ROS Implementation of a Kalman Filter

Francesco Paolo Carmone
s308126
Computer Engineering in Automation and Cyber-Physical Systems
Politecnico di Torino
francescopaolo.carmone@studenti.polito.it

Abstract: This document explains what a Kalman Filter is, sheds light on why we opted for the Extended Kalman Filter, explores its practical use on two different pendulum models, delves into its Python implementation, and concludes with its integration into the Robot Operating System (ROS).

Keywords: CoRL, Kalman Filter, Learning, ROS

1 The Kalman Filter

The Kalman Filter (KF) is an algorithm that takes into account a sequence of measurements observed over time, accounting for statistical noise and other inaccuracies affecting zero-mean Gaussian systems, and estimates the state x of a discrete-time process.

Given any linear dynamical system, a Kalman Filter can be seen as the discrete-time duplicate of such model with the addiction of noise:

$$\begin{cases} x_t &= A_t x_{t-1} + B_t u_{t-1} + \varepsilon_t \\ y_t &= C_t x_t + R_d + \delta_t \end{cases} \tag{1}$$

Where:

1. ε_t random variable representing the process noise with Q_d variance
2. δ_t random variable representing the measurement noise with R_d variance

Both these two variables are assumed to be indipendent and normally distribuited.

During the **prediction phase**, the algorithm predicts the next state of the system based on its current state and the known dynamics of the system, and it is represented as a Gaussian distribution with mean $\bar{\mu}$ and covariance $\bar{\Sigma}$ that describe the expected state and the uncertainty associated with it. During the **update phase** (or correction phase), the covariance and mean are fine-tuned with the actual measurement, in order to better approximate the gaussian.

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2. Prediction:
3. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
4. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t$
5. Correction:
6. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + R_t)^{-1}$
7. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
8. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
9. Return μ_t, Σ_t

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

This model assumes the perfect knowledge of inputs, observations, of the transition and observation model. Moreover, it assumes that the system is linear, and its random variables are Gaussian.

Satisfied such assumptions, and if the model matches the real system and we know the noise covariances, then the Kalman Filter is said to be *optimal*, and it minimizes the mean square error between the state and its estimate

Such convenient algorithm, however, cannot be used when dealing with nonlinear functions, that are used to describe most real problems.

One way to solve the previously formulated problem is by using the so called Extended Kalman Filter, that consists in dinamically linearizing the system at each time instant using the Tayler Expansion, in order to approximate the nonlinear model with a linear model around its mean. However generally speaking this solution is not an optimal estimator, i.e., we are not assured that it minimizes the mean square error between the state and its estimate.

2 Equations for a Single Pendulum

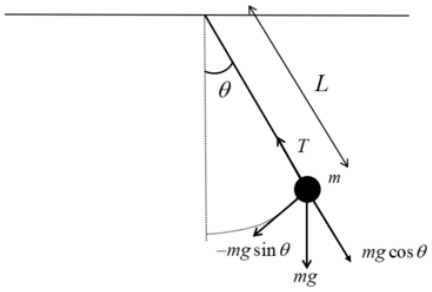


Figure 1: Single Pendulum forces diagram

The model chosen for the application of the Extended Kalman Filter is a single pendulum which is free to oscillate. The components of the state vector x are respectively the angle θ and its angular velocity $\dot{\theta}$. The expected output vector, is simply the angle, hence $h(x) = x_1$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \doteq \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

First and foremost one has to rewrite the system as follows

$$\begin{cases} \dot{x} &= \beta(x) \\ y &= Cx \end{cases}$$

We can immediatly find $\beta(x)$ and C

$$\dot{x} = \beta(x) \doteq \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ -\frac{g}{l}\sin(\theta) \end{bmatrix}, \qquad \frac{\partial y}{\partial x} = C = [1 \quad 0]$$

