CSE 276A - Introduction to Robotics

HW3

Drishti Megalmani A59001657
Sai Ashritha Kandiraju A59001630

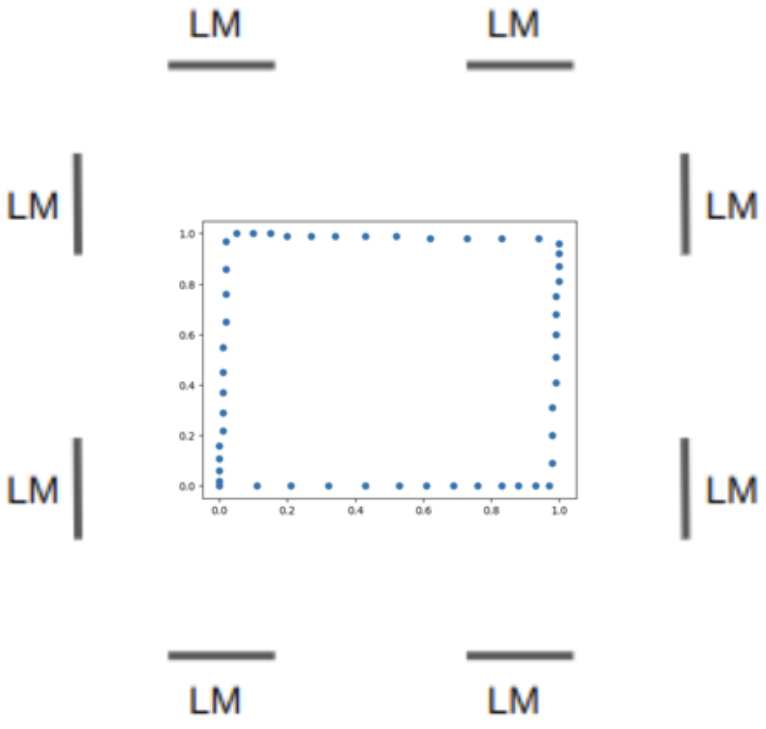
In this homework, we aim to implement a version of the Simultaneous Localization and Mapping (SLAM) technique using the Discrete Kalman Filter algorithm and evaluate its performance.

METHODOLOGY

We set up an environment in a 10ft x 10ft area with 8 distinct landmarks, with 2 apriltag landmarks on each side. We implemented the Discrete Kalman Filter based SLAM system as described in the lectures. We used the april_detection_node given to us in the solution codebase to detect our landmarks.

For the square map, we start from a position inside the square, which we assume as the origin of the map frame [0,0,0]. We perform a square motion of about 3.2 ft X 3.2 ft and cover the waypoints as given below:

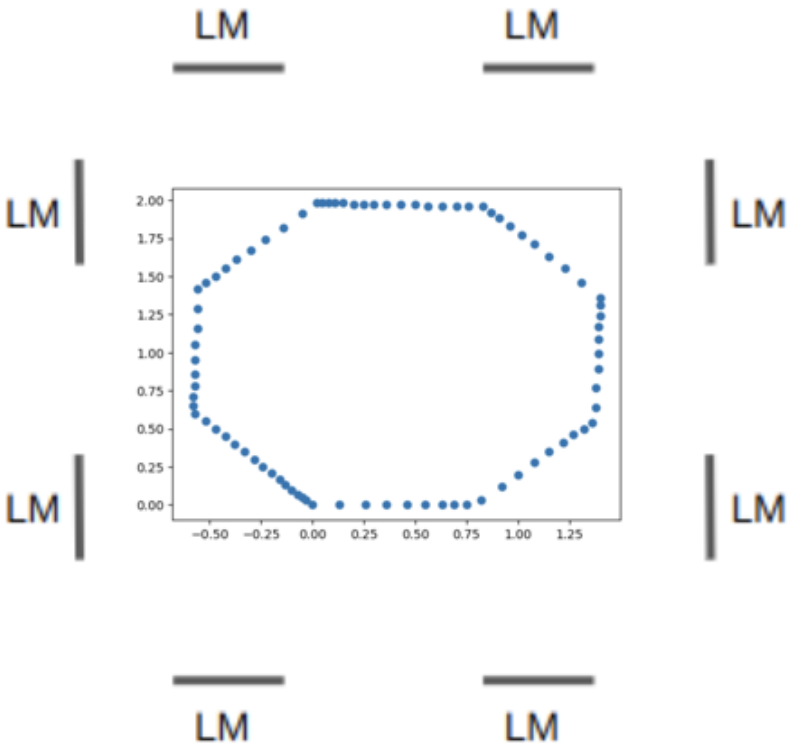
[0,0,0] , [1, 0, pi/2] , [1, 1, pi] and [0, 1, -pi/2] and [0, 0, 0].



