ECE-225A Final Project: State Estimation in Robotics Using Kalman Filtering

Nathan Cusson-Nadeau

I. Introduction

In the field of robotics and navigation, determining the state of a system is a problem of paramount importance. Using sensors, a system can get information about its environment and, with the proper ones, can get a very good estimate of its current state. For instance, using an inertial measurement unit (IMU) in a plane can allow the plane to determine its changes in orientation or attitude. However, sensors are not perfect and a subject to noisy measurements, and thus mean that the state is a random variable. As such, if a system wishes to get a near-perfect estimate of its current state, it is necessary to correct for noise using probabilisitc means. There are a plethora of techniques in order to solve this problem, but one of the most commonly used is a probabilistic technique called the Kalman Filter. Here, I plan to explain the theory behind the Kalman Filter, limitations, and ways of circumventing the limitations then to demonstrate the technique on a toy robot example.

II. Theory

There are a few flavors of the Kalman Filter. The original Kalman Filter can only be employed on *linear* systems. To *extend* the concept to nonlinear ones it is necessary to use the extended Kalman Filter (EKF). Whatever the choice, every Kalman Filter is based on Bayesian Filtering.

A. Baysesian Filtering

To derive a Kalman Filter requires understanding the Bayes Filter. A Bayes Filter in continuous time uses a system's dynamics (in robotic settings also referred to as the motion model) $f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t)$ and observation model $h(\mathbf{x}_t, \mathbf{v}_t)$ in the presence of process and measurement noise to make predictions and update the state of a system. Where $\mathbf{x}_t \in \mathbb{R}^n$ is a vector of the n different states of interest in the system:

$$\mathbf{x}_t = \begin{bmatrix} x_t^1 \\ x_t^2 \\ \vdots \\ x_t^n \end{bmatrix} \tag{1}$$

and $\mathbf{u}_t \in \mathbb{R}^p$ are the control inputs (i.e. the commands given to the system):

$$\mathbf{u}_t = \begin{bmatrix} u_t^1 \\ u_t^2 \\ \vdots \\ u_t^p \end{bmatrix} \tag{2}$$

and $\mathbf{w}_t \in \mathbb{R}^n$ and $\mathbf{v}_t \in \mathbb{R}^n$ are the process and observation noises respectively.

The motion model maps the current state and control knowledge to the next state $\mathbf{x}_{t+1}$. Due the uncertainty from process noise it can be modeled as a probability density function (pdf):

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \sim p_f(\cdot|\mathbf{x}_t, \mathbf{u}_t) \tag{3}$$

Similarly, the observation model can also be modeled as a pdf:

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{v}_t) \sim p_h(\cdot|\mathbf{x}_t) \tag{4}$$

The Bayes Filter keeps track of two pdfs through time, the predicted state distribution:

$$p_{t+1|t}(\mathbf{x}) = \int p_f(\mathbf{x}|\mathbf{s}, \mathbf{u}_t) p_{t|t}(\mathbf{s} d\mathbf{s} \tag{5}$$

and the updated state distribution:

$$p_{t+1|t+1}(\mathbf{x}) = \frac{p_h(\mathbf{z}_{t+1}|\mathbf{x}) p_{t+1|t}(\mathbf{x})}{\int p_h(\mathbf{z}_{t+1}|\mathbf{s}) p_{t+1|t}(\mathbf{s}) d\mathbf{s}} \tag{6}$$

A prior distribution of the state is also necessary:

$$p_{t|t}(\mathbf{x}) := p(\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}) \tag{7}$$

B. Kalman Filter

A Kalman filter is a special case of a Bayes Filter and is based on the following assumptions:

1) The prior pdf $p_{t|t}$ is a Gaussian/Normal distribution.
$$\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1} \sim \mathcal{N}(\mu_{t|t}, \Sigma_{t|t}) \tag{8}$$
where $\mu_{t|t}$ is the expected value of the prior state and $\Sigma_{t|t} \in \mathbb{R}^{nxn}$ is its covariance matrix.
2) The motion model is linear in the state $\mathbf{x}_t$ with additive Gaussian noise $\mathbf{w}_t \sim \mathcal{N}(0, W)$ with covariance matrix W.
3) Similarly the observation model is linear in the state $\mathbf{x}_t$ with additive Gaussian noise $\mathbf{v}_t \sim \mathcal{N}(0, V)$ with covariance matrix V.
4) The motion noise $\mathbf{w}_t$ and observation noise $\mathbf{v}_t$ are independent of each other and the state across time. $(\mathbf{w}_t \perp \mathbf{v}_t \perp \mathbf{x}_t)$

In most engineering applications, it is fair to assume that process and sensor noise can be modeled by additive independent random variables distributed normally, thus assumption 1 holds.

