Robotics: Capstone
Week 5: An Extended Kalman Filter for State Estimation

1 Introduction

For this project, you will need to implement Kalman filter to create a robust controller for your robot. Here, we assume that the measurements for the robot location using the Apriltag markers are noisy, so you need to fuse them with your estimate for the state of the robot, to be able to deal with the uncertainty that there is both in the measurements and in the estimate of your state.

2 Kalman Filtering

Because of the nature of your system, we will need to use the Extended Kalman Filter to deal with the nonlinearities in the system equations. For a full derivation of the equations, please refer to the lecture slides.

3 Transformations for the AprilTags

When dealing with transformations with only a rotation and translation (rigid transformations), such as the transformation between the world and the robot, it is convenient to express it as a single matrix:

$$H(x,y,\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{pmatrix}$$

When we receive an AprilTag measurement, it is in the robot frame, so we need to perform a transformation to get the robot's pose in the world frame. To do this, we can use the pose of the tag in the world frame, as well as the pose of the tag in the robot frame. If the tag in the world frame is at (x_w, y_w, θ_w) and the AprilTag is seen relative to the robot at (x_r, y_r, θ_r), the new transformation is given by:

$$^{world}H_{robot} = H(x_w,y_w,\theta_w)H(x_r,y_r,\theta_r)^{-1} = \begin{pmatrix} \cos(\theta_w) & -\sin(\theta_w) & x_w \\ \sin(\theta_w) & \cos(\theta_w) & y_w \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta_r) & -\sin(\theta_r) & x_r \\ \sin(\theta_r) & \cos(\theta_r) & y_r \\ 0 & 0 & 1 \end{pmatrix}^{-1}$$

Given a matrix in the form $H(x,y,\theta)$ you can extract the x, y, θ using $x = H_{0,2}$, $y = H_{1,2}$, and $\theta = \text{atan2}(H_{1,0}, H_{0,0})$ (0 indexed).

4 Interface

Now however you won't be using the measurements directly but pass them through a filter first. At a high level, your goal is to implement the Kalman Filter that fuses the localization information from the Apriltag markers and the estimate of your state. We use `self.x_t` to denote your state and `self.P_t` to denote the covariance estimate (it should be helpful to use `self` to keep track of your state thoughout your code, similarly to a global variable). We provide the skeleton code for the filter in the class `KalmanFilter`. You will need to implement the following functions:

1. `self.__init__(markers)`: You will need to initialize your state variables here, as well as tune the measurement covariances Q and R.

2. `self.step_filter(v,imu_meas,z_t)`: This is the function that will be called from `process_measurements` in the `RobotControl` class. It takes in the commanded velocity `v`, the IMU measurement `imu_meas` (of which you will only use the gyroscope measurements for your prediction step)

3. `self.prediction(v, imu_meas)`: Implement the prediction step on your current estimate of your state and covariance. It takes in as input your commanded velocity `v` and the IMU readings `imu_meas` (from `get_imu`) of which you will just use the `time` and `omega` fields.

4. `self.update(z_t) -` This implements the update step of the Kalman Filter, taking in the `z_t` from `get_measurements` function.