The waypoints indicate coordinates in meters. We decided on a square side length of 1m as it gives enough room to get visual feedback from all the positioned landmarks.

For the octagon map, we start from a position inside the square, which we assume as the origin of the map frame [0,0,0]. We chose the square of side 2m within which the 8 point octagon was nested in. We chose this to allow for visual feedback and to not let the movement be too small (the smallest in our case is 0.58m). We move in an octagon and cover the waypoints as given below:

[0.0, 0.0, 0.0], [0.82, 0.0, np.pi/4], [1.4, 0.58, np.pi/2], [1.4, 1.4,3*np.pi/4], [0.82, 1.98, np.pi], [0, 1.98, -np.pi/4], [-0.58, 1.4, -np.pi/2], [-0.58, 0.58, -3*np.pi/4], [0.0, 0.0, 0.0]



KALMAN STRATEGY

The Kalman strategy that we employed for SLAM has two main steps - prediction and update.

Prediction step

In the prediction step, we try to estimate the state and uncertainty of the map (pose of landmarks + pose of robot) using it' s previous estimates using the equations below.

$$s_{t|t-1} = Fs_{t-1|t-1} + Gu_t$$
$$\Sigma_{t|t-1} = F\Sigma_{t-1|t-1}F^T + Q_t$$

The variables used in the above equations are described as follows

$s_{t|t-1}$ - indicates the state of the system at timestep *t* given the information about the system state at time *t-1*. The state of the robot can be denoted by x_r, y_r, θ_r which indicate (x_r, y_r) position and θ_r orientation of the robot in the map frame. Similarly, we can denote the poses of the landmarks using $x_{l_x}, y_{l_x}, \theta_{l_x}$ in the map frame. The assumption here is we know how the landmarks are placed in the map (stuck to the wall). So in the map frame, we need only 3 coordinates to get the complete state information of the landmarks instead of 6. Thus we can stack up x, y, θ in a column vector to obtain the state vector in the map frame. At the end of the SLAM algorithm it would result in a state vector of size 27 x 1. We also need to note that, even though the state vector contains pose information of the robot as well as the landmarks, we have the assumption that the landmarks we are using are stationary so their state does not change due to any motion of the robot. So in our implementation of the prediction model, we only update the state of the robot in each prediction.

F - is a matrix that relates the state of system at time t to the state of system at time t-1 in the absence of any control update from the robot

u_t - Denotes the $[v_x, v_y, v_\theta]$ velocities of the robot motion at a timestep *t*

G - is a matrix that linearly relates the robot motion(velocities) to the change in state after a time difference of Δt

$\Sigma_{t|t-1}$ - is called a covariance matrix that indicates the uncertainty/covariance in the state of a system at a timestep *t* given the information about the uncertainty of the system at time *t-1*. This is used as a measure of how much we can “trust” the prediction at a timestep *t.* The dimension of the covariance matrix is equal to *n* x *n* where n denotes the number of values in the state vector. The diagonal of the covariance matrix indicates the variance of a state variable and the non-diagonal elements indicate the covariance amongst themselves. We need to note that the covariance matrix is symmetrical as the covariance between two variables is commutative.

Q_t - denotes the control noise of the system. Since the motors of the mbot are not very reliable, noise gets introduced into the motion of the robot. This noise can be assumed to be white and can be modeled using a gaussian distribution. The control noise can be factored into our prediction using the matrix Q_t which denotes the covariance of the noise.

Update step

In the update step, we use the previously computed predictions and the information from the camera measurements to update the state of the system. We use the prior estimate of our state and a weighted difference between the actual measurement and predicted measurement to obtain the posterior estimate of the state as shown in the first equation below. This equation helps in achieving simultaneous localization and mapping of the system.

