Report Laboratory 01 - Robot Learning

Federico Pulinas
s319881@studenti.polito.it

Abstract: The purpose of this report is the present and discuss the results obtained from carrying out the laboratory 01 assigned about the usage of the Extended Kalman Filter applied on a pendulum.

Keywords: EKF, Pendulum, Prediction

1 Introduction

This laboratory aim is to demonstrate the main application of the Extended Kalman Filter (EKF) and deal with the filtering problem, which revolves around estimating the state of a stochastic system due to noisy observations. The case study is about a simple unactuated pendulum system, with noisy measurements of its angle.

2 Results

2.1 Unactuated pendulum system

In this section we focus on a simple pendulum system free to oscillate, given its initial conditions, and solely affected by gravity.
The system is described by 2 state variables, the **angle** and the **angular velocity**:

$$x_1 = \theta, x_2 = \dot{\theta}$$

thus

$$\dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{g}{l} \sin x_1 \end{bmatrix}$$

where $f_1 = x_2$ and $f_2 = -\frac{g}{l} \sin x_1$ represent the *state-space* of the system. For the implementation of the python algorithm it's necesasry to derive the **discrete-time** version of the dynamical system, hence passing to the discrete domain.
The system evolves over (discrete) time as:

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

where

$$\beta(x) = \dot{x}$$

with Δt sufficiently small so it can approximate the continuous time properly.
In order to implement the forward dynamic of the discrete system in the algorithm we have to compute the jacobian with respect to the state variables.
Expanding the $\dot{x_k}$ and replacing $\beta(x_{k-1})$ we obtain:

$$\begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} = \begin{bmatrix} x_{1,k-1} \\ x_{2,k-1} \end{bmatrix} + \Delta t \cdot \begin{bmatrix} x_{2,k-1} \\ -\frac{g}{l} \sin x_{1,k-1} \end{bmatrix}$$

To implement this nonlinear system in the EKF we have to implement the extended version:

$$f(t, x_{k-1}) = f(x_{k-1})$$

Thus

$$f(x_{k-1}) = \begin{bmatrix} f_1(x_{k-1}) \\ f_2(x_{k-1}) \end{bmatrix} = \begin{bmatrix} f_1(x_{1,k-1}, x_{2,k-1}) \\ f_2(x_{1,k-1}, x_{2,k-1}) \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} x_1 + \Delta t x_2 \\ x_2 - \Delta t \frac{g}{l} \sin x_1 \end{bmatrix}$$

By definition of jacobian we obtain that

$$J_f = \begin{bmatrix} 1 & \Delta t \\ -\Delta t \frac{g}{l} \cos x_1 & 1 \end{bmatrix}$$

As the measurement model, since we want to output the angle then

$$z_t = h(x_k) = x_1 = [1 \quad 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