However, in practicality assumptions 2 and 3 tend not to be true as most systems (especially robotic ones), have nonlinear motion and observation models. Therefore, the predicted and updated pdfs are no longer Gaussian and cannot be evaluated in a closed form. By using moment matching, these pdfs can be forced to be Gaussian through evaluating their first and second moments and approximating them with Gaussians with the same moments.

In applying this transformation to the prediction and update pdfs there are two common approaches:

1) Taylor series approximations to the motion and observation models around the state and noise means
2) Using a finite set of sigma points to approximate the prior Gaussian pdfs

if approach 1 is taken, this results in a nonlinear Kalman Filter of called the **Extended Kalman Filter (EKF)**. If using approach 2, this is called the Unscented Kalman Filter (UKF). For this project, I elected to use the results of the Extended Kalman Filter derivation. Of note, the UKF can give better results than the EKF but, as will be demonstrated later, the EKF still does an excellent job of estimating the states. Lastly, due to the linearization required in the motion and observation models, the state estimate will no longer be optimal but very closely approximate the true state.

C. Extended Kalman Filter

In applying approach 1, we arrive at the following suite of equations:

Prior:

$$\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1} \sim \mathcal{N}(\mu_{t|t}, \Sigma_{t|t}) \tag{9}$$

Motion Model:

$$F_t := \frac{\partial f}{\partial \boldsymbol{x}_t}(\mu_{\mathbf{t}|\mathbf{t}}, \mathbf{u}_t, \mathbf{0}) \tag{10}$$

$$Q_t := \frac{df}{d\mathbf{w}}(\mu_{t|t}, \mathbf{u}_t, \mathbf{0}) \tag{11}$$

Observation Model:

$$H_t := \frac{\partial h}{\partial \boldsymbol{x}_t}(\mu_{t|t-1}, \mathbf{0}) \tag{12}$$

$$R_t := \frac{\partial h}{\partial \boldsymbol{v}_t}(\mu_{t|t-1}, \mathbf{0}) \tag{13}$$

Prediction:

$$\mu_{\mathbf{t+1|t}} = f(\mu_{t|t}, \mathbf{u}_t, \mathbf{0}) \tag{14}$$

$$\boldsymbol{\Sigma}_{t+1|t} = F_t \Sigma_{t|t} F_t^T + Q_t W Q_t^T \tag{15}$$

Update:

$$\boldsymbol{\mu}_{t+1|t+1} = \boldsymbol{\mu}_{t+1|t} + K_{t+1|t}\tilde{\boldsymbol{y}}_{t+1} \tag{16}$$

where $\tilde{\boldsymbol{y}}_{t+1}$ is the innovation or residual between the observed state and its predicted observation:

$$\tilde{\mathbf{y}}_k = \boldsymbol{z}_{t+1} - h(\boldsymbol{\mu}_{t+1|t}, \mathbf{0}) \tag{17}$$

$$\Sigma_{t+1|t+1} = (I - K_{t+1|t}H_{t+1})\Sigma_{t+1|t} \tag{18}$$

where $K_{t+1|t}$ is the Kalman Gain matrix defined by:

$$K_{t+1|t} := \Sigma_{t+1|t} H_{t+1}^T S_{t+1|t}^{-1} \tag{19}$$

where S is the innovation covariance matrix given by:

$$S_{t+1|t} = H_{t+1}\Sigma_{t+1|t}H_{t+1}^T + R_{t+1}VR_{t+1}^T \tag{20}$$

Using these equations, a near optimal state estimate can be accomplished given enough time steps.

D. EKF Algorithm

The EKF uses the above equations in an algorithmic way to get accomplish its goal. The algorithm can be divided into two phases, prediction and update.

1) Initialization Phase: Initially, the algorithm requires some information about the current expected state $\mu_{t|t}$ and covariance matrix $\Sigma_{t|t}$. These values must be assigned manually by the algorithm designed in code. This is typically a trivial task as the original state can be expressed as the origin, $\mu_{t|t} = \mathbf{0} \in \mathbb{R}^n$. The covariance is typically initialized as a diagonal matrix with incredibly small elements $\Sigma_{t|t} \approx 0_{[n,n]}$.