At this point can discretize the system by using the forward Euler's method

$$x_k \sim x_{k-1} + \Delta t \cdot \beta(x)$$

$$x_k \sim \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \Delta t \begin{bmatrix} x_2 \\ -\frac{g}{l}\sin(x_1) \end{bmatrix} = \begin{bmatrix} x_1 + \Delta t\, x_2 \\ x_2 - \Delta t \frac{g}{l}\sin(x_1) \end{bmatrix}$$

The Jacobian of the discrete system dynamics w.r.t. the state variables is

$$\frac{\partial x_k}{\partial \theta} = \begin{bmatrix} 1 & \Delta t \\ -\Delta t \frac{g}{l} \cos(x_1) & 1 \end{bmatrix}$$

3 Extra, Equations for a Double Pendulum

The double pendulum is a deterministic but chaotic system

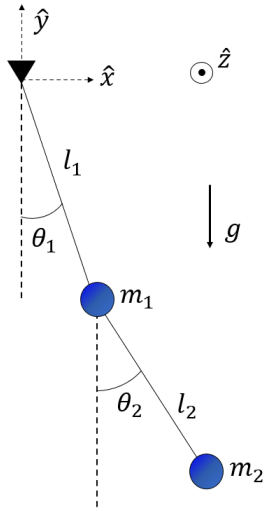


Figure 2: Diagram of a double pendulum

The components of the state vector x I chose are respectively the angles θ_1, θ_2 and their angular velocities $\dot{\theta}_1, \dot{\theta}_2$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \doteq \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Out of a burst of creativity, all parameters have been set to one: $m_1 = m_2 = 1$ KG, $l_1 = l_2 = 1$ m.

Retracing all the steps in section 2 we have to calculate

$$\dot{x} = \beta(x) = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$

Where

$$\theta_1'' = \frac{-g\,(2\,m_1 + m_2)\,\sin\theta_1 - m_2\,g\,\sin(\theta_1 - 2\,\theta_2) - 2\,\sin(\theta_1 - \theta_2)\,m_2\,(\theta_2'^2\,L_2 + \theta_1'^2\,L_1\cos(\theta_1 - \theta_2))}{L_1\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

$$\theta_2'' = \frac{2\,\sin(\theta_1 - \theta_2)\,(\theta_1'^2\,L_1\,(m_1 + m_2) + g(m_1 + m_2)\cos\theta_1 + \theta_2'^2\,L_2\,m_2\cos(\theta_1 - \theta_2))}{L_2\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

We can now apply Euler's forward method to obtain the discrete time dynamic

$$x_k \sim x_{k-1} + \Delta t \cdot \beta(x)$$

And finally we can its jacobian:

$$\frac{\partial x_k}{\partial \theta} = \begin{bmatrix} 1 \\ 0 \\ \frac{981\,\cos(x_1-2\,x_2)}{10000} + \frac{\cos(x_1)}{100} + \frac{\cos(x_1-x_2)\left(\cos(x_1-x_2)\,{x_3}^2+{x_4}^2\right)}{50\,(\cos(2\,x_1-2\,x_2)-3)} - \frac{{x_3}^2\,\sin(x_1-x_2)^2}{50\,(\cos(2\,x_1-2\,x_2)-3)} + \frac{\sin(x_1-x_2)\,\sin(2\,x_1-2\,x_2)}{25\,(\cos(2\,x_1- \\ -\frac{\sin(x_1-x_2)\left(\frac{981\,\sin(x_1)}{50}-{x_4}^2\,\sin(x_1-x_2)\right)}{50\,(\cos(2\,x_1-2\,x_2)-3)} - \frac{\cos(x_1-x_2)\left(2\,{x_3}^2+\cos(x_1-x_2)\,{x_4}^2-\frac{981\,\cos(x_1)}{50}\right)}{50\,(\cos(2\,x_1-2\,x_2)-3)} - \frac{\sin(x_1-x_2)\,\sin(2\,x_1-2\,x_2)\,\left(2\,x \end{bmatrix}$$

The jacobian is so big it does not fit on the paper. However the matrix on its entirety can be seen here, and its source code can be found here.

