A cartoon character with arms crossed

AI-generated content may be incorrect.Proiect smecher cu kalman simplu kalman da mai extins putin si kalman fara miros

Circioroaba Radu-Andrei

Roland Groza

Dumitru Stefan

333AB

1. Context

The Kalman filter is a type of linear quadratic estimator which has been proposed in 1960 by R.E. Kalman as a solution to the linear filtering and estimation problems[1]. The most widely recognized application of the filter was its use in the Apollo 11 lunar landing project directed by NASA.

Since then, the Kalman filter has benefitted of a variety of uses in every domain of engineering where data filtering and estimation is required, from simple applications such as robotic arms and drones[2] to the most complex tasks such guidance and navigation systems in the aeronautical industry[3][4][5].

Many solutions have been proposed regarding the implementation of Kalman filters, most of which involve extensive calculations using microcontrollers. However, the major drawback of this method is that of prolonged execution time due to the sequential nature of the microcontroller. Naturally, the immediate solution to this would be the partitioning of tasks on the several cores of the microcontroller, which is not always possible and, even if it is, may not provide significant speed improvements.

That being said, we propose an alternative implementation of the Kalman filter using a Field Programmable Gate Array (FPGA), which provides great parallelism without a noticeable tradeoff of numerical precision.

1. Theoretical basis

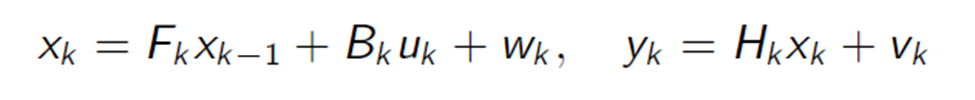
In order to arrive at the solution of the implementation problem, we need to expand on the theory of Kalman filters[6][7], their characteristics and components, as well as the advantages and disadvantages of using each filter. In this document, we propose three variations of the Kalman filter:

* Linear Kalman Filter (LKF)
* Extended Kalman Filter (EKF)
* Unscented Kalman Filter (UKF)

LINEAR Kalman Filter (ideal case)

1. Introduction

The (Linear) Kalman Filter works based on the assumption that the modelled dynamical system is linear and the system and measurement noises are polluted by Gaussian noise. The system produces an estimate based on the uncertainties in the process model, as well as the quality of the measurements, producing better and better results the longer it runs.

2. How it works

Fk = Ak

We will use the following notations:  
 xk = state vector

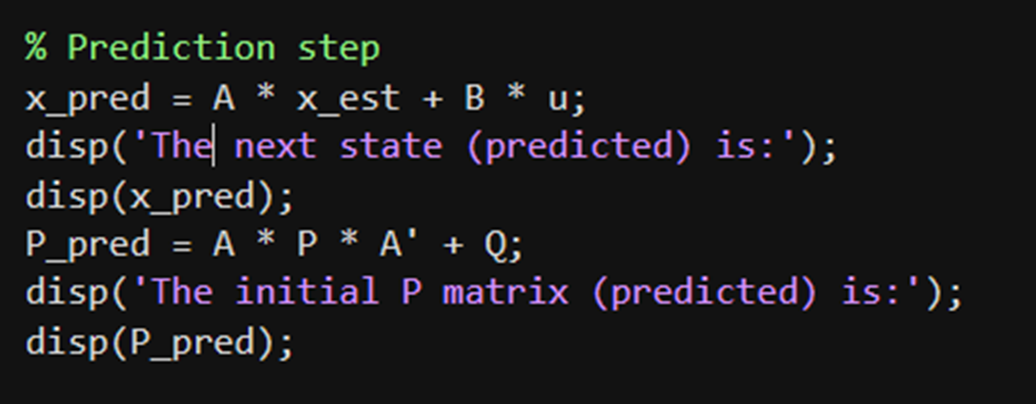
Yk = Zk = measurement

Ak = stare-transition matrix (n x n dimension matrix)