2) Prediction Phase: In the prediction phase, the algorithm uses the prior expected value of the state $\mu_{t|t}$ with the prior control inputs $\mathbf{u}_t$ as functions of the motion model (10) to compute the next expected value of the state $\mu_{t+1|t}$. Additionally, the prediction covariance $\Sigma_{t+1|t}$ (15) is computed using the Jacobian of the motion model w.r.t the state F_t (10) and evaluated at the prior expected state and control input, and the prediction covariance Jacobian Q_t (11). For every time step, a prediction phase is necessary.

3) Update Phase: In the update phase, the algorithm computes the current expected state $\mu_{t+1|t+1}$ using the current time steps predicted state $\mu_{t+1|t}$, the Kalman Gain $K_{t+1|t}$ and innovation $\tilde{\mathbf{y}}_k$ (16). Similarly, the current covariance $\Sigma_{t+1|t+1}$ is computed using the Kalman Gain, Jacobian of the observation model H_{t+1} evaluated at the previous expected state $\mu_{t|t-1}$. Because the innovation (17) is dependent on measurement observations z_{t+1}, the update phase may not always be able to sync up with the prediction phase. This is because a sensor may only be able to provide observation data at a rate slower than that of the next control inputs being computed. Hence, if the algorithm waited for an observation the prediction of the state would no longer be accurate.

To circumvent this, update phases are typically only conducted when a measurement is available. As a result, the faster a sensor can provide information the more accurate a state estimate should become.

III. EKF - Differential Drive Robot

To demonstrate the ability of the EKF, an example of the known trajectory of a differential drive robot will be estimated in the presence of process and observational noise.

ALGORITHM 1
Extended Kalman Filter

Input: v_0,ω_0,$\Sigma_{t|t}$, Q_t, R_t, $f(\cdot)$, $h(\cdot)$, $\mathbf{u}_{0:T}$
Update Rate: r
for $t = 0$ to T **do**
  if t=0 **then**
    Initialize EKF:
    $\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1} \sim \mathcal{N}(\mu_{t|t}, \Sigma_{t|t})$
  else
    Prediction Phase:
    $\mu_{\mathbf{t+1|t}} = f(\mu_{t|t}, \mathbf{u}_t, \mathbf{0})$
    $\mathbf{\Sigma}_{t+1|t} = F_t \Sigma_{t|t} F_t^T + Q_t W Q_t^T$
  end if
  if $t\%r$ and $t \neq 0$ **then**
    Update Phase:
    $\tilde{\mathbf{y}}_k = \boldsymbol{z}_{t+1} - h(\boldsymbol{\mu}_{t+1|t}, \mathbf{0})$
    $S_{t+1|t} = H_{t+1} \Sigma_{t+1|t} H_{t+1}^T + R_{t+1} V R_{t+1}^T$
    $K_{t+1|t} := \Sigma_{t+1|t} H_{t+1}^T S_{t+1|t}^{-1}$
    $\boldsymbol{\mu}_{t+1|t+1} = \boldsymbol{\mu}_{t+1|t} + K_{t+1|t} \tilde{\boldsymbol{y}}_{t+1}$
    $\Sigma_{t+1|t+1} = (I - K_{t+1|t} H_{t+1}) \Sigma_{t+1|t}$
  end if
end for

A. Motion and Observation Models

We start with a differential drive robot's motion model in continuous time:

$$f(\mathbf{x}, \mathbf{u}, \mathbf{w}) = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} + \mathbf{w} \tag{21}$$

where

$$\mathbf{u} = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

and where v,w are the linear and angular velocities of the robot respectively, ω is the heading angle of the robot. Lastly, $\mathbf{w} \in \mathbb{R}^3$ is additive Gaussian process noise $\mathbf{w} \sim \mathcal{N}(0, W)$. However, given that the EKF algorithm will ultimately run on a computer it is necessary to discretize the motion model:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) = \mathbf{x}_t + \Delta t \begin{bmatrix} v_t\cos\theta \\ v_t\sin\theta \\ \omega_t \end{bmatrix} + \mathbf{w}_t \tag{22}$$

where Δt is the time between the algorithm's loop rate. This will be set to $\Delta t = 0.1$s

Next, we define the observation model of the system. For sake of simplicity, we model this as perfect state feedback but with observation noise:

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{v}_t) = \mathbf{x}_t + \mathbf{v}_t \tag{23}$$

where $\mathbf{v}_t \in \mathbb{R}^3$ is additive Gaussian observation noise $\mathbf{v}_t \sim \mathcal{N}(0, W)$.