4 Python EKF Implementation and ROS Integration

I completed the given Python code with calculations I derived earlier, and used the library `matplotlib` to produce the plots. Then I ran the simulation with different values for Q_d and R_d, as shown in the next section.

As it concerns the ROS integration for the single pendulum, I designed a package with three nodes:

1. `pendulum`, which produces and publishes the real system state at each time instant in the topic `real_system`

2. `sensor`, which receives the real measurement, and emits the measure with noise in the topic `sensor_measures`

3. `ekf_singlependulum`, which receives the noisy measurement, runs the EKF algorithm, and emits its result in the topic `estimated_system`

A video example can be found by clicking here or here

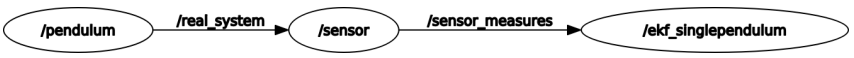


Figure 3: ROS topics relationship between the nodes for the Simple Pendulum

Equivalently, the nodes for the double pendulum are

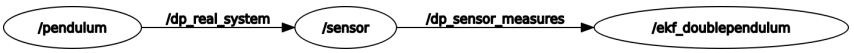


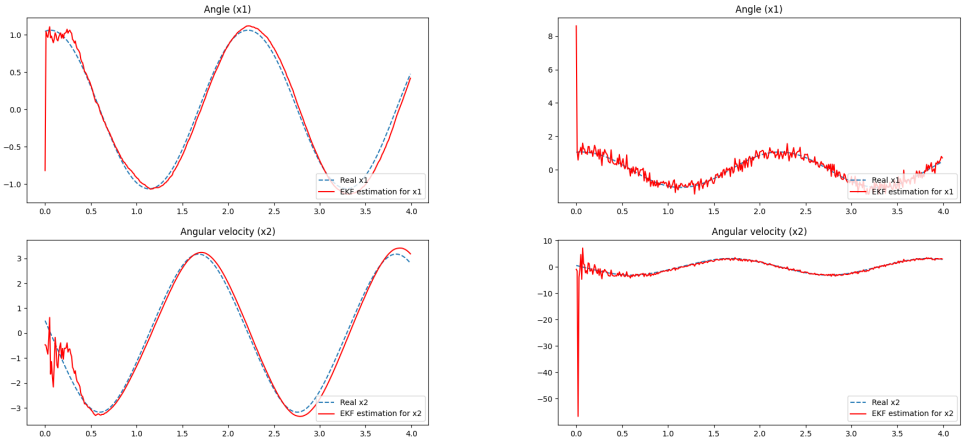
Figure 4: ROS topics relationship between the nodes for the Double Pendulum

The main difference between the Python code and the ROS one is the way the real system is computed. Infact, the Python code integrates the continuous system in its entirety and only then shows its trajectory, while the ROS code computes the next state at each time instant through the discrete dynamics. By doing this, the ROS code can run indefinitely. Or at least until your computer runs out of RAM.

5 Experimental Results

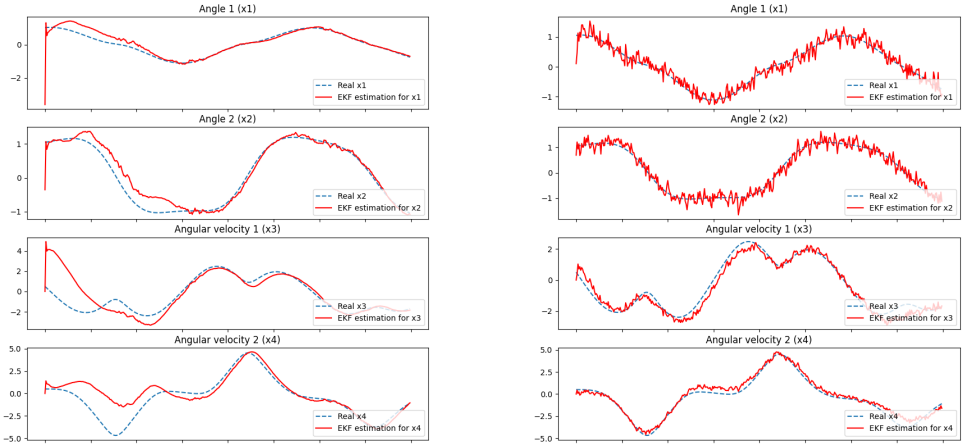
The algorithm (1) involves two random variables representing the noise, with covariance matrices Q_d and R_d, respectively affecting the state and the measurements.

