot0) * movement
    deltaY = np.sin(rot0) * movement
    x0 += deltaX
    y0 += deltaY
    temp.append((x0, y0, rot0))
file = open("ground_truth_" + str(K) + ".txt", "w")
file.write(str(K) + "\n")
for i in temp:
    msg = str(i[0])+" "+str(i[1])+" "+ str(i[2])+"\n"
    file.write(msg)
file.close()
return temp
```

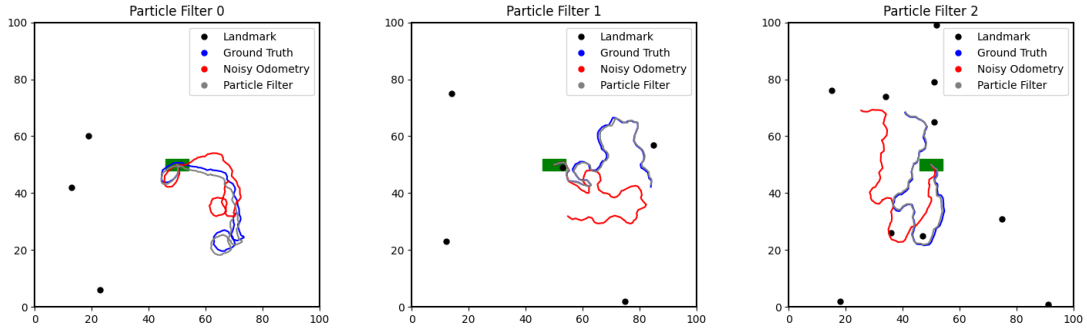
The parameter it accepts are K, init, maxRot, maxTrans, and dt. K is the number of controls it wants to proceed with. init is a tuple of the robot's initial pose (x,y,theta). maxRot is the limit in which the robot can turn clockwise or counterclockwise. maxTrans is limit of the amount in which the robot can traverse. dt is the amount of time it is allowed to translate. For this implementation, K = 100, init = (50,50,0), maxRot = .9, maxTrans = 2, and dt = 1.

4.3 Noisy Odometry Input



For defining the challenge given the landmarks and the ground truth data, you will need to run **sample(enviro, groundTruth)** in **landmarkLocalization.py** to produce a measurement file. The parameters of this methods are both the file name(or directory), landmark_0.txt and ground_truth_0.txt for example. For both the rotation and translation, it will sample at the supposed control with a variance of $\sigma^2 = .1$. For the bearing, it will sample at the supposed bearing with a variance of $\sigma^2 = .0523$. It will sample until it reaches the end of the ground truth file and save a measurement file accordingly. In the figures, the red paths are the predicted paths given the noisy odometry while the blue paths are the actual ground truth paths. You can see that for any landmark environment, the odometry never match the ground truth path.

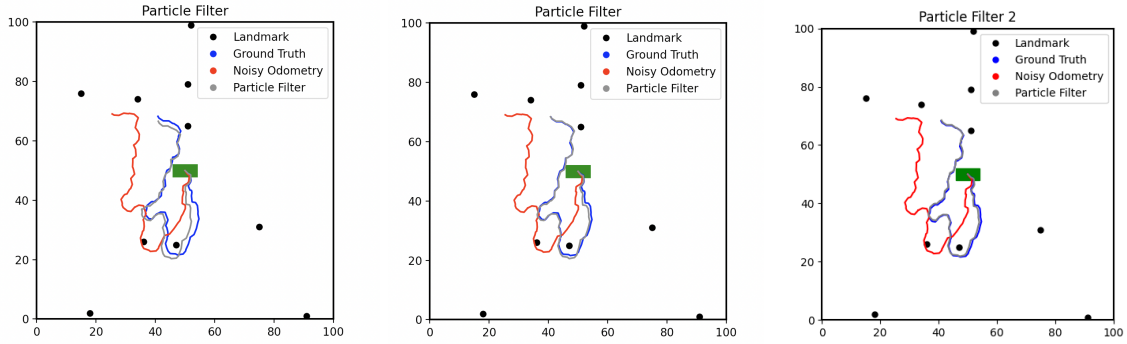
4.4 Localization with Particle Filter



In order to solve the localization problem, this implementation uses a particle filter approach given the measurement and landmark file. The method is called **particleFilter(landmark, measurement, particle)** inside of **landmarkLocalization.py**. landmark and measurement are both file names or directory of the files. particle is the number of particles it will sample at each iteration for each particle. After getting the particle filter path, it will save the path in a PF_estimate_ID.txt file to be used later in the visualization aspect.

```
def particleFilter(landmark, measurement, particle):
    read landmark and measurement file
    Initialize start pose and path = []
    For control and observation in measurement,
        At pose, do a weighted sample of different translations
            and rotation particles given the control for
            particle amount
        Get bearing at each particle
        Get error of those bearings given the observations
        Reject poor sample
        Set pose to best particle
        Append best particle to path
    Save path to a file
```

In this implementation, Blue = ground truth, Red = noisy odometry, Gray = Particle Filter path. You can see that given an increase in the number of landmarks, the accuracy of the localization increases. In Particle Filter 3, the particle filter path is basically on top of the ground truth data point while in Particle Filter 0, it is quite off by the end of the path. In these figures, the number of particles sampled were set to 100.



From left to right, the number of particles sampled were **3, 10, and 100**. As you can see, with lower particle samples at each iteration, the particle filter estimated path does not quite converge to the ground truth path.

4.5 Visualization

In order to visualize at the different levels, there are 4 visualization methods.

```
env = "landmark_2.txt"
truth = "ground_truth_2.txt"
measurement = "measurement_2.txt"
pf = "PF_estimate_2.txt"
init = (50,50,0)

drawLandmark(init,env)
drawGroundTruthEnv(init, env, truth)
drawNoisyOdom(init, env, truth, measurement)
drawPF(init, env, truth, measurement, pf)
```

- init is the initial pose of the robot
- env is the directory of the landmark file
- truth is the directory of the ground truth file
- measurement is the directory of the measurement file
- pf is the directory of the pf estimate file

4.6 EKF