Thus, we can define the Jacobians necessary for the prediction and update phases as so:

$$F_t = \begin{bmatrix} 1 & 0 & -v_t\sin\theta_t\Delta t \\ 0 & 1 & v_t\cos\theta_t\Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$

$$H_t = I \in \mathbb{R}^3 \tag{25}$$

$$Q_t = I \in \mathbb{R}^3 \tag{26}$$

$$R_t = I \in \mathbb{R}^3 \tag{27}$$

B. Simulation Setup

With the motion and observation models defined, we now just need to numerically define the initial conditions, noise standard deviation, and time horizon. Conveniently, in many situations a systems initial state can be expressed as zero vectors of the state and control input because a system should initially start at rest and at a predetermined reference state. We will also do this except start the robot with a linear velocity of 1 to create interesting trajectories. Thus:

$$\mathbf{x}_0 = \mathbf{0} \in \mathbb{R}^3$$

$$\mathbf{u}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

We will simulate the standard deviation of the noise as:

$$\mathbf{w}_t \sim \mathcal{N}(0, W)$$

$$W = \begin{bmatrix} 0.03^2 & 0 & 0 \\ 0 & 0.03^2 & 0 \\ 0 & 0 & 0.01^2 \end{bmatrix}$$

$$\mathbf{v}_t \sim \mathcal{N}(0, V)$$

$$V = \begin{bmatrix} 0.015^2 & 0 & 0 \\ 0 & 0.015^2 & 0 \\ 0 & 0 & 0.005^2 \end{bmatrix}$$

and initialize the prediction covariance matrix as:

$$\mathbf{\Sigma}_{t|t} = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}$$

We conduct the simulation over the time horizon of 1000 steps, i.e. $t = 0 : 1000$, where each time step is the equivalent of 0.1 seconds (our Δt).

The EKF will attempt to get an accurate measurement of a randomly generated trajectory that is generated through adding a small normally distributed perturbation to the control inputs at each time step.

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \epsilon$$

$$\epsilon \sim \mathcal{N}(0, U)$$

$$U = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.01 \end{bmatrix}$$

The control inputs will also be constrained to prevent the robot from exceeding unrealistic speeds.

$$|v_t| \leq 2$$
$$|\omega_t| \leq 1$$

To determine the accuracy of EKF in this situation, we will compare the estimated state values through time with the equivalent dead-reckoned state estimates. Dead-Reckoning is an assumption that the dynamics of the system are perfect and to guess the next state without accounting for noise. It can be viewed as purely using the prediction step in the EKF without factoring in the covariance matrix. This will generally be a poor estimate of the state because as time progresses the noise error accumulates.

IV. Simulation Results

Results from the simulation clearly illustrate the efficacy of the EKF over dead-reckoned trajectory estimation. There are two sets of results, those with an update rate of 1 (i.e. the update phase will occur at every time step) and those with an update rate of 10 (the update phase will occur every 10 time steps).

First, the true trajectory and estimated trajectories are presented. As can be seen in Figures 1 2 3, the EKF estimate (red line) almost perfectly overlaps with the true trajectory (blue line), whereas the dead-reckoned estimate (magenta line) gets worse with each time step.

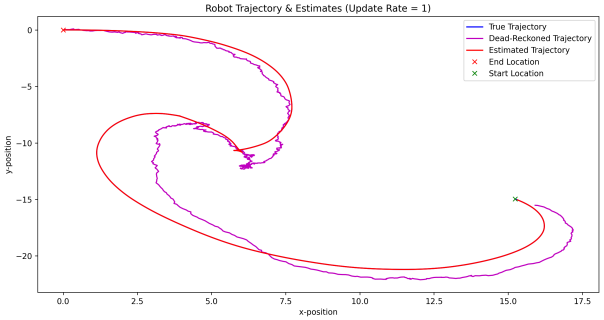

Fig. 1: Trajectories (Update Rate = 1)

As expected, the dead-reckoning state estimates are quite off from the true value. Each state (x,y,θ) are shown in Figure 4.

