EECS C106B: Project 3 - State Estimation *

Due Date: March 22nd at 11:59pm

Goal

In this project, we will experiment with three classical state estimation methods: dead reckoning, Kalman filter, and extended Kalman filter. The purpose of the project is to implement the three estimation algorithms, as detailed below:

- In part A, you will implement a dead reckoning method to estimate the trajectory of a turtlebot and a planar quadrotor in simulation.
- In part B, you will implement a Kalman filter to estimate the trajectory of a turtlebot in simulation.
- In part C, you will implement an extended Kalman filter to estimate the trajectory of a planar quadrotor in simulation.
- Optionally, you will get 10 bonus points if you manage to successfully implement the extended Kalman filter on a real turtlebot in the lab, regardless of you being an undergraduate or graduate student. If available, you will receive bonus points for implementing the extended Kalman filter on a real quadrotor – details for this will be released after spring break.

Contents

*Developed by Fangyu Wu, Spring 2023. Adapted for Spring 2024 by Nima Rahmanian and Karim El-Refai

1 Theory

Recall from the lectures that estimation refers to the problem of identifying the state of a system using sensor measurements and a prior model of the system to be estimated. In the context of robotics, when the state to be estimated is the location of the robot, the problem is called localization; when the state is the location of landmarks in the environment, the problem is called mapping; when the state includes both locations of the robot and landmarks, the problem is called simultaneous localization and mapping (SLAM). In this project, we will focus exclusively on the problem of localization.

1.1 Problem Formulation

In a state estimation problem, we have the state vector $\mathbf{x} \in \mathbb{R}^n$ that we want to estimate, the input vector $\mathbf{u} \in \mathbb{R}^m$, the output vector $\mathbf{y} \in \mathbb{R}^p$, the dynamics model $g : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ of the system with a process noise $\mathbf{w} \in \mathbb{R}^n$, and a measurement model $h : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^p$ of the sensors with a measurement noise $\mathbf{v} \in \mathbb{R}^p$.

In this project, we only consider discrete-time systems. As such, the system under consideration may be written as follows:

$$\begin{aligned} \mathbf{x}[0] &= \mathbf{x}_0, \\ \mathbf{x}[t+1] &= g(\mathbf{x}[t], \mathbf{u}[t], \mathbf{w}[t]), \\ \mathbf{y}[t] &= h(\mathbf{x}[t], \mathbf{v}[t]), \end{aligned} \tag{1}$$

for some initial state $\mathbf{x}_0$ and $t \geq 0$.

Provided with a sequence of outputs $(\mathbf{y}[0], \ldots, \mathbf{y}[t])$ and sequence of inputs $(\mathbf{u}[0], \ldots, \mathbf{u}[t-1])$, we want to find the sequence of state *estimates* $(\hat{\mathbf{x}}[0], \ldots, \hat{\mathbf{x}}[t])$ such that it is as close to the true state sequence $(\mathbf{x}[0], \ldots, \mathbf{x}[t])$ as possible. Note that here we use the hat notation, $\hat{\cdot}$, to distinguish an estimate of the state $\hat{\mathbf{x}}$ from the true state $\mathbf{x}$. Ideally, we want $|\mathbf{x} - \hat{\mathbf{x}}| = 0$.

1.2 Unicycle Model

We define the unicycle model as per the Equation 13.14 of [2]:

$$\begin{aligned} \mathbf{x} &\coloneqq \begin{bmatrix} \phi \\ x \\ y \\ \theta_{\mathrm{L}} \\ \theta_{\mathrm{R}} \end{bmatrix}, \quad \mathbf{u} \coloneqq \begin{bmatrix} u_{\mathrm{L}} \\ u_{\mathrm{R}} \end{bmatrix}, \\ \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) &\coloneqq \begin{bmatrix} -\frac{r}{2d} & \frac{r}{2d} \\ \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_{\mathrm{L}} \\ u_{\mathrm{R}} \end{bmatrix}, \end{aligned} \tag{2}$$

where r is the radius of the robot wheels, $2d$ is the wheel base, ϕ is robot orientation, x and y are robot 2D location, θ_{L} and θ_{R} are the rolling angles of the left and right wheels, and u_{L} and u_{R} are the rotation speeds of the left and right wheels. The unicycle model is illustrated in Figure 1.