Every result has to be compared with with respect to Figure 5a and 6a, representing an arbitrarily *good estimate*. Contrariely to intuition, choosing all covariances noise equal to zero not guarantee the best possible approximation, as it can be seen in Figure 5b and 6b.



(a) Benchmark graph with an arbitrarily good estimate (b) Example with null (very close to zero) covariances

Figure 5: The different choices of covariance matrices on a single pendulum

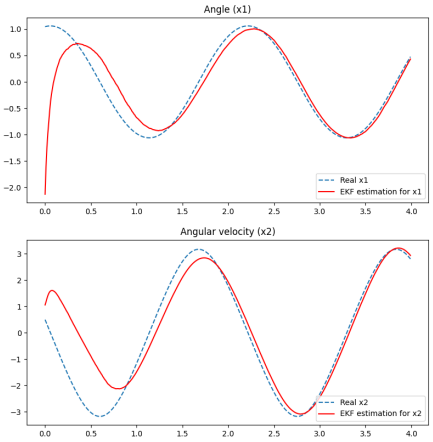


(a) Benchmark graph with an arbitrarily good estimate (b) Example with null (very close to zero) covariances

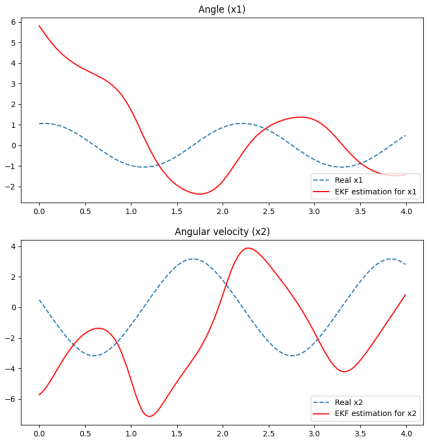
Figure 6: The different choices of covariance matrices on a double pendulum

5.1 Fixing Q_d and increasing R_d

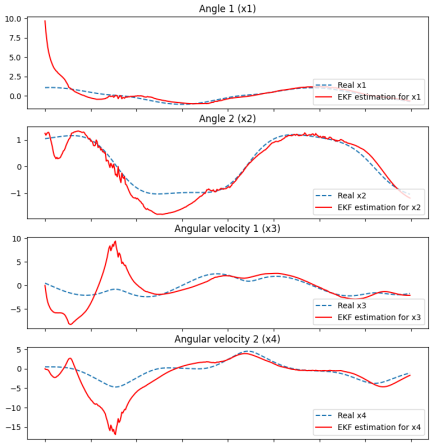
Increasing R_d means increasing the noise in measurement, which the EKF uses compute its predictions. Ideally, that value should be zero, as it would represent the real measures uneffected by noise.