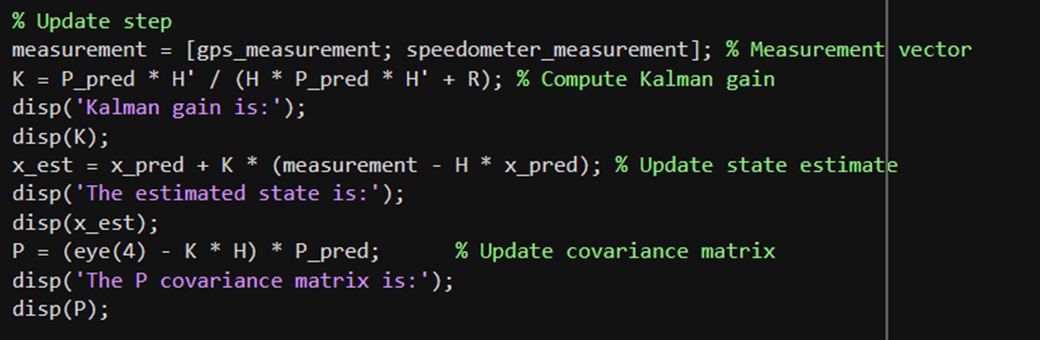
Bk = control matrix (n x m matrix dimension)

Hk = measurement matrix

Q,R = covariance matrices, Q for process and R for measurement



This is the first step, where we predict the a priori state and the covariance estimate (P is the initial covariance matrix, in this case being considered as )



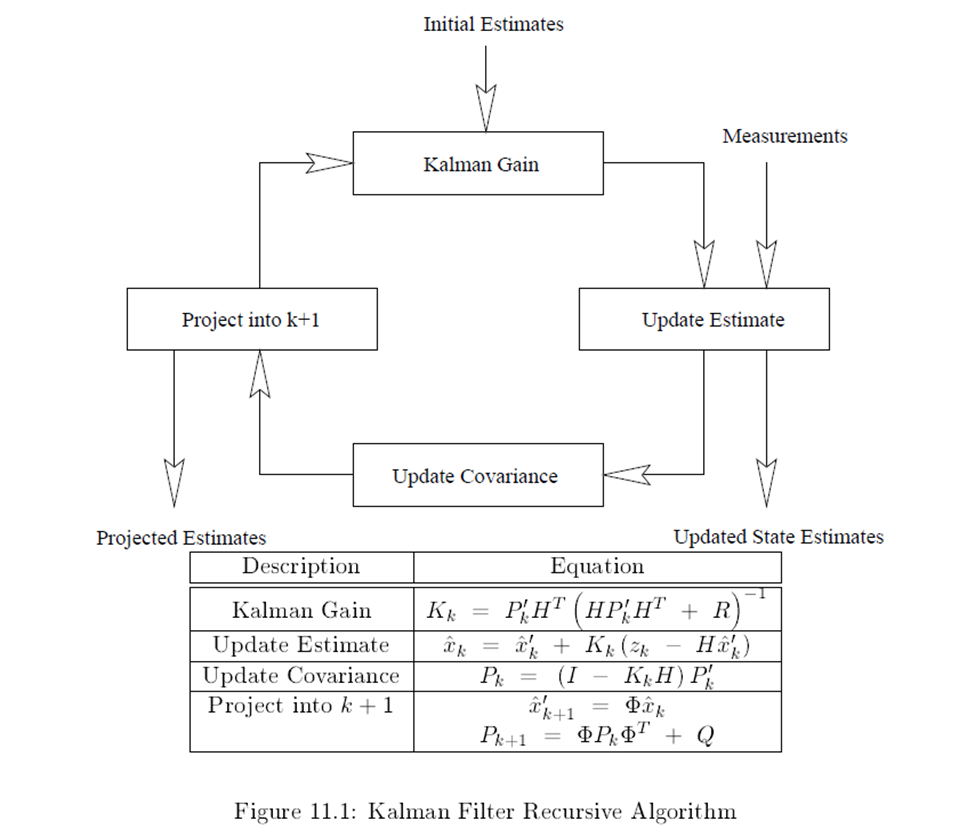
Now to the second part, the update. Here we update the Kalman gain, the covariance matrix P and the state estimate