therefore

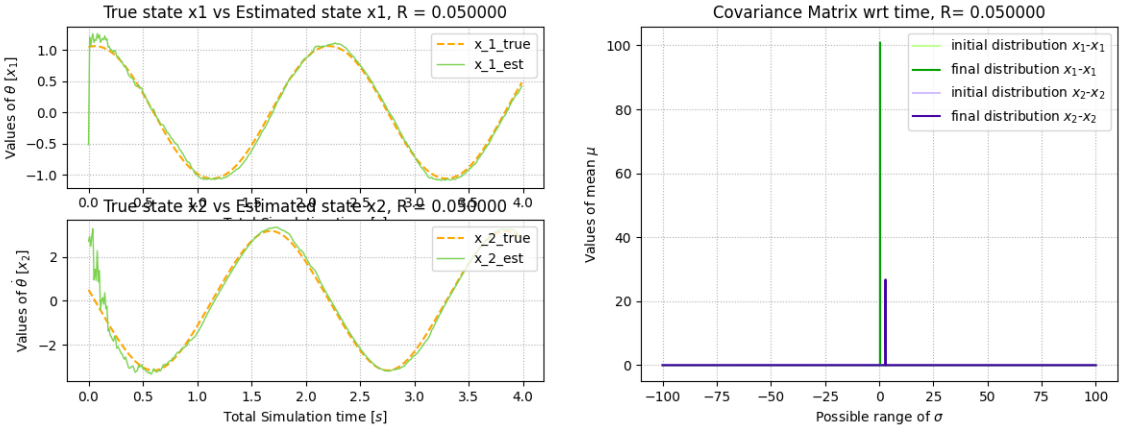
$$J_h = [1 \quad 0]$$

2.2 Implement an Extended Kalman Filter in python

After discretizing the system we use the previous results to implement the EKF. The aim of the filter is to estimare both the angle and the angular velocity (the state variables) of the system. Considering the provided datas

$$Q = \begin{bmatrix} 0.00001 & 0 \\ 0 & 0.00001 \end{bmatrix}, R = 0.05$$

we obtain the following results:



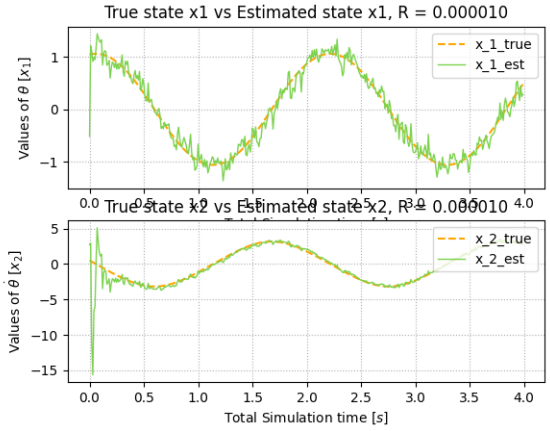
(a) true states vs estimated states plots

(b) normal distribution of $P_{11(t_0)}$ and $P_{22(t_{end})}$

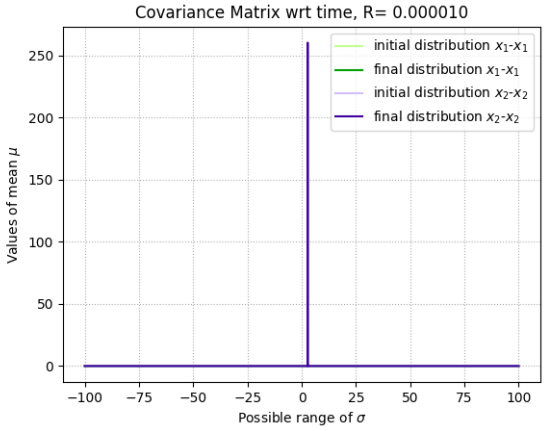
Figure 1

Studying the two plot we can conclude that with these values of Q and R we can get a pretty good estimate of the state, even though at the initial time it's a bit noisy, given by the fact that, differently from the linear Kalman Filter, the Extended version is not an optimal estimaor. In addition to this, since it's a recursive algorithm, with few samples at the beginning the estimate is less accurate. As it reguard the covariance we can notice that the final distribution resembles a Dirac's delta, meaning that the accuracy of the filter is such that the predicted value is very similar to the true one.

An interesting test is to vary the value of R, that in the EKF, influences the gain matrix K_t and consequently the estimate of the mean and covariance matrix.