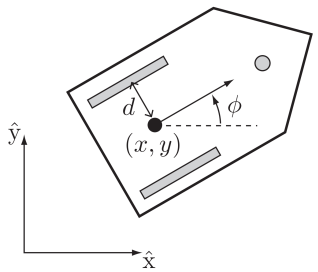

Figure 1: Illustration of an unicycle robot [2].

Note that dynamics model $f(\cdot)$ defined in Equation (2) is in continuous time. In order to perform state estimation, we will need to approximate the continuous-time dynamics model with a discrete-time model. In this project, we use

the following surrogate discrete-time model based on forward Euler method:

$$\mathbf{x}[t+1] = \mathbf{x}[t] + f(\mathbf{x}[t], \mathbf{u}[t]) \cdot \Delta t, \tag{3}$$

where $\Delta t > 0$ is some small sampling time interval.

1.3 Planar Quadrotor Model

In this section, we introduce the planar quadrotor dynamics. This work is heavily influenced by the work of Venkatesh, Vadhvana, and Jain in Analysis and Control of a Planar Quadrotor.

Below is a figure of a planar quadrotor viewed from the XY plane. Note: in our analysis we'll be analyzing a planar quadrotor that lives on the **XZ plane**, where $y = 0$ in order to properly represent it living in $\mathbb{R}^3$, but everything else remains the same.

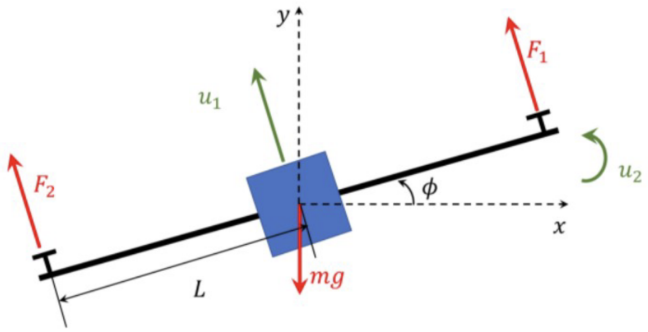

Figure 2: Planar Quadrotor Model

We say that the thrust force exerted by the motors on a side of a quadrotor, F_i, is equivalent to the angular velocity of the rotors on that side w_i times a scaling factor k_i. Giving us:

$$F_i = k_i \omega_i^2 \tag{4}$$

We now define u_1 to be the total thrust force of the quadrotor and u_2 to be the net moment of the quadrotor

$$u_1 = F_1 + F_2 \tag{5}$$

$$u_2 = \frac{L}{2}(F_1 - F_2) \tag{6}$$

We can use Newton's 2nd Law of Motion in the X and Z directions and analyzing the torque around the center of mass to get:

$$\ddot{x} = \frac{-u_1 \sin\phi}{m} \tag{7}$$

$$\ddot{z} = -g + \frac{u_1 \cos\phi}{m} \tag{8}$$

$$\ddot{\phi} = \frac{u_2}{J} \tag{9}$$

Note: J is the moment of inertia for the quadrotor. Finally, we can compactly summarize these equations as follows:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{\phi} \\ \ddot{x} \\ \ddot{z} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{\phi} \\ 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{-\sin(\phi)}{m} & 0 \\ \frac{\cos(\phi)}{m} & 0 \\ 0 & \frac{1}{J} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = f(\mathbf{x}, \mathbf{u}) \tag{10}$$

We can use the same surrogate discrete-time model as introduced in the previous section for the planar quadrotor:

$$\mathbf{x}[t+1] = \mathbf{x}[t] + f(\mathbf{x}[t], \mathbf{u}[t]) \cdot \Delta t, \tag{11}$$

1.4 Dead Reckoning

We begin with dead reckoning method, a classical localization method that dates back to the early ages of navigation. The term dead reckoning refers to the method of using only the dynamics model, the input sequence, and the initial state to estimate the state evolution of the system. The dynamics model can be written as

$$g(\mathbf{x}, \mathbf{u}) = \mathbf{x} + f(\mathbf{x}, \mathbf{u}) \cdot \Delta t, \tag{12}$$

with process noise $\mathbf{w}$ assumed to be zero. Measurement model h is irrelevant here, since it does not use any sensor measurements. The algorithm of dead reckoning method is described in Algorithm 1.