**Advantages**

* Optimal for linear Gaussian systems
* Computationally fast
* Simple to implement

**Disadvantages**

* Only works for **linear systems**
* Sensitive to model inaccuracies
* Cannot handle strong nonlinearities



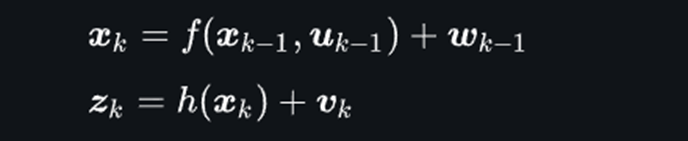
EXTENDED Kalman Filter (more non-linear)

1. Introduction

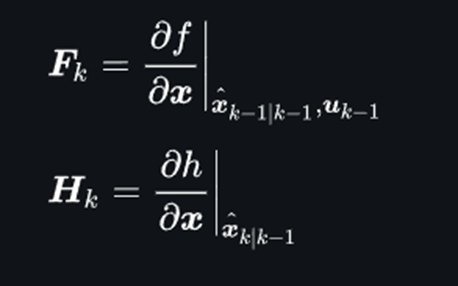
The Extended Kalman Filter extends the Kalman Filter (that’s why it is called an EXTENDED KALMAN FILTER - get it?) to nonlinear systems by linearizing the system around the current estimate. It is widely used in robotics, navigation, and control.