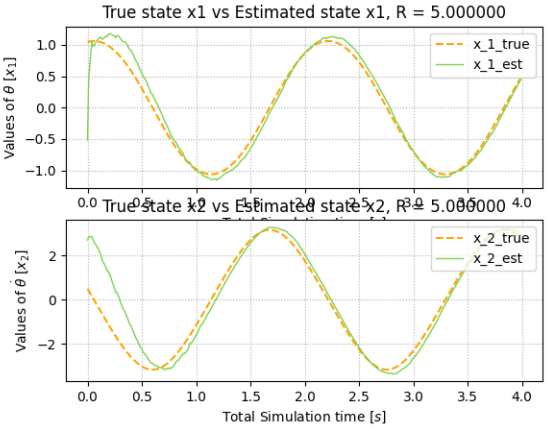


(a) true states vs estimated states plots

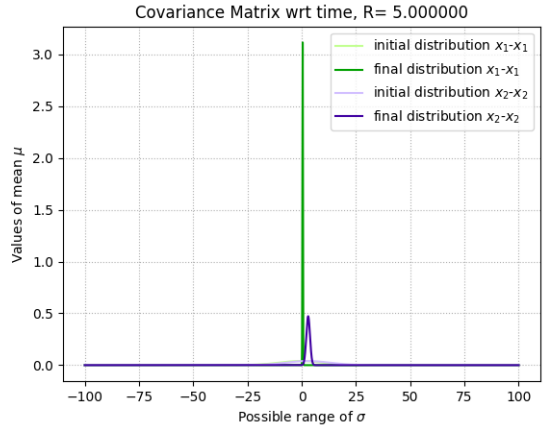


(b) normal distribution of $P_{11(t_0)}$ and $P_{22(t_{end})}$

Figure 2: $R = 0.00001$

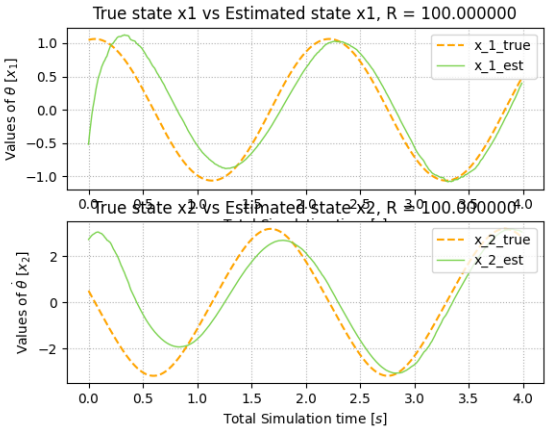


(a) true states vs estimated states plots

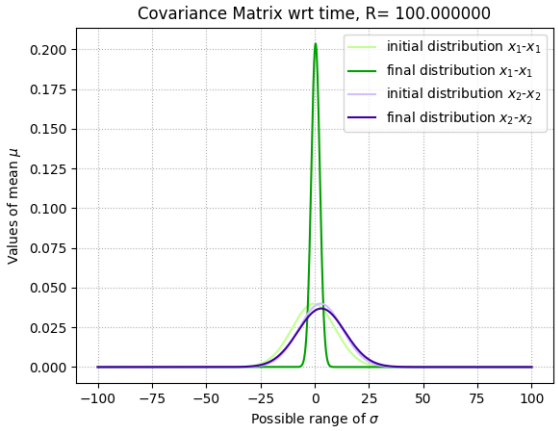


(b) normal distribution of $P_{11(t_0)}$ and $P_{22(t_{end})}$

Figure 3: $R = 5.0$

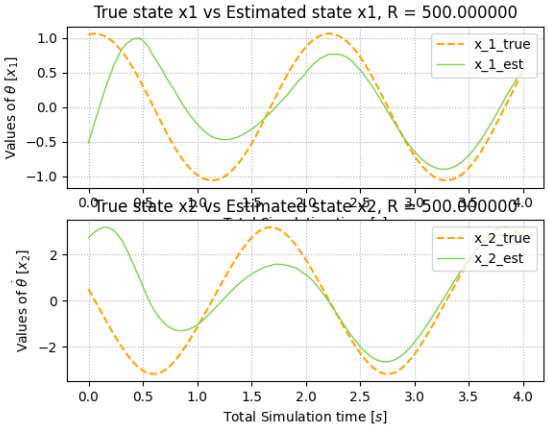


(a) true states vs estimated states plots

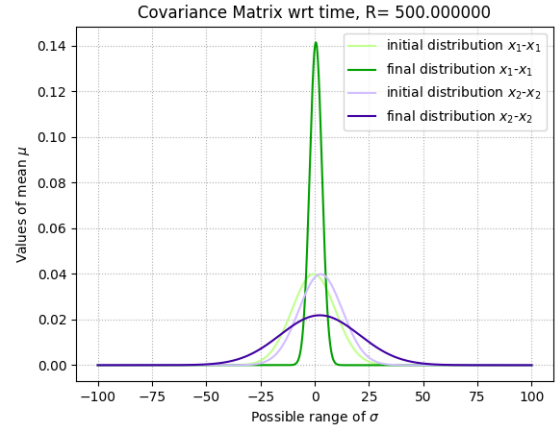


(b) normal distribution of $P_{11(t_0)}$ and $P_{22(t_{end})}$

Figure 4: $R = 100.0$

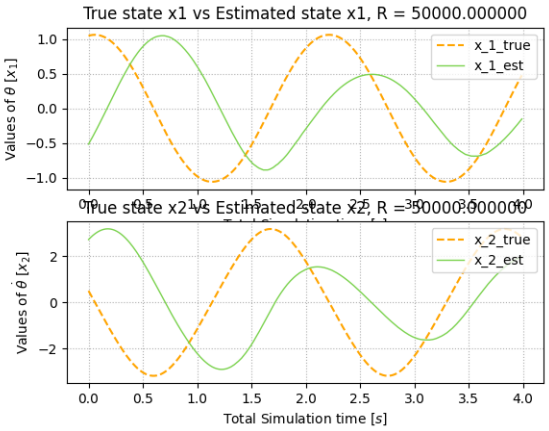


(a) true states vs estimated states plots

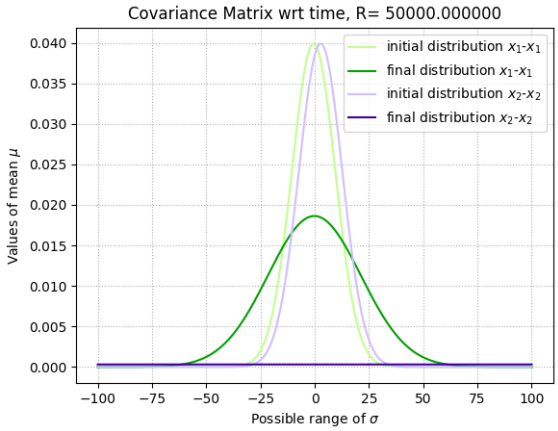


(b) normal distribution of $P_{11(t_0)}$ and $P_{22(t_{end})}$

Figure 5: $R = 500.0$



(a) true states vs estimated states plots



(b) normal distribution of $P_{11(t_0)}$ and $P_{22(t_{end})}$

Figure 6: $R = 50000.0$

As R represents the measurement noise we can conclude that by increasing its value we are consequently increasing the influence of the noise in the correction phase of the filter, thus resulting in a less accurate prediction as R grows. From a practical point of view we can see that until $R = 500$, given a time of 4 seconds, the estimated value still converges to the real one. Indeed, for very high values like $R = 50000.0$, in the same time interval, the estimate is pretty far from being the same as the real one, hence requiring more time to converge to it. The observation above are also confirmed by the figures (b) that show how the normal distribution is less similar to the Dirac's delta, i.e., the estimate is less accurate.

2.3 ROS Integration

In this section we study the implementation of the EKF in a ROS-based application in order to simulate a real-world example.