Similarly, the state estimate from the EKF is as expected, quite close to the true states (x,y,θ), as shown in Figures 5 and 6.

The disparate accuracy between the estimates can be seen in Figures 7, 8, 9, 10, 11, 12. Blue is the state estimate from Dead-Reckoning and Red is the EKF estimate. Unanimously, the EKF estimate is orders of magnitude better than Dead-Reckoning's for the majority of the time horizon. The Github repository to recreate this simulation can be found here: EKF-Simulation.

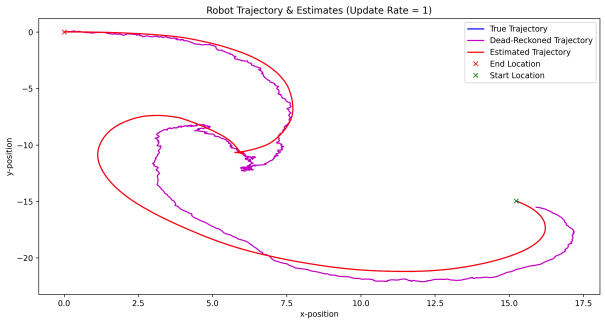

Fig. 2: Trajectories (Update Rate = 10)

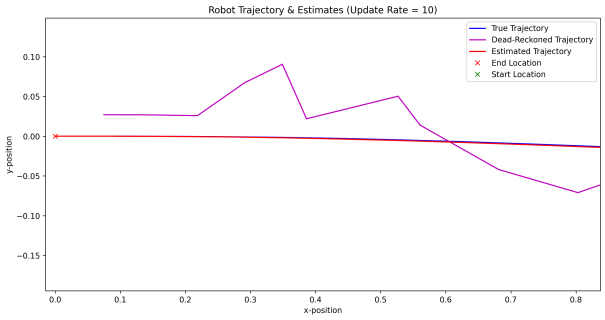

Fig. 3: Trajectories Zoom In (Update Rate = 10)

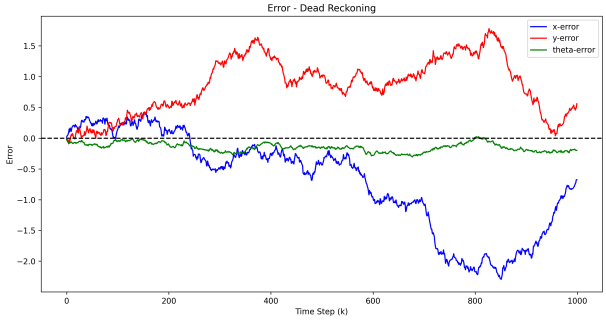

Fig. 4: Error - Dead Reckoning

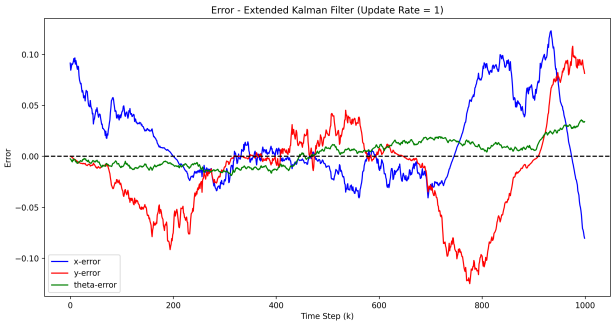

Fig. 5: Error - Extended Kalman Filter (Update Rate = 1)

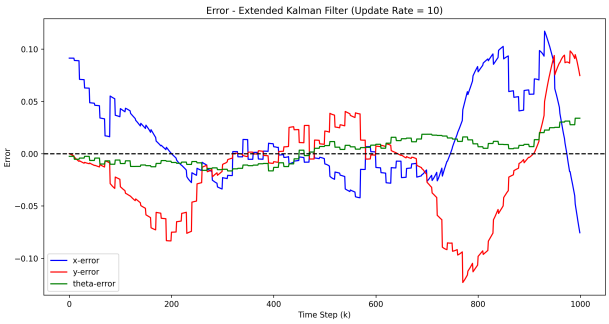

Fig. 6: Error - Extended Kalman Filter (Update Rate = 10)