Algorithm 1 Dead Reckoning

Inputs: $\mathbf{x}_0, (\mathbf{u}[0], \ldots, \mathbf{u}[T-1]), \Delta t, T$
Outputs: $\hat{\mathbf{x}}[0], \ldots, \hat{\mathbf{x}}[T]$

$t \leftarrow 0$
$\hat{\mathbf{x}}[t] \leftarrow \mathbf{x}_0$
while $t \leq T-1$ **do**
  $\hat{\mathbf{x}}[t+1] \leftarrow g(\hat{\mathbf{x}}[t], \mathbf{u}[t])$
  $t \leftarrow t+1$
end while
return $\hat{\mathbf{x}}[t]$ for $t = 0, \ldots, T$

The key assumption of dead reckoning method is that the process noise $\mathbf{w}$ remains identically zero throughout time. However, this is hardly true in real world. With time, the errors due to the nonzero noise $\mathbf{w}$ accumulate, eventually rendering dead reckoning estimates too erroneous to be useful.

This error accumulation problem is analogous to the robustness issue of feedforward control. Just like what we have done in PID control, we can improve the robustness issue using the idea of *feedback*. In the context of estimation theory, feedback amounts to using sensor measurements to improve estimation accuracy. To this end, in the next section we introduce the celebrated Kalman filter.

1.5 Kalman Filter

The drawback of dead reckoning leads us to Kalman filter, an early hallmark of estimation theory developed by Kalman and others in 1960. Unlike dead reckoning, Kalman filter attempts to correct error accumulation by using sensor measurements.

Kalman filter is known as the *optimal* estimator for linear systems with additative Gaussian noises, as elaborated in Chapter 3.3.4 of [1]. Specifically, dynamics model $g(\cdot)$ and measurement model $h(\cdot)$ should be linear. Process noise and measurement noise should be normally distributed, that is, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{Q}), \mathbf{v} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{R})$ for some covariance matrices $\boldsymbol{Q} \in \mathbb{R}^{n \times n}, \boldsymbol{R} \in \mathbb{R}^{p \times p}$. Note that we always assume that $\mathbf{w}$ and $\mathbf{v}$ have zero means because any nonzero means in the noise terms should be considered as part of the models.

Unfortunately, we cannot apply Kalman filter to our unicycle robot in the most general way because the unicycle model (2) is nonlinear. For learning purpose, we assume in this part of the project that the robot heading ϕ is fixed at $\pi/4$, i.e. $\phi \coloneqq \pi/4$. This renders our system dynamics linear. Furthermore, let's assume that we have some noisy measurements on x and y. Consequently, we can write the dynamics model and measurement model can as follows:

$$\mathbf{x} \coloneqq \begin{bmatrix} x \\ y \\ \theta_{\mathrm{L}} \\ \theta_{\mathrm{R}} \end{bmatrix}, \quad \mathbf{u} \coloneqq \begin{bmatrix} u_{\mathrm{L}} \\ u_{\mathrm{R}} \end{bmatrix},$$

with dynamics model

$$\mathbf{x}[t+1] = g(\mathbf{x}[t], \mathbf{u}[t], \mathbf{w}[t]) \coloneqq \boldsymbol{A}\mathbf{x}[t] + \boldsymbol{B}\mathbf{u}[t] + \mathbf{w}[t] = \boldsymbol{I} \cdot \mathbf{x}[t] + \begin{bmatrix} \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \Delta t \cdot \mathbf{u}[t] + \mathbf{w}[t], \tag{13}$$

Algorithm 2 Kalman Filter

Inputs: $\mathbf{x}_0, (\mathbf{u}[0], \ldots, \mathbf{u}[T-1]), (\mathbf{y}[0], \ldots, \mathbf{y}[T]), \Delta t, T, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{P}_0$
Outputs: $\hat{\mathbf{x}}[0], \ldots, \hat{\mathbf{x}}[T]$