The node graph of the system as been implemented as follows:

Figure 7: ROS node graph

Three **nodes** have been created in order to manage:

- the pendulum and its dynamics
- the sensor that measures the real values of the pendulum and add the noise
- the EKF that takes the sensor's output and tries to predict the dynamics of the pendulum and outputs the results

In between the node two **topics** have been added for the communication of the nodes:

- pendolum_true_state
- sensor_noisy_states

From a technical point of view each node has a Publisher to output the results of their computations and a Subscriber, with the exception of the pendulum_talker, to read the datas published on the topic. The implementation is the same as the one used in the **section 2.2**, but managed in order to work in

synergy with the nodes-topic.
Moving on to the examination of the graphs, to effectively compare the results obtained with the previous one:

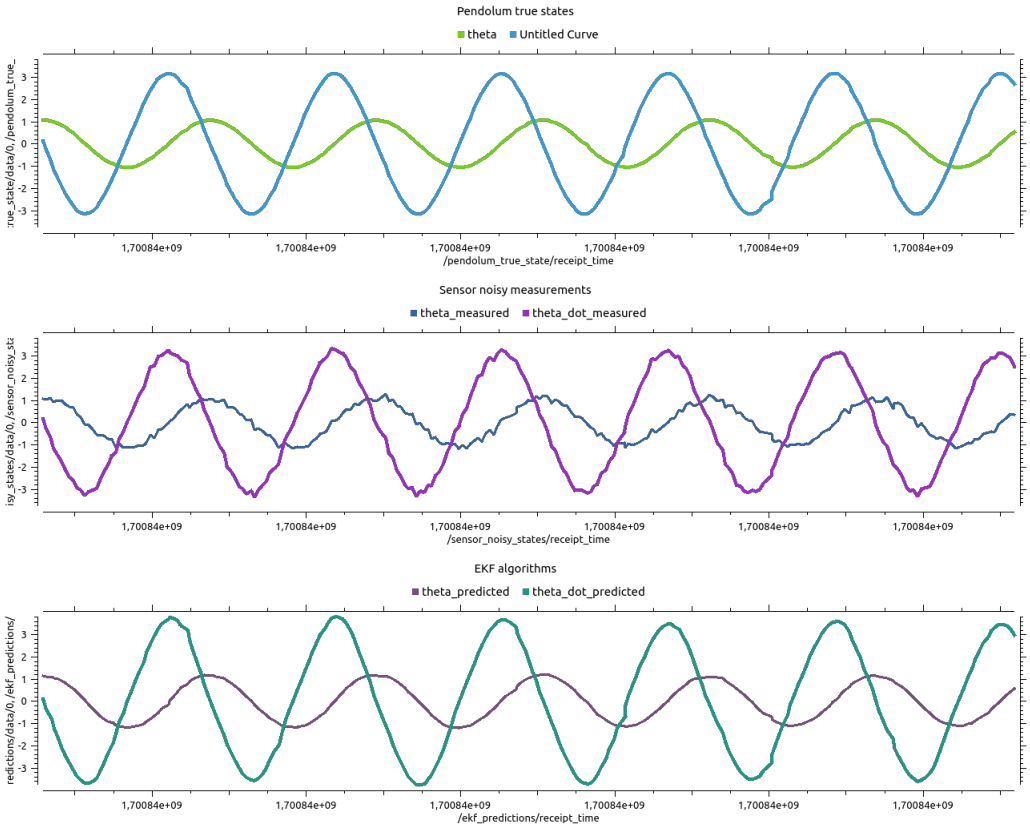


Figure 8: from top to bottom: Pendulum true states, Sensor noisy measurements, EKF algorithm

Taking a deeper look at the graph we can see how the third graph, the ekf estimate, is pretty similar to the first one, pendulum dynamics, meaning that with $R = 0.05$ the filter is resilient to the noise given by the sensor. If we want to do some considerations about the ROS implementation, in this case we used a rate of $\frac{1}{detaTime}$ where $deltaTime = 0.05$ using a $queue\_size = 100$ in order for the system to have enough storage to keep up with the publishing frequency. It's also possible that, given an higher frequency, the machine may lag resulting in a mismatched plot, meaning that also the mentioned parameters need to be tuned to have a proper result.