Bratosin Andrei - grupa 333AB
Pătrașcu Gabriel-Alexandru - grupa 333AB
Bicu David-Mihai - grupa 334AB

Computer Architecture project 2025: Kalman All the Way!

Introduction

The Kalman filter is a recursive method used for estimating unknown state variables based on current indirect measurements. It estimates the current state of a given linear system through a combination of an internal model of the system and external measurements, taking into account both process (w_k) and measurement (v_k) noise. Since its invention in 1960, it has found various real-world applications in domains such as signal processing, economics, and automatic control; famously, one of its very first large-scale applications was estimating the rocket's trajectory during the Apollo missions.

Over the past 50 years, as engineering ambitions have grown in tandem with computational resources, researchers have sought to extend the standard Kalman filter to nonlinear applications as well, having developed an entire class of state estimators based on Rudolf Kalman's original design. Of these, two have risen to prominence: the Extended Kalman Filter (EKF), and the Unscented Kalman Filter (UKF).

Historically, most Kalman filter implementations have been at the software level only. However, for many time-critical state estimation applications (SLAM, neural interfaces, etc.), such an approach has proven inadequate. As such, over the past couple of years, research interest in hardware-accelerated Kalman filter implementations, most targeting FPGAs (Field Programmable Gate Arrays) for their versatility and ease of use.

The goal of our project is to design hardware implementations for the Standard, Extended, and Unscented versions of the Kalman filter, targeting an Artix-7 FPGA. Following this introduction, we describe each of these three algorithms in detail, as well as a review of existing design strategies for each one. At the end of this document, we describe our own approach, which we will seek to implement in the next stages of the project.

Bratosin Andrei - grupa 333AB
Pătrașcu Gabriel-Alexandru - grupa 333AB
Bicu David-Mihai - grupa 334AB

Literature review

Conventional Kalman Filter (KF)

The Kalman Filter can be written in this state-space model:

$$x_{k+1} = Fx_k + w_k$$
$$z_k = Hx_k + v_k$$

where

- $x_k \in R^n$ is the state vector at the discrete moment k
- $F \in R^{n\,x\,n}$ is the system-dynamic matrix between 2 states at consecutive moments
- $z_k \in R^m$ is the measurement vector at the discrete moment k
- $H \in R^{m\,x\,n}$ is the measurement matrix that relates the measurements to the states of the system
- $w_k \in R^n$ - process noise (white noise with mean $\mu = 0$ and the covariance matrix Q_k known)
- $v_k \in R^m$ - measurement noise (white noise with mean $\mu = 0$ and the covariance matrix R_k known)

The task of the Kalman Filter is to estimate the state x_k of the system from the noise-pertubated measurements z_k through optimising the prediction of the state vector $x_{k+1}^{+} = x_{k+1|k}^{+}$.

The algorithm has 2 phases:

- Measurement *update* stage, when the parameters of the Kalman Filter are updated in the following sequence, where the K_k is the Kalman gain, x_k^{-} is the a priori state vector, x_k^{+} is the a posteriori state vector, P_k^{-} is the a priori estimation state covariance error matrix and P_k^{+} is the a posteriori estimation state covariance error matrix. The Kalman gain is the optimal gain that minimizes the expectation of the square of the magnitude of the error in the a posteriori state estimation $\mathbb{E}[\,||x_k - x_k^{+}||^2\,]$. By using the optimal Kalman gain, the update of the a

posteriori state estimation error covariance matrix (4) can be simplified to $P_k^+ = [I - K_k H]P_k^+$.