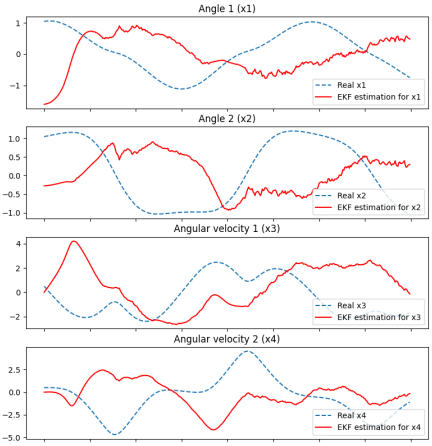
(a) $R_d = 50$



(b) $R_d = 5000$



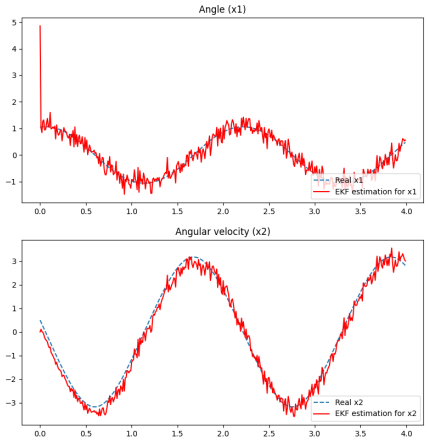
(a) $R_d = 50$



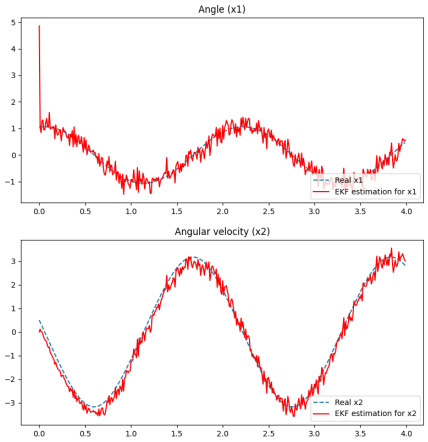
(b) $R_d = 5000$

5.2 Fixing R_d and increasing Q_d

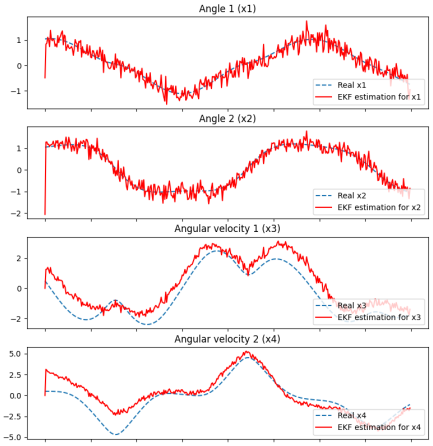
An higher value of Q_d adds more noise in the state prediction, thus generating a spiky behaviour. However, the *accuracy* of the prediction is quite high, (the prediction follows the real trajectory), since the matrix R_d was fixed with a low value.



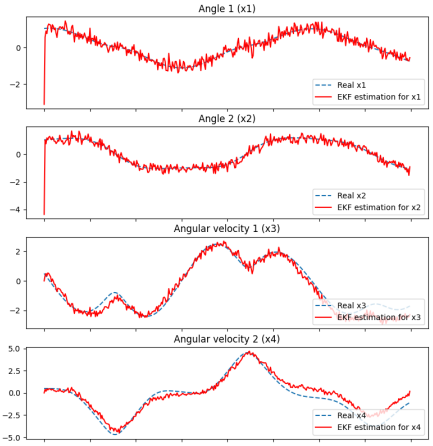
(a) $Q_d = I_2$



(b) $Q_d = 100I_2$



(a) $Q_d = I_4$



(b) $Q_d = 100I_4$

6 Conclusion

When attempting to estimate the state of a nonlinear system with the Extended Kalman Filter, one must carefully select the values of the covariances Q_d and R_d. However, due to the non-optimal nature of the algorithm, perfect convergence of the estimation to the actual state is not guaranteed, and almost only work with systems that are almost linear. In such cases, it is recommended to explore alternative solutions, one of which is the Unscented Kalman Filter. However, that solution is, just like all other Kalman Filters, assumes a unimodal gaussian belief, which can be limitating with many applications.