$t \leftarrow 0$
$\hat{\mathbf{x}}[t] \leftarrow \mathbf{x}_0$
$\boldsymbol{P}[t] \leftarrow \boldsymbol{P}_0$
while $t \leq T-1$ **do**
  $\hat{\mathbf{x}}[t+1 \mid t] \leftarrow \boldsymbol{A}\hat{\mathbf{x}}[t] + \boldsymbol{B}\mathbf{u}[t]$ (state extrapolation)
  $\boldsymbol{P}[t+1 \mid t] \leftarrow \boldsymbol{A}\boldsymbol{P}[t]\boldsymbol{A}^\top + \boldsymbol{Q}$ (covariance extrapolation)
  $\boldsymbol{K}[t+1] \leftarrow \boldsymbol{P}[t+1 \mid t]\boldsymbol{C}^\top \left(\boldsymbol{C}\boldsymbol{P}[t+1 \mid t]\boldsymbol{C}^\top + \boldsymbol{R}\right)^{-1}$ (Kalman gain)
  $\hat{\mathbf{x}}[t+1] \leftarrow \hat{\mathbf{x}}[t+1 \mid t] + \boldsymbol{K}[t+1]\left(\mathbf{y}[t+1] - \boldsymbol{C}\hat{\mathbf{x}}[t+1 \mid t]\right)$ (state update)
  $\boldsymbol{P}[t+1] \leftarrow (\boldsymbol{I} - \boldsymbol{K}[t+1]\boldsymbol{C})\boldsymbol{P}[t+1 \mid t]$ (covariance update)
  $t \leftarrow t+1$
end while
return $\hat{\mathbf{x}}[t]$ for $t = 0, \ldots, T$

and measurement model

$$\mathbf{y}[t] = h(\mathbf{x}[t], \mathbf{v}[t]) \coloneqq \boldsymbol{C}\mathbf{x}[t] + \mathbf{v}[t] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{x}[t] + \mathbf{v}[t]. \tag{14}$$

With the dynamics model and measurement model defined above, we are ready to present the Kalman filter algorithm in Algorithm 2. For a classical probablistic derivation of the method, please refer to Chapter 3.3.3 of [1]. Note that covariance matrices $\boldsymbol{Q}, \boldsymbol{R}$ and $\boldsymbol{P}_0$ are given as additional inputs of the algorithm. Note that $\boldsymbol{Q}, \boldsymbol{R}$ and $\boldsymbol{P}$ are often written as $\boldsymbol{\Sigma}_{\mathbf{w}}, \boldsymbol{\Sigma}_{\mathbf{v}}$, and $\boldsymbol{\Sigma}$, respectively, as you may have seen in lectures and homeworks. You will need to search for a combination of $\boldsymbol{Q}, \boldsymbol{R}$ and $\boldsymbol{P}_0$ that produces accurate state estimation.

As discussed before, Kalman filter only deals with systems of linear dynamics and measurement models. As most of the robotic applications are nonlinear, such as our unicycle robots, the original form of Kalman filter is quite limited. Next, we introduce extended Kalman filter, an *extension* to Kalman filter that can be applied to nonlinear systems.

1.6 Extended Kalman Filter

Shortly after the development of Kalman filter, scientists at NASA developed extended Kalman filter (EKF) to solve nonlinear estimation problems for controlling rockets. EKF deals with nonlinearity in the systems by means of first-order approximation, i.e., linearization.

Recall that the planar quadrotor dynamics (15) paired with (10) are nonlinear.

$$g(\mathbf{x}, \mathbf{u}) = \mathbf{x} + f(\mathbf{x}, \mathbf{u}) \cdot \Delta t. \tag{15}$$