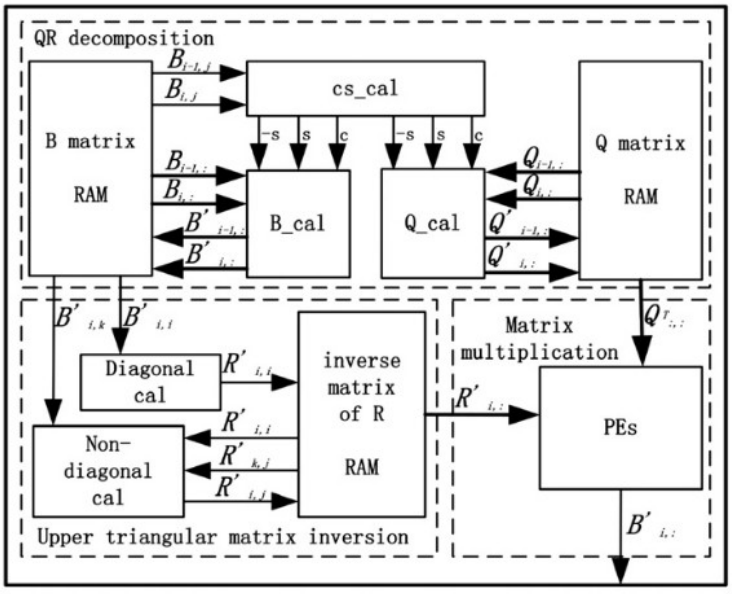
$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T [\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}]^{-1} \quad (2)$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-) \quad (3)$$

$$\mathbf{P}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}]\mathbf{P}_k^-[\mathbf{I} - \mathbf{K}_k\mathbf{H}]^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \quad (4)$$

- Prediction stage, where the future a priori state estimations are predicted by using the equations:
  - State: $x_k^- = Fx_{k-1}^-$
  - Covariance: $P_k^- = FP_{k-1}^- F^T + Q_k$

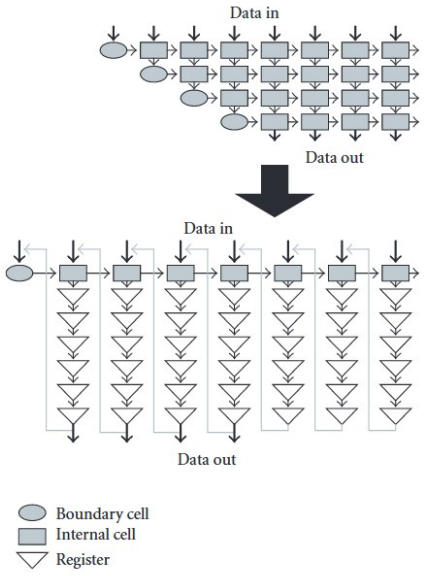
During the hardware implementation of the standard variant of the Kalman filter, one of the biggest problems that arises is the computation of the Kalman gain (K) at each step of the algorithm. Since it requires calculating the inverse of a matrix at each state update, it represents the most costly step of the Kalman filter algorithm. As such, circumventing this computational cost has been an important point of interest in hardware implementations.



Matrix inversion data flow using QR decomposition (fig 4 from [1])

The most common approach for calculating the inverse has been the *matrix decomposition method* (MDM) [1]. Thus, the algorithm computes the QR decomposition of the matrix to be inverted (usually through Givens rotations, as they tend to perform better in sparse matrices

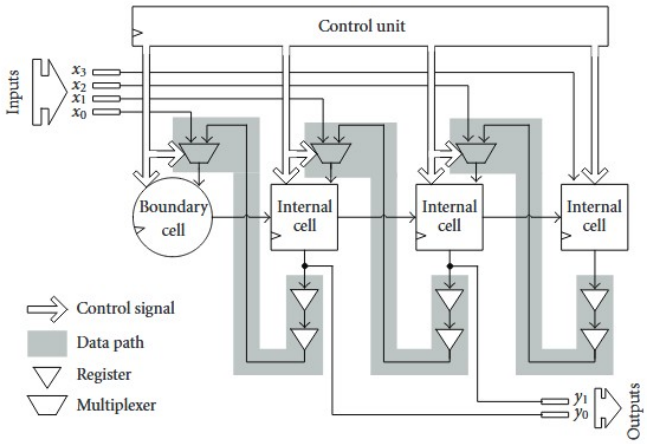
[7]), and the resulting upper triangular matrix R is inverted. Afterwards, computing the inverse is as simple as multiplying R^{-1} with Q^T. This is the approach employed by [1] and [2], and is generally considered the standard method.



Conversion from standard to pipelined systolic array form (fig. 1 from [3].)