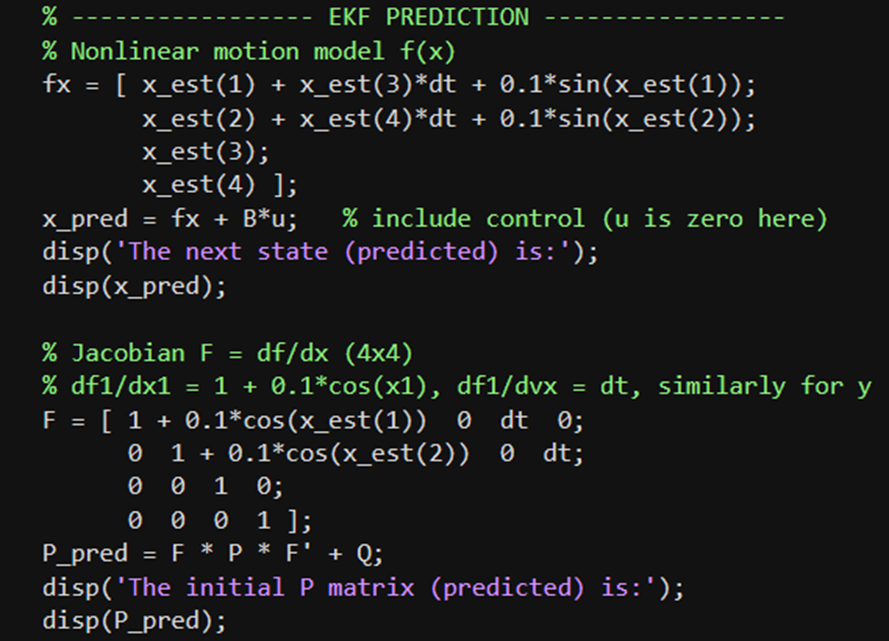
2. How it works

It is almost identical to Kalman filter, but there are some changes

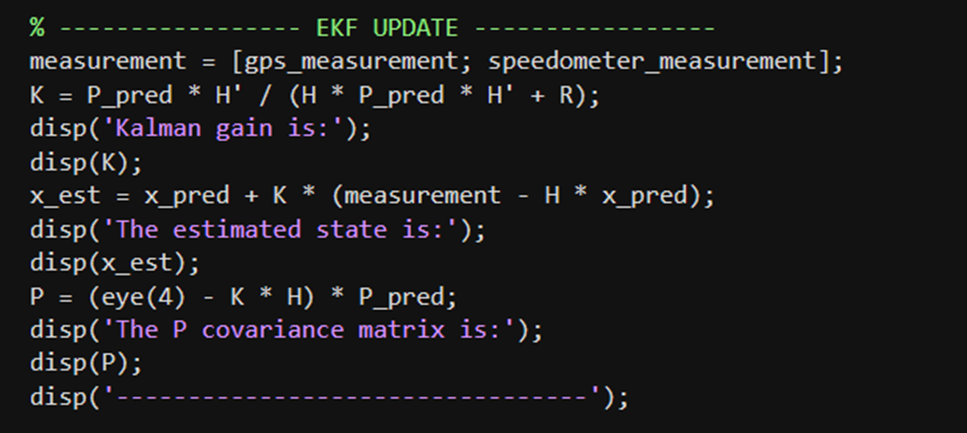


The linearisation is done using the Jacobian matrices:





The predict part is almost the same as the LKF predict, but A is replaced by F and the linear update by the non-linear function f()



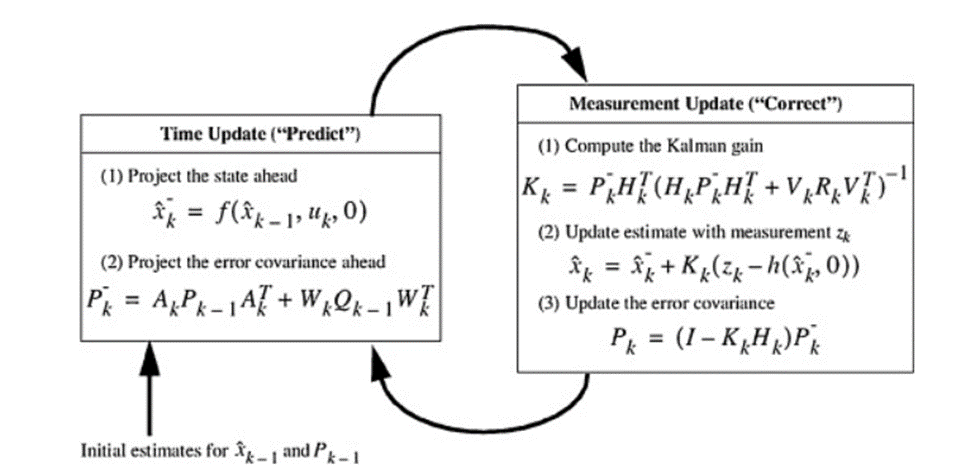
The only changes are that the H from KF is replaced by the Jacobian Hk

**Advantages**

* Handles **mild nonlinearities**
* Extends KF to nonlinear systems
* Conceptually simple (linearize + use KF equations)

**Disadvantages**

* Accuracy depends on linearization
* Can diverge if system is strongly nonlinear
* Requires computing **Jacobians** (analytical or numerical)

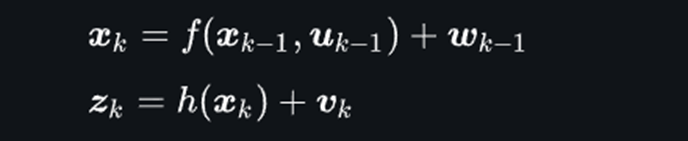


UNSCENTED Kalman Filter (Even more non-linear)