Intuitively, the update process combines information from two different random variables to arrive at a distribution whose covariance is relatively lower. When the update step converges, the resulting state vector denotes the mapping of the landmarks.

$$
\begin{aligned}
s_{t|t} &= s_{t|t-1} + K_t(z_t - Hs_{t|t-1}) \\
K_t &= \Sigma_{t|t-1}H^T S_t^{-1} \\
S_t &= H\Sigma_{t|t-1}H^T + R_t \\
\Sigma_{t|t} &= (I - K_t H)\Sigma_{t|t-1}
\end{aligned}
$$

R_t - denotes the measurement noise in our observation model. Since any observation captured using sensors is noisy and contains some uncertainty, this noise can be modeled as a gaussian distribution with mean 0 and covariance defined using R_t. Note that the measurement noise R_t is assumed to be independent of control noise Q_t

K_t - known as the *Kalman gain* which is a factor that weighs the amount of information from the camera measurement that gets passed into the update step. The error $z_t - Hs_{t|t-1}$ is known as innovation. Intuitively, if we trust our measurements more than predictions, we increase the gain and vice versa. If measurement noise R_t is very minimal (tends to 0), this means that we can trust the measurements a lot, indicating that we can use a large gain to weigh the innovation heavily. Similarly, when $\Sigma_{t|t-1}$ is very minimal, it indicates that the covariance in the system is very low thus increasing our confidence in our state predictions. So we use a very small gain to weigh the innovation and try to keep the posterior estimates close to the predictions. The dimensions of this matrix are *n* x *m* where *n* indicates the total number of state vector values and *m* indicates the total number of measurement values.

z_t - indicates the measurements obtained from landmark detections at timestep *t*. The measurement vector for a landmark can be denoted by x, y, θ which indicate the pose of the tag in the optical/robot frame. For simplicity, we aren't doing a static transform between camera and robot and considering the camera as the robot frames origin. Note here that in the update step,

only measurements of known landmarks (with map coordinates in the state vector) are considered. Any new landmarks detected will not be used in the update step. Detection of new landmarks during the algorithm run is discussed later.

H - This matrix is used in the update step to ensure that the map coordinates of the landmarks in the state vector are transformed to the robot coordinates before computing the innovation/error. The H matrix thus denotes the linear transformation that maps us from the state space to the measured space. The dimensions of this matrix are *m* x *n* where *n* indicates the total number of state vector values and *m* indicates the total number of measurement values.

IMPLEMENTATION DETAILS

F Matrix - If we don't drive the robot, F explains how the previous state of the system and the current state of the system are related. In our case, when the controls to the robot motors are not applied, we know that the robot does not move. So, F in this case would be an identity matrix. In our implementation however, we ignored the F term as it was an identity matrix to avoid unnecessary stacking when state/covariance gets updated.

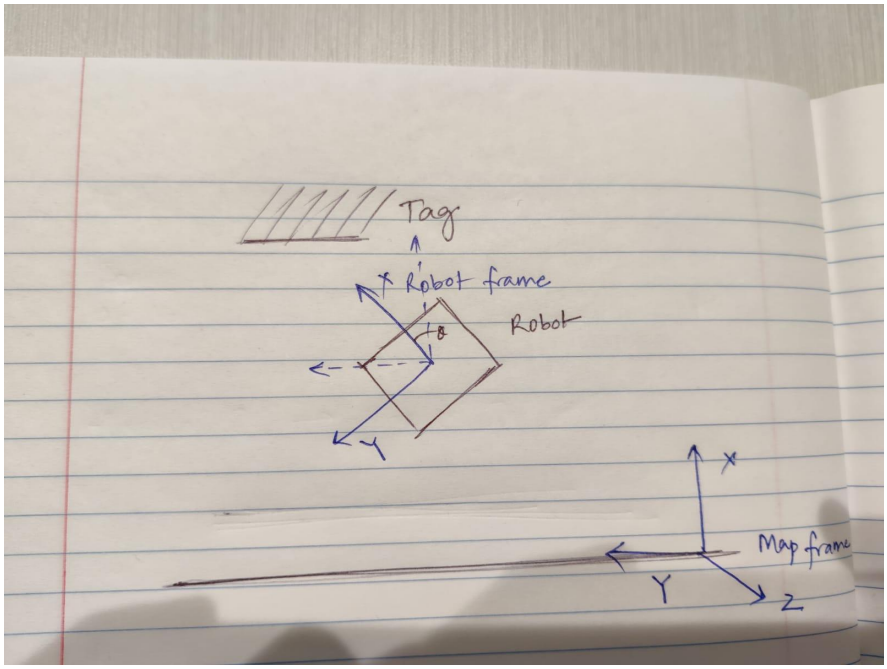
G Matrix - As mentioned earlier, u_t is a control variable or input variable - a measurable (deterministic) input to the system. u_t denotes the $[v_x, v_y, v_\theta]^T$ velocities of the robot motion at a timestep *t. G* is the control matrix or input transition matrix (mapping control variables to state variables). Since we need the change in state for the prediction step, we use *G* to linearly transform the robot motion(velocities) to the change in state after a time difference of Δt. Our system is a linear velocity system so, G would be [Δt, Δt, Δt]. However, in the implementation we compute the displacement $G * u_t$ as the update_value directly. Thus $G * u_t$ would be $\Delta x, \Delta y, \Delta\theta$.

