Extended Kalman Filter Algorithm:
We use the following EKF algorithm: (source: Probabilistic Robotics)

1: **Algorithm Extended_Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \; \Sigma_{t-1} \; G_t^T + R_t$
4: $K_t = \bar{\Sigma}_t \; H_t^T (H_t \; \bar{\Sigma}_t \; H_t^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6: $\Sigma_t = (I - K_t \; H_t) \; \bar{\Sigma}_t$
7: return μ_t, Σ_t

Our Motion Model to predict the next state given our control input:
$x_k = x_{k-1} + v\cos(\theta_{k-1}) * \delta t$, $y_k = y_{k-1} + v\sin(\theta_{k-1}) * \delta t$, $\theta_k = \theta_{k-1} + w * \delta t$
Where x_k, y_k, θ_k, are the pose coordinates of the jetbot. v is the translational velocity of the robot.
The state vector μ_k consists of the above 3 variables and the x and y coordinates of the landmarks discovered *so-far*. Since the landmarks are stationary, they do not change with robot motion. Hence, the motion model equation for them is identity (in addition to the noise).
Since our motion algorithm is still *point and move*, The translational velocity remains unchanged during translational motion as well as the orientation. The orientation only changes during rotation when v is 0.
We call the EKF predict/update functions after both rotation and translation in our motion.

We then take the Jacobian of the above motion model function $f(\mu_{k-1}, u_{k-1})$, which becomes our matrix G. This matrix is calculated in the function `get_G()` in the EKF class.

For the function $h(\hat{x}) = \hat{z}$, this is simply a frame transformation of the landmark coordinates from the global frame to the robot's reference frame. The equations (calculated in the function h() in the EKF class):
$z[j] = (x_i - x_r) * cos(\theta) + (y_i - y_r) * sin(\theta)$ for the x coordinate, and
$z[j+1] = -1 * (x_i - x_r) * sin(\theta) + (y_i - y_r) * cos(\theta)$ for the y coordinate, where x_i and y_i are state values of the landmark and x_r, y_r and θ are state values of the jetbot's pose.
The matrix H the Jacobian of the above function, calculated in the function `get_H()`.

We initialize the motion model noise matrix R (mentioned Q in our code) and measurement noise matrix Q (mentioned R in our code) as follows:
Motion model noise: x and y coordinate gaussian noise of 0.15 meters variance and for rotation, gaussian noise of pi/8 radians. For the measurement noise, we add a noise of 0.1 meters.

We initialize our Sigma (covariance matrix) as a linear combination of Q and R.

We experiment with the noise:

- A high noise was not able to converge properly, with the covariance trace blowing up and state estimations/corrections not correct.
- A low noise would allow the covariance to converge but the state vectors would not account for the drift in motion of the robot.
- Overall a noise was chosen that would account for the drift to some extent but also allow the covariance trace to go down with more iterations (most of the times)

When do we call the EKF prediction/updation?

We take 10 measurements of april tags (in the localization node) instead of 1 so that we can update the states based on multiple observations and if possible using multiple landmarks before we move again.

But at a time we only update using one observed landmark. We only calculate the H jacobian for the indices of that landmark and only update that state for the observed landmarks.

If the landmark observed is considered a new landmark: We add the x and y coordinates (as calculated in the global frame based on the observation) to the state vector. We increase the dimensions of all the matrices by 2x2 and add the values to them that we used for initialization for noise matrices.

How to detect new landmarks:

We only use id 1 and 2 april tags. For each landmark observed, we calculate (estimate) it's coordinates in the world frame. We calculate the mahalanobis distance from each existing landmark and if the minimum distance is greater than a threshold of 60 cm (chosen based on experimental results that did not blow up the landmarks), we consider it as a new landmark. Else, we consider it as the landmark with the minimum distance.

Planning: We follow Open-Loop planning, using the estimated positions using our motion model. Also, the g function is simply replaced by the planned waypoints as they represent our estimated positions in the code.

We tried to replace open-loop motion with closed loop motion by replacing the estimated current_pose with the corrected current-pose from the EKF state vector after reaching a waypoint. But since the state vector correction was not good especially in the initial stages, we did not go ahead with this approach.

Results:

Robot State results: Our robot state results are not good:

- After 1 circle: the Actual drift in the robot's final position is very low (less than 5 cm) even after using open loop motion. But the state vector showed positions within 30 cm radius on average (ignoring 2 runs where the final position was 100 cm off).
- After 2 circles: these results were slightly better, with the final state position being within 15-20 cm of the final position (which was not the same as origin because of drift).

