CSE 276A: Intro to Robotics
Homework 3

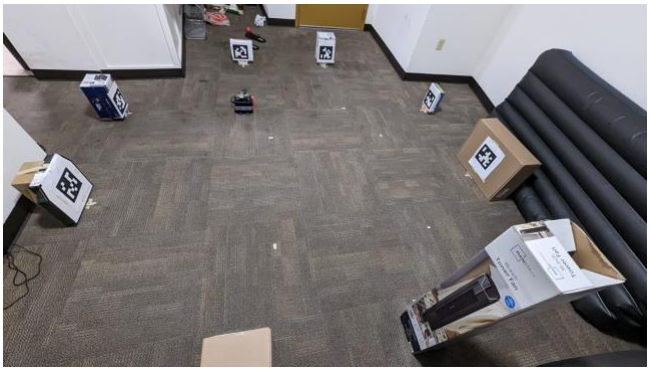
Orish Jindal (A59010554)

Introduction:
Implementation of Kalman Filter, a version of the Simultaneous Localization and Mapping (SLAM) technique and evaluation of its performance over variations in robot trajectory.

Environment and Problem Statement:
The environment that needs to be mapped is in the form of a 10ft ×10ft area with 8 landmarks (AprilTags) placed two on each side of the square environment facing inwards.

The provided robot needs to be placed at a random position inside the environment with no prior information of the pose of landmarks. It needs to navigate through the environment for a pre-decided path while preparing a map of the same environment and simultaneously correcting its own position with respect the map that is being prepared.

Visualization of Environment:
The environment that is used for this project is shown below:



Complete Methodology:
The methodology to complete this project can be viewed in terms of five major nodes interacting with each in a specific manner.

1. rb5_vision:
   This node activated the camera which is further used for detection of the appropriate landmarks.

2. apriltag_detection_node:
   This node is responsible for detecting the landmarks, i.e., the AprilTags. The topic published from this node is subscribed by the new node ‘slam’ to repeatedly obtain the observations (in form of the pose of AprilTags) at any given robot pose.

3. slam:
   This new node receives messages from the topics published by the two nodes, i.e., Twist messages from the hw3_sol (includes PID controller) which gives information about the control input in form of 3D velocity vector (for x, y and yaw) and apriltag_detection_array

messages from apriltag_detection_node which gives information of the pose of the visible AprilTags in optical frame of robot's camera at any state.

This set of information is called at particular places in the code of the slam node which then calculates the updated state of the whole environment (including robot) with the help of Kalman Filter SLAM and updates it in the global state variable and publishes pose2d messages, subscribed by again, the hw3_sol to obtain the corrected pose estimation of the robot. The implementation of Kalman Filter algorithm is explained in later in this paper.

4. hw3_sol:
   This node is using the inputs published by slam node in the form of the updated current pose of the robot and is implementing the PID controller by calculating desired short-term increment in the position of the robot by taking the desired path as target and then publishing 'Twist()' which is further used by the control node (along the slam node for next run of the loop as stated above).

5. hw2_mpi_control_node:
   this node receives the messages from the Twist() node and then converts it to the form of PWM signals for each motor which is responsible for the resulting motion of the robot.

Implementation of Kalman Filter:

Kalman Filter is a special case of Bayesian Filter technique, used to implement Simultaneous Localization and Mapping. There are three basic elements of Kalman Filter SLAM: estimation of normally distributed prior pose of the robot, predicting the next pose based on odometry and finally with the help of observation model, updating the predicted mean and covariance of the robot pose to use it as a prior in the next repetition of the loop.

Firstly, two global variables: *mu* (denoting the state of the environment) and *sigma* (denoting the corresponding covariances) are declared at the start of the code which will keep track of the mean of the poses of robot and AprilTags in the world frame, repeatedly corrected over time. These numpy arrays start with just the pose of robot and grow as the robot starts to see the AprilTags on its way by appending the 3D vectors to the *mu* and 3x3 block matrices to *sigma*. Each tag is assigned a particular index in these arrays with the help of python dictionary which is useful while doing SLAM calculations on *temp_mu* and *temp_sigma* (the smaller versions of original arrays dependent on the number of visible tags at that instance) and writing back the updated values to the original arrays at the right places. That way, if a tag disappears and then reappears, the program will know which indexes in the global arrays (*mu* and *sigma*) should be interacted with for further calculations and maintain a correct record of everything.