H Matrix - As mentioned earlier the H matrix denotes the linear transformation that maps the state space to the measured space. Thus the H matrix should enable the transformation from map frame to robot frame. We have the coordinates of the (one) tag in map frame in the state vector, let these be denoted by (x_m, y_m, θ_m). Let the measurement of the landmark obtained in the robot frame be denoted by (x_r, y_r, θ_r). Let the robot in the map frame be denoted by (α, β, θ).
Thus the state vector would be $[\alpha, \beta, \theta, x_m, y_m, \theta_m]^T$. The measurement vector would be $[x_m, y_m, \theta_m]^T$

To easily compute the H matrix, let us assume a frame of reference parallel to the map frame but aligning with the origin of the robot (as shown in the below figure in dotted lines). Thus

obtaining the coordinates of the landmark in this frame of reference would be a simple translation from the map frame to this frame. Let us assume that the robot frame makes an angle of θ with this frame. Thus to convert the landmark coordinates to the robot frame, we have to now just multiply the coordinates after translation with the rotation matrix $\begin{bmatrix} cos\,\theta & sin\,\theta & 0 \\ -sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Thus H = $\begin{bmatrix} cos\,\theta & sin\,\theta & 0 \\ -sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$



The H matrix is not constant and changes with every update step. The input to compute the H matrix is the state vector and the current measurements at that time step. Thus the dimensions of the H matrix are *m* x *n* where *n* indicates the total number of state vector values and *m* indicates the total number of measurement values.

When more than one landmark is detected, the same logic for computing H for one landmark can be extended and H can be stacked as required. For multiple landmarks, the template for H matrix will be as follows

$$H = \begin{bmatrix} R & 0 & \ldots & 0 \\ 0 & R & \ldots & 0 \\ 0 & 0 & \ldots & R \end{bmatrix} \begin{bmatrix} -I & I & 0 & 0 & \ldots & 0 \\ -I & 0 & I & 0 & \ldots & 0 \\ -I & 0 & 0 & 0 & \ldots & I \end{bmatrix}$$

where R = $\begin{bmatrix} cos\,\theta & sin\,\theta & 0 \\ -sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and I = $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Q Matrix

The Q matrix initially has dimensions of 3 x 3 to indicate the control noise in the system. Every time a landmark is added, Q has to be stacked with a 3 x 3 matrix in the bottom right corner. This matrix denotes the control noise for landmarks in the system. We initially assumed that the noise for landmarks is 0 in the control process. But later added very small values as part of tuning as that improved the trajectory.

Since Q denotes the control noise of the system and as the mbot is not very reliable, noise gets introduced into the motion of the robot. This noise can be assumed to be white and can be modeled using a gaussian distribution. The control noise can be factored into our prediction using the matrix Q_t which denotes the covariance of the noise.

To get an initial estimate for the noise, we moved the robot by 1m in multiple consecutive runs and obtained the error for this movement. We then took the average of this error and considered that as the noise in the x, y directions. Similarly, we rotated the robot by pi in multiple consecutive runs and obtained the error for rotation. We increased this a little bit to make it 1/10th of the noise in x,y directions. We then tuned the initial estimates by decreasing this noise and increasing initial uncertainty.

Assumptions
The control noise is also assumed to be constant throughout and not dynamically changing as the robot moves. We think this is a fair assumption.

R Matrix

The R matrix denotes the measurement noise in our observation model. Since any observation captured using sensors is noisy and contains some uncertainty, this noise can be modeled as a gaussian distribution with mean 0 and covariance defined using R_t. This matrix will be stacked up every time a landmark is added. R has to be stacked with a 3 x 3 matrix in the bottom right corner of the existing matrix. Since the camera has been well calibrated as part of the last HW, we observed that the measurement noise was lesser than the control noise.

To get an initial estimate for the noise, we took the april tag detection consecutively and got the average error across all readings. We started off with this value but this was too low. We tuned this initial estimate and finalized the value that gave the best mapping

Assumptions

The measurement noise R_t is assumed to be independent of control noise Q_t.