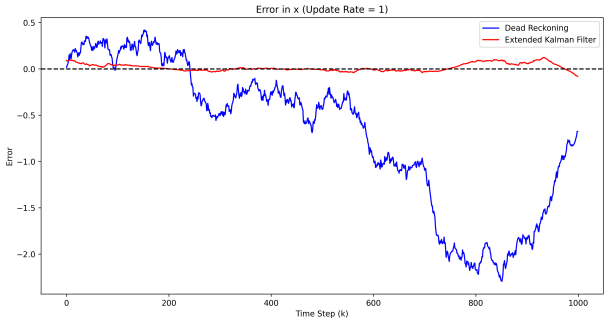

Fig. 7: Error - x State (Update Rate = 1)

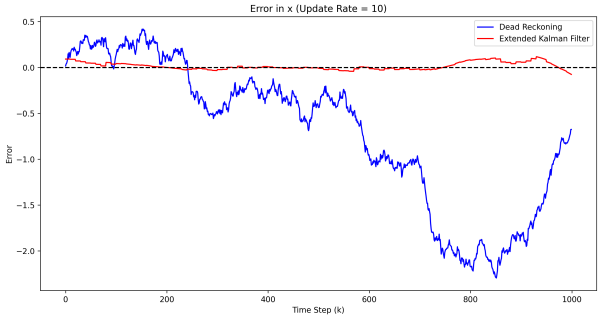

Fig. 8: Error - x State (Update Rate = 10)

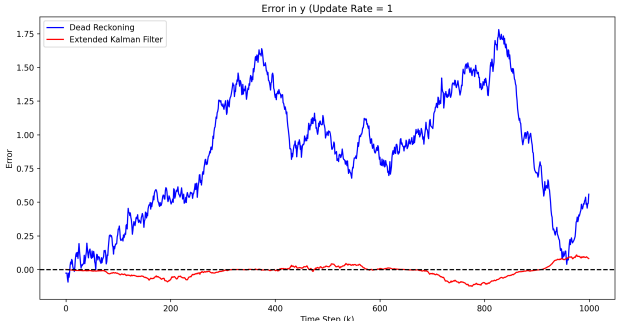

Fig. 9: Error - y State (Update Rate = 1)

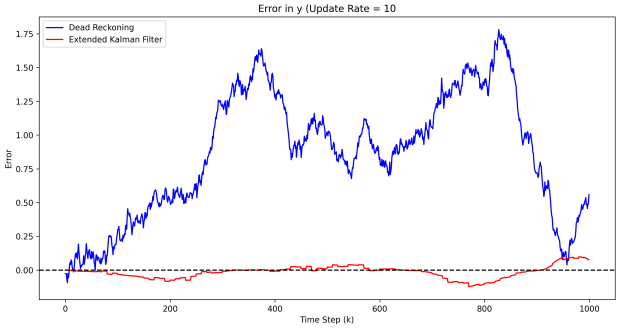

Fig. 10: Error - y State (Update Rate = 10)

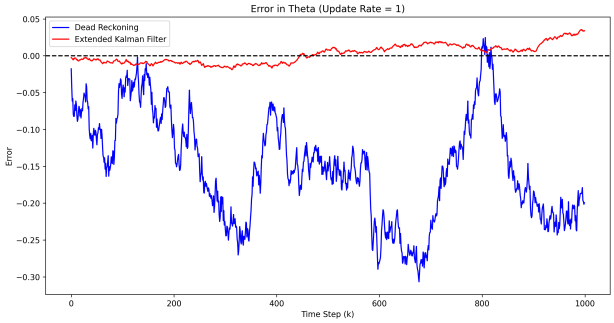

Fig. 11: Error - θ State (Update Rate = 1)

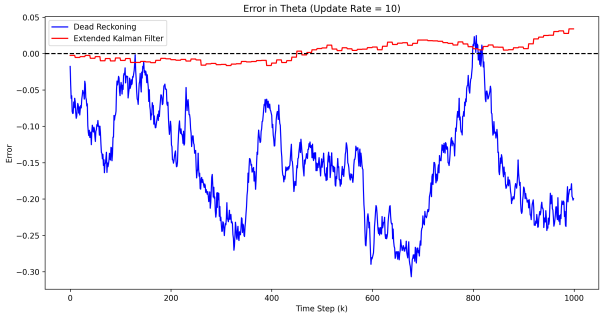

Fig. 12: Error - θ State (Update Rate = 10)