The following two steps are repeated in a loop until the program runs:

1. **Prediction**
   In this step, the mean and co-variance of future pose of the robot is predicted based on the motion model. The equations are as follows:

$$\boldsymbol{\mu}_{t+1|t} = F\boldsymbol{\mu}_{t|t} + G\mathbf{u}_t$$
$$\Sigma_{t+1|t} = F\Sigma_{t|t}F^{\top} + W$$

   Where, F is an identity matrix, G is identity times Δt and W is the co-variance matrix formed by random noise. $\mu_{t/t}$ and $\Sigma_{t/t}$ are the prior mean and co-variance of pose of the robot at

time t and $\mu_{t+1/t}$ and $\Sigma_{t+1/t}$ are the predicted mean and co-variance at time t+1 based on the Kalman updates of time t. u_t is the 3D velocity at time t.

2. **Update**
   This step is responsible for updating the predicted estimates based on the observations that are made when the robot is at timestamp t+1. The equations are as follows:

$$\boldsymbol{\mu}_{t+1|t+1} = \boldsymbol{\mu}_{t+1|t} + K_{t+1|t}(\mathbf{z}_{t+1} - H\boldsymbol{\mu}_{t+1|t})$$
$$\Sigma_{t+1|t+1} = (I - K_{t+1|t}H)\Sigma_{t+1|t}$$

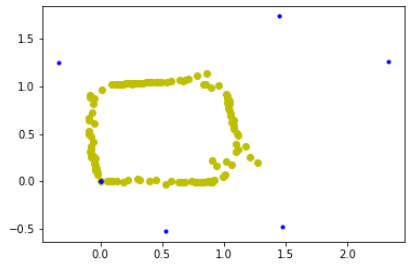
$$K_{t+1|t} := \Sigma_{t+1|t}H^\top \left(H\Sigma_{t+1|t}H^\top + V\right)^{-1}$$

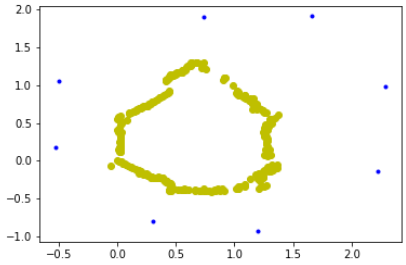
   Where, K is the Kalman gain calculated as shown. z_{t+1} represents the observations that are calculated by just rotating (without translation) the 2D poses of the AprilTags (obtained in the camera frame) with respect to the world so that the H (dim: *3n x 3(n+1)* where n is the number of tags visible) matrix becomes a combination of n vertically stacked n negative identity block matrices and a giant identity matrix stacked horizontally. Ex. If there are two tags visible at a particular instance, then H will be as follows:

```
array([[-1,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0, -1,  0,  0,  1,  0,  0,  0,  0],
       [ 0,  0, -1,  0,  0,  1,  0,  0,  0],
       [-1,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 0, -1,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0, -1,  0,  0,  0,  0,  0,  1]])
```

Results:

Plots obtained by running the robot is square and octagonal path are shown below:
Yellow is the trajectory of the robot and the blue dots represent the landmarks.





As seen from the plots, the map estimated by the octagonal path is more accurate than the one estimated by the square path. This is because the robot views the AprilTags from much more angles in the octagonal path than the square and results in better updates and hence better final estimation of the landmark poses.

Followings commands needs to be executed in order to mimic the results shown in the video.

1. source devel/setup.bash
2. roslaunch rb5_vision rb_camera_main_ocv.launch
3. rosrun april_detection april_detection_node

4. rosrun rb5_control hw2_mpi_control_node.py
5. rosrun rb5_control slam.py
6. rosrun rb5_control hw_sol.py