That is, if we commanded a zero control input, the system would respond in a non-zero manner (the quadrotor would accelerate downwards due to gravity). This violates the zero-input zero-output property of linear systems. Linearizing system (15) around some point $(\mathbf{x}^*, \mathbf{u}^*)$, we have

$$\begin{aligned} g(\mathbf{x}, \mathbf{u}) &= g(\mathbf{x}^*, \mathbf{u}^*) + \left.\frac{\partial g}{\partial \mathbf{x}}\right|_{(\mathbf{x}^*, \mathbf{u}^*)} (\mathbf{x} - \mathbf{x}^*) + \left.\frac{\partial g}{\partial \mathbf{u}}\right|_{(\mathbf{x}^*, \mathbf{u}^*)} (\mathbf{u} - \mathbf{u}^*) + \text{H.O.T.} \\ &\approx \bar{\boldsymbol{A}}(\mathbf{x}^*, \mathbf{u}^*)\mathbf{x} + \bar{\boldsymbol{B}}(\mathbf{x}^*, \mathbf{u}^*)\mathbf{u} + \boldsymbol{E}(\mathbf{x}^*, \mathbf{u}^*), \end{aligned}$$

where

$$\bar{\boldsymbol{A}}(\mathbf{x}^*, \mathbf{u}^*) \coloneqq \left.\frac{\partial g}{\partial \mathbf{x}}\right|_{(\mathbf{x}^*, \mathbf{u}^*)}, \quad \bar{\boldsymbol{B}}(\mathbf{x}^*, \mathbf{u}^*) \coloneqq \left.\frac{\partial g}{\partial \mathbf{u}}\right|_{(\mathbf{x}^*, \mathbf{u}^*)}, \quad \boldsymbol{E}(\mathbf{x}^*, \mathbf{u}^*) \coloneqq g(\mathbf{x}^*, \mathbf{u}^*) - \bar{\boldsymbol{A}}(\mathbf{x}^*, \mathbf{u}^*)\mathbf{x}^* - \bar{\boldsymbol{B}}(\mathbf{x}^*, \mathbf{u}^*)\mathbf{u}^*.$$

Because the nonlinear dynamics $g(\cdot)$ behaves similarly to the linear system $\bar{\boldsymbol{A}}(\mathbf{x}^*, \mathbf{u}^*)\mathbf{x} + \bar{\boldsymbol{B}}(\mathbf{x}^*, \mathbf{u}^*)\mathbf{u} + \boldsymbol{E}(\mathbf{x}^*, \mathbf{u}^*)$ near the point $(\mathbf{x}^*, \mathbf{u}^*)$, we can use the surrogate linear system to approximate the original nonlinear system.

With the power of linearization, we need not limit ourselves to linear measurement models (as seen in the case of the turtlebot with Equation (14)). Let us try the following nonlinear measurement model

$$y = h(\mathbf{x}) = \begin{bmatrix} \sqrt{(\ell_x - x)^2 + \ell_y^2 + (\ell_z - z)^2} \\ \phi \end{bmatrix} \tag{16}$$

where $(\ell_x, \ell_y, \ell_z) = (0, 5, 5)$ is the landmark position and ϕ is the measured rotation of the landmark's vertical line as illustrates in Figure 3 and Figure 4 below:

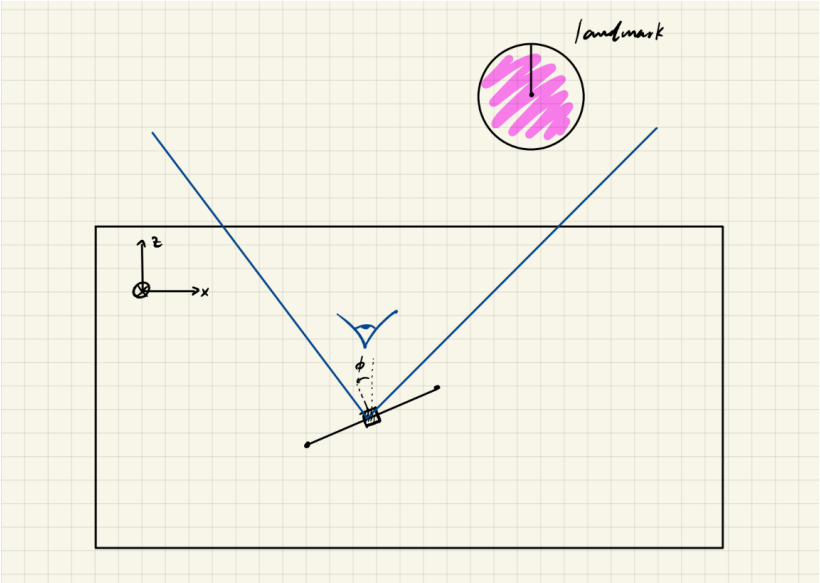

Figure 3: Environment Overview

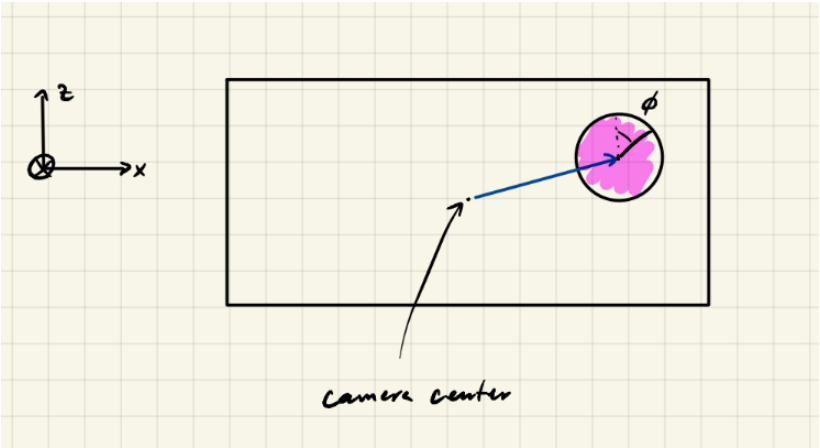

Figure 4: Planar Quadrotor Camera View

Once again, we can perform first-order approximation to the nonlinear measurement model $h(\cdot)$ to obtain the linear surrogate model at any given point. Integrating linearization with original Kalman filter algorithm, we obtain the EKF algorithm, as described in Algorithm 3.

Note that the only difference between Kalman filter and EKF is the introduction of dynamics linearization and measurement linearization steps.

Unlike Kalman filter, which is the provably optimal estimator for linear systems with Gaussian noises, EKF unfortunately has no optimality guarantee. In practice, people find it works well for a wide range of applications, but there is no guarantee that it will work well for your problem. To address the shortcomings of Kalman filter and EKF, people have proposed many other estimation methods, such as the well-known particle filter and receding horizon estimation. A complete treatment of these advanced methods is beyond the scope of this project. If you are interested in those topics, we recommend you to check out Chapter 4 of [1] and Chapter 4 of [3].

Algorithm 3 Extended Kalman Filter

Inputs: $\mathbf{x}_0, (\mathbf{u}[0], \ldots, \mathbf{u}[T-1]), (\mathbf{y}[0], \ldots, \mathbf{y}[T]), \Delta t, T, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{P}_0$
Outputs: $\hat{\mathbf{x}}[0], \ldots, \hat{\mathbf{x}}[T]$

$t \leftarrow 0$
$\hat{\mathbf{x}}[t] \leftarrow \mathbf{x}_0$
$\boldsymbol{P}[t] \leftarrow \boldsymbol{P}_0$
while $t \leq T-1$ **do**
  $\hat{\mathbf{x}}[t+1 \mid t] \leftarrow g(\hat{\mathbf{x}}[t], \mathbf{u}[t])$ (state extrapolation)
  $\boldsymbol{A}[t+1] \leftarrow \frac{\partial g}{\partial \mathbf{x}}\Big|_{(\hat{\mathbf{x}}[t], \mathbf{u}[t])}$ (dynamics linearization)
  $\boldsymbol{P}[t+1 \mid t] \leftarrow \boldsymbol{A}[t+1]\boldsymbol{P}[t]\boldsymbol{A}^\top[t+1] + \boldsymbol{Q}$ (covariance extrapolation)
  $\boldsymbol{C}[t+1] \leftarrow \frac{\partial h}{\partial \mathbf{x}}\big|_{\hat{\mathbf{x}}[t+1|t]}$ (measurement linearization)
  $\boldsymbol{K}[t+1] \leftarrow \boldsymbol{P}[t+1 \mid t]\boldsymbol{C}[t+1]^\top \left(\boldsymbol{C}[t+1]\boldsymbol{P}[t+1 \mid t]\boldsymbol{C}[t+1]^\top + \boldsymbol{R}\right)^{-1}$ (Kalman gain)
  $\hat{\mathbf{x}}[t+1] \leftarrow \hat{\mathbf{x}}[t+1 \mid t] + \boldsymbol{K}[t+1]\left(\mathbf{y}[t+1] - h(\hat{\mathbf{x}}[t+1 \mid t])\right)$ (state update)
  $\boldsymbol{P}[t+1] \leftarrow (\boldsymbol{I} - \boldsymbol{K}[t+1]\boldsymbol{C}[t+1])\boldsymbol{P}[t+1 \mid t]$ (covariance update)
  $t \leftarrow t+1$
end while
return $\hat{\mathbf{x}}[t]$ for $t = 0, \ldots, T$