1. Introduction

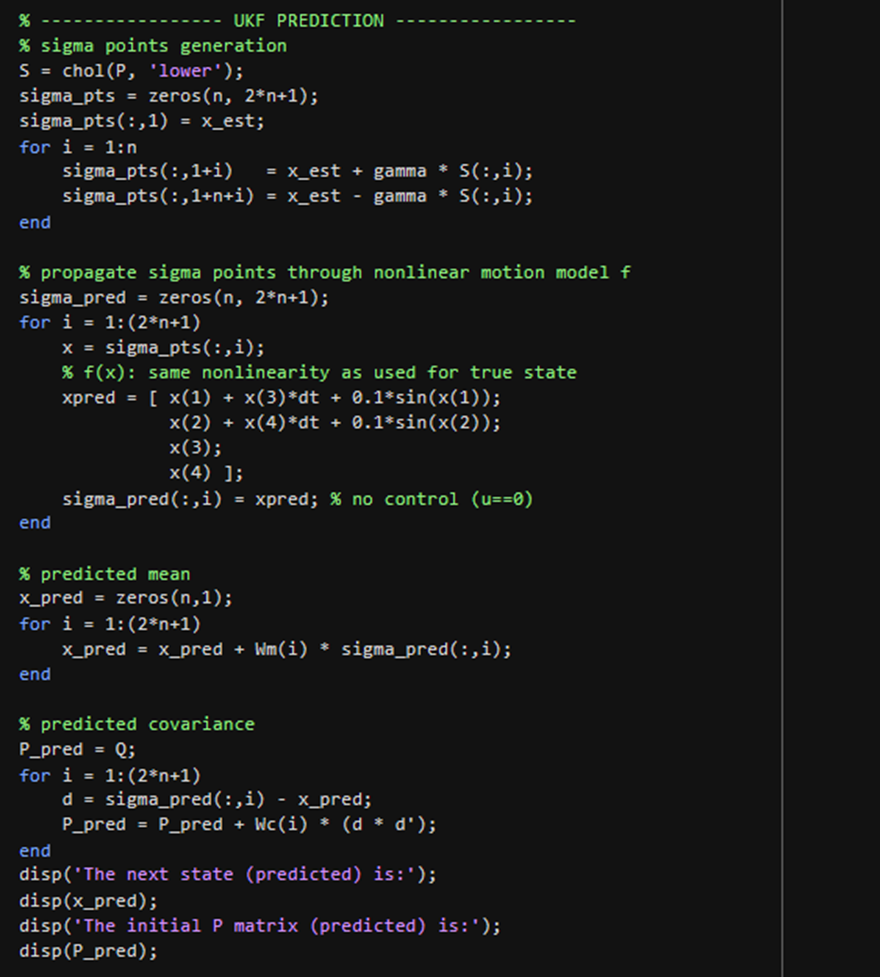
The UKF addresses EKF’s linearization limitation by using the Unscented Transform, which propagates a set of sample points (sigma points[8][9]) through the nonlinear functions. It provides higher accuracy for strongly nonlinear systems.