One interesting approach we have come across has been to calculate the Kalman Gain with the use of a systolic array designed to calculate the Schur complement of a given 4-tuple of matrices (A,B,C,D), a technique first described in [3], and applied to an Extended Kalman implementation in [4]. They are then able to determine the state estimate at each step by reformulating all of the steps of the Kalman filter as Schur Complement calculations. Thus, each state estimate can be determined through 9 executions of the systolic array circuit (**see table**). For better performance, they implement a *pipelined* architecture instead of the standard systolic array form.

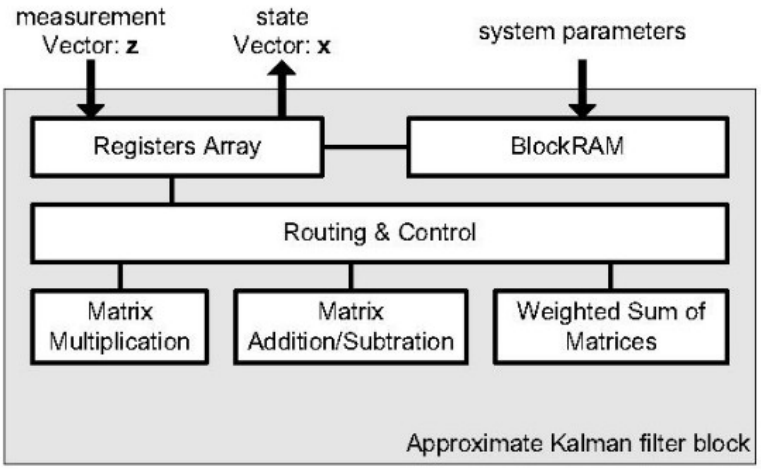
While [3] has found that this method performs significantly worse than an implementation of a standard MDM method, the systolic array structure still provides significant benefits in terms of ease of implementation and flexibility.



PSA structure block diagram (fig. 9 from [3].)

Another notable approach has been by [5]; in their paper, they seek to avoid calculating the exact Kalman gain entirely. Instead, through a series of simplifications, they deduce a formula

for an *approximate* Kalman gain, whose formula only requires the inverse of a matrix made up of constants, which can therefore be precomputed fetched from memory at each iteration. This new approach is called the Approximate Kalman Filter.



Approximate Kalman Filter block (fig. 1 in [5])

It is important to note that, owing to their approximations, the resulting Kalman gain is **no longer optimal**. However, because of the much simpler computation steps, their Approximate Kalman Filter can theoretically achieve similar convergence to algorithms which calculate the standard Kalman gain.

In their paper, Liu et al compare the performance of their method to an FPGA implementation of an UDU' factorization method initially described by [6], and find their algorithm to achieve better performance. However, the algorithm described in [5] isn't commonly used in such applications, and we have failed to find any subsequent papers which examine the algorithm's performance compared to more conventional FPGA approaches. Therefore, it's still a matter of debate whether the method's performance improvements justify the non-optimal choice for the Kalman gain.

Bratosin Andrei - grupa 333AB
Pătrașcu Gabriel-Alexandru - grupa 333AB
Bicu David-Mihai - grupa 334AB

Extended Kalman Filter (EKF)

The Extended Kalman Filter doesn't substantially change the equations of the Kalman Filter, compensating for the non-linear model by linearizing it in the current estimated state, applying the classic Kalman algorithm over the linearized model.

The nonlinear equations of the real system:

$$\dot{x} = f(x,u) + w_x$$
$$z = h(x) + w_z$$

We will use the Taylor series approximation of the functions f and h in the state x_k for the linear approximation of the system:

$$\dot{x} = x_k + \nabla_x f(x,u)|_{x=x_k}(x - x_k) + w_x$$
$$z = x_k + \nabla h(x)|_{x=x_k}(x - x_k) + w_z$$

From here, we apply the algorithm for the conventional Kalman Filter with these parameters:

EKF

$$\boxed{\mathbf{F} = \left.\frac{\partial f(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}}\right|_{\mathbf{x}_t, \mathbf{u}_t}}$$
$$\boxed{\mathbf{x} = f(\mathbf{x}, \mathbf{u})}$$
$$\mathbf{P} = \mathbf{F}\mathbf{P}\mathbf{F}^\mathsf{T} + \mathbf{Q}$$