2 Project Tasks

In this project, you will implement the three estimation algorithms: dead reckoning, Kalman filter, and EKF. For the unicycle model, you will implement the dead reckoning and Kalman filter algorithms. For the planar quadrotor model, you will implement the dead reckoning and Extended Kalman filter algorithms. We recommend first completing the tasks involving the turtlebot and later working with the planar quadrotor. In the starter code, you are given an unicycle simulator and a planar quadrotor simulator. The simulator code is obfuscated to emulate the real-world setting. Your job is to implement the three algorithms using only standard python libraries, i.e., numpy, sympy, and matplotlib. We provide you with $\mathbf{x}_0, (\mathbf{u}[0], \ldots, \mathbf{u}[T-1]), (\mathbf{y}[0], \ldots, \mathbf{y}[T]), \Delta t, T$, but you will need to calculate the $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{C}$. For Kalman filter and EKF, you will also need to fine tune $\boldsymbol{Q}, \boldsymbol{R}$, and $\boldsymbol{P}_0$.

All final results with your estimators **must** be collected with data that has **both** process and measurement noise. We've provided you with noisy data for the quadrotor simulator and the turtlebot simulator will generate it's own noisy data by default.

3 Deliverables

As before, we expect you to deliver a report on your findings to desmonstrate your undestanding in the subjects covered in this project. The purpose of these project reports are to prepare you for conference-style paper writing. Please format your report using the IEEE two-column conference template. In your report, make sure you have included the following content.

1. Abstract: An abstract, of at most 150 words, describing what you did and what results you achieved.
2. Methods: Explain in detail your approach towards each of the three estimators, making sure to address the following questions.
   (a) For dead reckoning, what is the theoretical motivation behind Equation (12) from the perspective of ordinary differential equations? Can we do better than that? ($\sim$ 1 column)
   (b) For Kalman filter, what is the physical intuition behind $\boldsymbol{Q}, \boldsymbol{R}$ and $\boldsymbol{P}_0$? How did you fine tune those parameters? Would you do this differently if you were to do this on a real robot without access to ground truth? ($\sim$ 2 columns)
   (c) For EKF, which technique do we use to address the problem of nonlinearity? What is the limitation of that technique? Can you mitigate the observed limitation?
   Bonus: If you have implemented EKF on a real turtlebot, describe your process model and measurement model and comment on how did you acquire group truth. ($\sim$ 1-2 columns)
3. Experimental Results: Summarize the performance of each of the estimators on each scenario, using both words and figures. ($\sim$ 1 column, plus figures)

(a) Summarize the performance of the three algorithms in a table. In the table, make sure to include quantitative measurements on estimation accuracy and per-step computational running time.

(b) For each estimation task, provide a set of plots of the estimates $\hat{\mathbf{x}}$ compared with the ground truth $\mathbf{x}$.

(c) For each estimation task, provide of a plot of the estimated trajectory compare with the ground truth trajectory.

(d) **Bonus**: If you have implemented EKF on a real turtlebot, provide a plot of the estimated trajectory overlaid with the "true" trajectory.

Remember that each table or figure should come with a detailed caption. An informed reader should be able to understand the table or figure without having to read the paper at all.