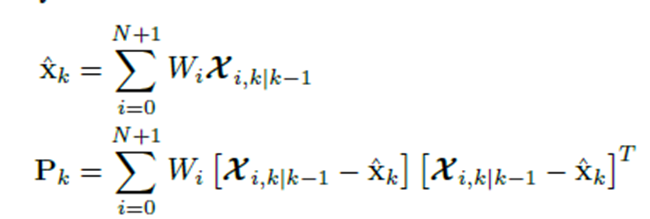
2. How it works



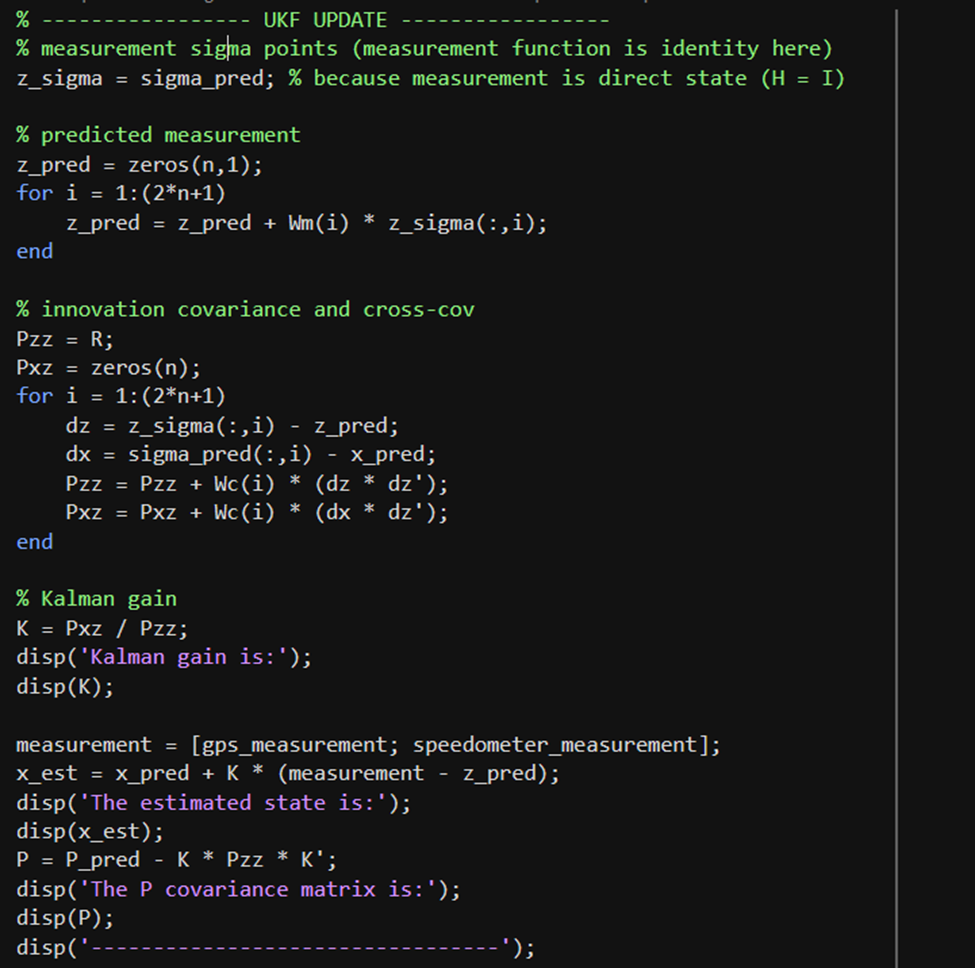
**Unscented Transform**

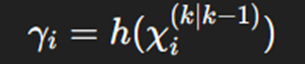
* Generates sigma points around current mean
* Propagates each sigma point through the non-linear functions f and h
* Recombine points to compute predicted mean and covariance

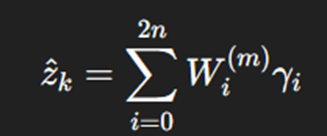


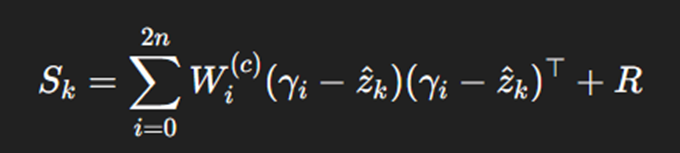
+Q

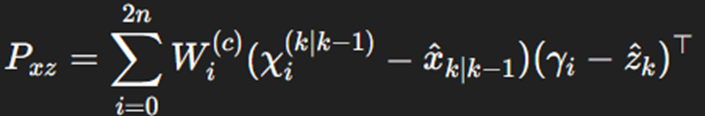
Pretty much like EKF, but it approximates the distribution by creating sigma points, pushes them through the nonlinear function, and sums them back into the mean and covariance.