The covariance is the same for all landmarks.
The matrix is constant with every timestep.

Initialization - We initialized the following variables as below:

1. statevector (s) = [0 0 0], where s is the state at timestep 0 containing the robot initial coordinates [0,0,0] in the map frame

2. Covariance (sigma) = np.array([[1.0, 0.0, 0.00],
   [0.0, 1.0, 0.00],
   [0.0, 0.0, 0.01]]),
   We have assumed that the uncertainty in x direction is the same as uncertainty in y direction (as motion is uniform). We also assumed the uncertainty wrt x and y coordinates is higher than the orientation. We tried multiple values to initialize sigma and finalized on the above matrix. When we tried with very low values, the predictions were very close to the updates. When we tried very large uncertainty, after the next update step, it would shoot down and become approximately equal to the uncertainties of the measurements since the update step does a weighted mean.

3. self.R = np.array([[0.01, 0.0, 0.0],
   [0.0, 0.01, 0.0],
   [0.0, 0.0, 0.001]])
   Here R defines the covariance in the camera readings. Since we calibrated the camera pretty well in HW2, we did not observe (explained later) too much noise in the measurements. Thus we initialized R with low values.

4. self.Q = np.array([[0.03, 0.0, 0.0],
   [0.0, 0.03, 0.0],
   [0.0, 0.0, 0.003]])
   Here Q defines the covariance of the control/process noise. Since mbot motors are not very precise, we ran some experiments (explained later) and captured the noise estimates. Initially we set Q to a higher value but later tuned this and made it lower by increasing the initial uncertainty sigma

5. self.H = np.array([[0.0, 0.0, 0.0],
   [0.0, 0.0, 0.0],
   [0.0, 0.0, 0.0]])
   We initialized H to be zeros as initially we don't capture any measurements so we don't need a transformation.

6. F for our system is an identity matrix. This is because, when there is no control provided i.e, no movement of the robot, the state doesn't change.

ALGORITHM

We initialize all the variables with the above defined initial values and assume that the position of the robot is at [0,0,0].
For each time step of 0.1, we publish the *update_value* onto the twist controller to initiate forward motion of the robot (based on the waypoints defined).
Whenever a new april tag is detected, we update the state vector with the new detected april tag and restack the sigma matrix, Q, R to match the new updated state and covariance matrices. After the predict step, if april tags are detected we also perform the update Kalman filter steps.

To avoid updates being too frequent, we are performing it for every 3rd timestep given that the april tag is detected. This design choice gave a better run. The predict and update statements are performed until the error between the current_state and the waypoint is greater than 0.05 (as tuned in HW2). The edge cases with april tag detection and our solution to them are explained below:

Detection of Landmarks

No april_tag detected:
We fall back to the open loop system where the updates are made via the PID controller at a specific timestep. The uncertainty matrix increases in value for good reason since we are operating in an open loop system. The update value for the state will be the same as the prediction.

New april_tag detected:
Whenever a new tag is detected, we first update the state vector and stack the state of the april tag to the vector, and resize the corresponding sigma and noise matrices. We then perform the update step with the measurements we get from the camera.

Existing april_tag detected again:
If an april tag that has previously been detected is seen, we directly perform the predict step (with no additional stacking required) and follow up with the update step specifically for the tags detected and the robot. The uncertainty decreases for the april tag detected.

OBSERVATIONS

1. When we don't run the update, the uncertainty in the prediction increases with every timestep. When the april tags were detected, the april
2. Tuning how fast or slow the robot moved was important to find the right balance between the april tag detections resulting in either too frequent updates or very few updates. Both

resulted in inaccurate results. The right balance was achieved with the PID values [0.0185, 0.0015, 0.09] and a timestep of 0.02 for our setup.

CHALLENGES

1. We faced issues with the initial april tag detection setup from the solution provided. The hw2_solution file and the corresponding april detection node provided weren't coherent. The lookup transform resulted in a *'Lookup Exception'.* We took some time to solve this issue and had to experiment with the tree structure and the commands.
2. We also felt a huge learning curve and struggled with the transformation matrices associated with the algorithm and figuring out the right values and tuning them.
3. Despite having the algorithm coded up and trying to perform a couple of runs for both the path structures, due to the challenge faced in the transformations, we unfortunately weren't able to achieve the right results and the right localizations.

CONTRIBUTION

Both of us contributed equally on all sections of the homework (from calibration, design thinking, implementation, testing and report).