4. Discussion: ($\sim$ 1-2 columns)

   (a) Discuss the performance of each estimator. What worked, what didn't, and why?

   (b) Compare and contrast the performance of the estimators against one another. When would you use each type of estimator? How might you improve them?

   (c) **Bonus**: If you have implemented EKF on a real turtlebot, discuss what lessons have you learned from doing it in the real-world setting.

5. Bibliography: A bibliography section citing any resources you used. This should include any resources you used from outside this class. Please use the IEEE citation format.

6. Appendix:

   (a) GitHub Link: Provide the link to your GitHub classroom repo. Simply push your code to the private repository that was created for your team, and we will be able to see any changes you push to your assignment repository.

   (b) **Bonus**: What do you think the learning goals of this assignment are? How effective was this assignment at fostering those learning goals for you? How can the lab documentation and starter code be improved? We are especially looking for any comments you have regarding the pedagogical success of the assignment.

7. Page Limit: Reports are **limited to 6 pages**, not including the bibliography or appendix.

4 Getting Started

Create a new ROS workspace to store your code for this project. Remember to build and source your workspace after you download the starter code.

4.1 GitHub Classroom

For projects in this class, we will be using GitHub Classroom. To access the starter code, simply head over to the Project 3 assignment page to accept the assignment. If you are asked to associate yourself with a school identifier, just click "Skip to the next step."

You should now be asked to create a team or join from a list of existing teams. If one of your teammates has already created a team, you should join that team instead of creating a new one. After you have joined a team, you will not be able to switch teams by yourself. If you make a mistake or something else comes up that requires you to switch teams, let us know.

Your team should now have a private project repository located at https://github.com/ucb-ee106-classrooms/project-3-your-team-name.

4.2 Pulling Starter Code

If there are any updates to the starter code that you wish to pull you may do so with the commands.

```
cd your-repo
git remote add public https://github.com/ucb-ee106/proj3_pkg.git
git pull public main
```

4.3 Setup environment

You can install all the required packages by running the following command.

```
pip install -r -U requirements.txt
```

4.4 Running Code

The code for the turtlebot is located in `turtlebot_proj3_pkg` and needs to be ran in ROS. Note: You only need to implement dead reckoning and the Kalman filter. The commands to run each are found in the associated `README.md`.

Note: You only need to implement dead reckoning and the Kalman filter. The commands to run each are found in the associated `README.md`.

The code for the quadrotor is located in `drone_proj3` and requires no ROS. You will be editing `drone_estimator.py`. To see how well your estimator is doing, simply run

```
python drone_estimator_node.py --estimator <your estimator here>
```

We have provided you with two datasets `data.npy` (no process nor measurement noise) and `noisy_data.npy` (both process and measurement noise). We expect your drone results to be shown using `noisy_data.npy`. To visualize the quadrotor without noise you can run

```
python visualize_test_case.py
```

If you want to generate your own noisy data you can modify `environment.py` but be careful with what you edit!

5 Scoring

The project is scored out of 85 points with 10 additional points awarded if you have completed the bonus task.

Table 1: Point Allocation for Project 3

Section	**Points**
Page Limit	5
Figures: Quality and Readability	5
Abstract	5
Methods: Dead Reckoning	10
Methods: Kalman Filter	10
Methods: Extended Kalman Filter	10
Results	20
Discussion	10
Bibliography	5
Code	5
Total	85
Bonus*	10

6 Submission

You will submit your writeup on Gradescope and push your code to GitHub Classroom. Only one submission is needed per group. Please add your group mates as collaborators on both GitHub and Gradescope. Your code will be checked for plagiarism, so please submit your own work with proper citations.

You can use a repo from this and add your teammates here: https://classroom.github.com/a/53uRCrYS. **If you already made one in Project 4, you may use the SAME repo for Project 3 (starter code for both projects should be accessible through this link).**

7 Improvements

If you notice any typos or things in this document or the starter code which you think we should change, please let us know. Next years' students will thank you.

References

[1] Timothy D Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.

[2] Kevin M Lynch and Frank C Park. *Modern robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[3] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model Predictive Control: Theory, Computation, and Design*, volume 2. Nob Hill Publishing Madison, WI, 2017.