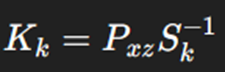


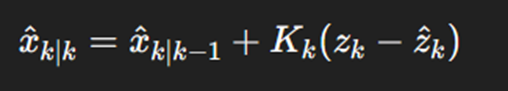


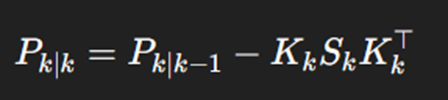












Same thing here

**Advantages**

* Handles **strong nonlinearities** better than EKF
* No Jacobians required
* More accurate than EKF in many practical cases

**Disadvantages**

* Slightly higher computational cost
* Complexity increases with **state dimension**

1. Proposed solution to the problem

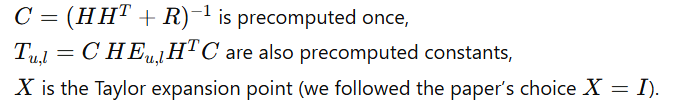
The Kalman filter requires evaluating



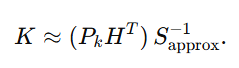
which is costly in hardware. Using a taylor approximation method[10] we can approximate this inverse using:



Where:



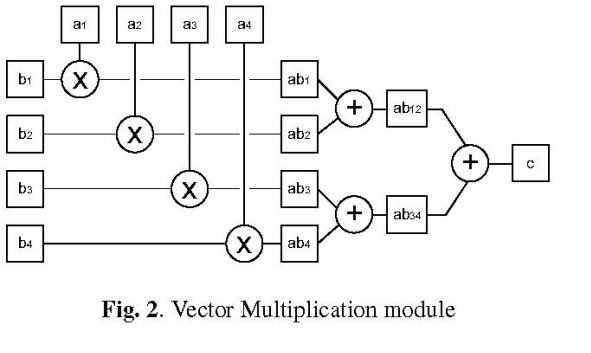
The Kalman gain is then approximated by:



This reduces the inversion operation to a series of scalar-weighted constant matrices, which is extremely FPGA-friendly.

Following the structure proposed in the paper, we organized our hardware into the following components:

* BRAM storage for constants
* Parallel matrix multiplication units using DSP slices
* Module for Weighted-sum
* Update and prediction pipeline



All computations are performed in fixed-point Q-format to keep the design efficient and resource-friendly.

**Processing pipeline:**

1. Compute the residual
2. Form
3. Compute the scalar weights
4. Build the correction matrix
5. Obtain the approximate inverse
6. Compute the gain
7. We update the state and covariance
8. Predict the next state and covariance with and

This structured pipeline allowed us to achieve deterministic latency and high throughput on the FPGA.

References:

[1] *A New Approach to Linear Filtering and Prediction Problems,* R.E. Kalman

[2] *A FPGA-based Unscented Kalman Filter for System-On-Chip Application,* Jeremy Soh, Xiaofeng Wu

[3] *FPGA implementation of multi-dimensional Kalman filter for object tracking and motion detection,* Praveenkumar Babu, Eswaran Parthasarathy

[4] *Adaptive Unscented Kalman Filter for Target Tracking with Unknown Time-Varying Noise Covariance,* Baoshuang Ge, Hai Zhang, Liuyang Jiang, Zheng Li, Maaz Mohammed Butt

[5] *Dual Extended Kalman Filter for the Identification of Time-Varying Human Manual Control Behavior,* Alexandru Popovici, Peter M. T. Zaal, Daan M. Pool

[6] *Kalman and Bayesian Filters in Python,* Roger R. Labbe Jr.

[7] *Reduced Sigma Point Filters for the Propagation of Means and Covariances Through Nonlinear Transformations,* Simon J. Julier, Jeffrey K. Uhlmann

[8] *Unscented Kalman Filter using Augmented State in the Presence of Additive Noise,* Fuming Sun, Guanglin Li, Jingli Wang

[9] *A Modular FPGA-based Implementation of the Unscented Kalman Filter,* Jeremy Soh, Xiaofeng Wu

[10] *Efficient Mapping of a Kalman Filter into an FPGA using Taylor Expansion,* Yang Liu, Christos-Savvas Bouganis, Peter Y K. Cheung

[11] NL State Estimation (Sigma Points) MAE 546 2017

Useful links:

* For Unscented Kalman Filters <https://medium.com/@simonleung5jobs/applying-the-unscented-kalman-filter-ukf-to-predict-stock-prices-8812e6511d7c>
* For Extended Kalaman Filter <https://simondlevy.github.io/ekf-tutorial/>