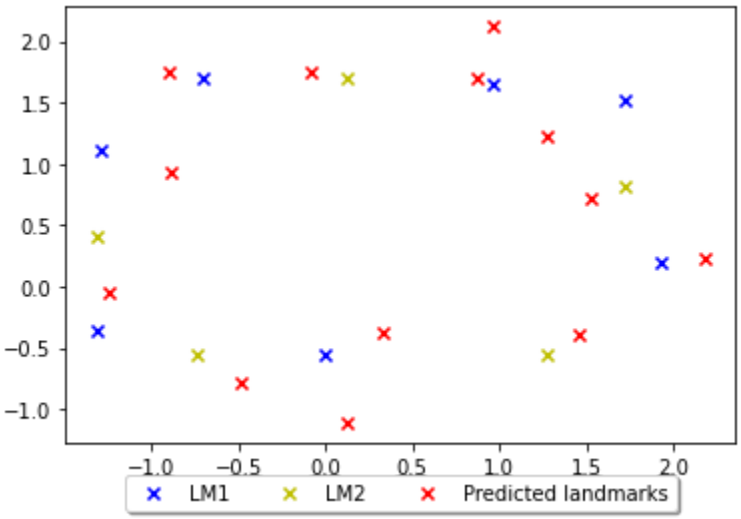
- After 4 circles: This did not help with the convergence much, as the robot started drifting from it's path. The covariance trace decreases but then again increases with more changes to the state vector and new measurements. The final state position is off by a significant distance from the actual robot position. It's closer to the origin whereas the robot has drifted by almost 100cm. *We plan on working towards improving our state correction by more experiments and figuring out what exactly is going wrong.*
- Figure 8: The robot's actual positions were close to the estimated (planned) waypoints. The state positions are within 20 cm of the actual position on an average. However as the motion progresses, the variance increases, Perhaps multiple figure 8s would be better but with open loop motion that would result in more drift.

Our mapping of the environment is not perfect but most of the observed landmarks are close to the actual positions.

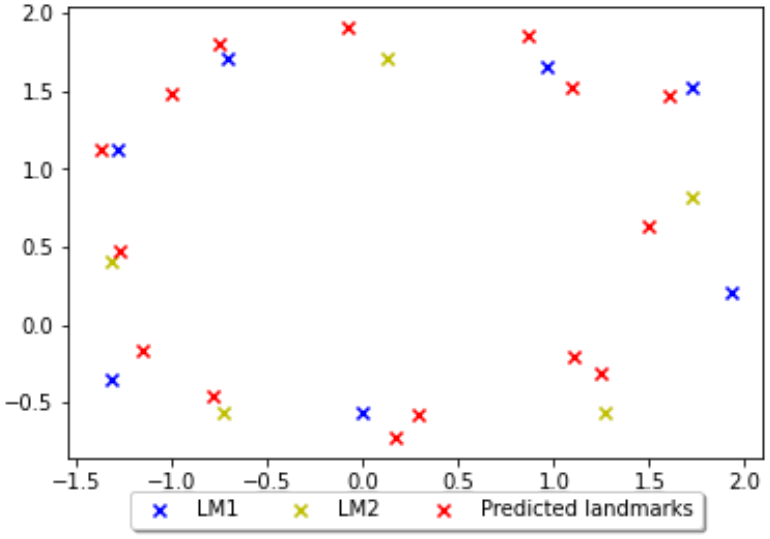
We also think that identifying 2 landmarks instead of 1 for 1 real landmark is not a problem if their final corrected positions are close to the actual position.

These are the following mapping estimates we obtained for each of the above trajectories:

Single Circle: In this case, one of the landmarks was not observed, and there was major drift in the predicted position of some of the landmarks.



Double Circle : We obtained the best results in this case. Almost All the landmarks were detected with minor drifts.



Four Circles - In this case, although all the landmarks were detected, we see that some of the landmarks were detected more than once, and there is also deviation for some of the landmarks.

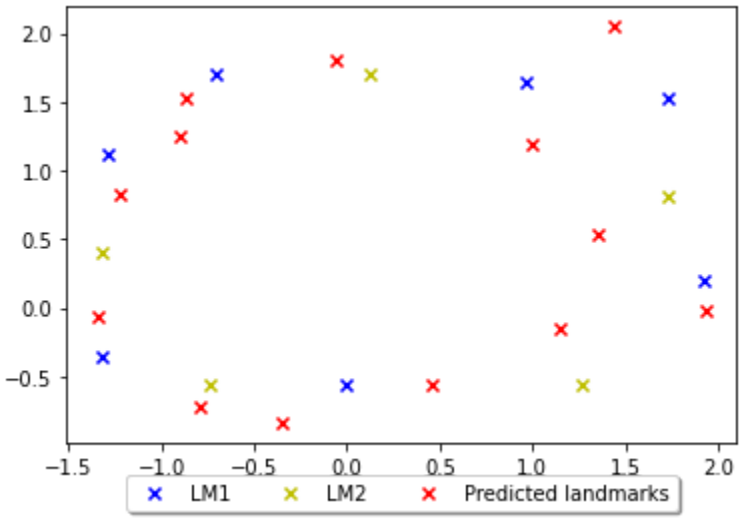


Figure Eight - Although we observed all the landmarks, there were major deviations as can be seen from the below graph