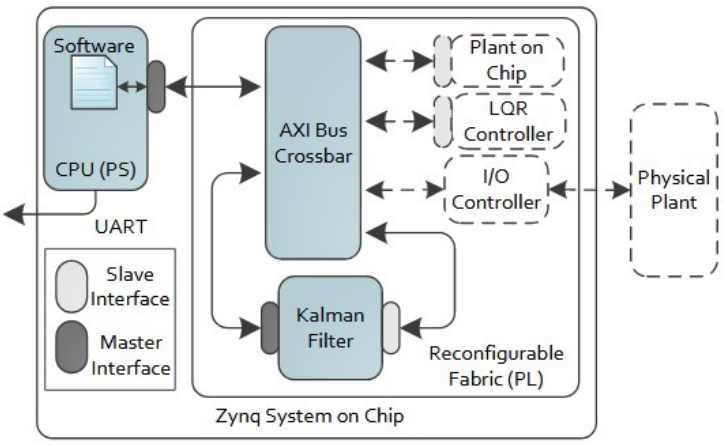
$$\boxed{\mathbf{H} = \left.\frac{\partial h(\bar{\mathbf{x}}_t)}{\partial \bar{\mathbf{x}}}\right|_{\bar{\mathbf{x}}_t}}$$
$$\mathbf{y} = \mathbf{z} - \boxed{h(\bar{x})}$$
$$\mathbf{K} = \mathbf{P}\mathbf{H}^\mathsf{T}(\mathbf{H}\mathbf{P}\mathbf{H}^\mathsf{T} + \mathbf{R})^{-1}$$
$$\mathbf{x} = \mathbf{x} + \mathbf{K}\mathbf{y}$$
$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}$$

Most hardware implementations of the Extended Kalman Filter employ a dual software-hardware approach: since the non-linear model is usually to implement directly at the hardware level, it's usually defined through software, whilst the generic, unchanging portions of the algorithm are implemented at the hardware level via an FPGA. For examples of such an implementation, see [2] [7]. While effective for most applications, such an approach introduces a significant communication overhead between the CPU and FPGA, and calculating the Jacobian of the nonlinear model at each step can be quite costly.

Bratosin Andrei - grupa 333AB
Pătrașcu Gabriel-Alexandru - grupa 333AB
Bicu David-Mihai - grupa 334AB

Step	Computation	Fadeev Input (M)
1	$T = AP^T$	$\begin{bmatrix} I & P^T \\ A & 0 \end{bmatrix}$
2	$P = AT^T + Q$	$\begin{bmatrix} I & T^T \\ A & Q \end{bmatrix}$
3	$T = CP^T$	$\begin{bmatrix} I & P^T \\ C & 0 \end{bmatrix}$
4	$K = CT^T + R$	$\begin{bmatrix} I & T^T \\ C & R \end{bmatrix}$
5	$K = TK^{-1}$	$\begin{bmatrix} K & I \\ T & 0 \end{bmatrix}$
6	$P = P - KT^T$	$\begin{bmatrix} I & T^T \\ -K & P \end{bmatrix}$
7	$T = Ax$	$\begin{bmatrix} I & x \\ A & 0 \end{bmatrix}$
8	$x = T + Bu$	$\begin{bmatrix} I & u \\ B & T \end{bmatrix}$
9	$T = Cx$	$\begin{bmatrix} I & x \\ C & 0 \end{bmatrix}$
10	$y = T + Du$	$\begin{bmatrix} I & u \\ D & T \end{bmatrix}$
11	$T = z - y$	$\begin{bmatrix} I & I \\ -y & z \end{bmatrix}$
12	$x = x + KT$	$\begin{bmatrix} I & T \\ K & x \end{bmatrix}$

Input schedule for the Fadeev systolic array block for a given state estimate in the PWAKF (Table 1 in [4].)



System architecture of the PWAKF system (fig. 1 in [4])

One novel approach to reduce communication overhead we found was the Piecewise Affine Kalman Filter (PWAKF) [4]. Instead of linearizing the system at every iteration, the software component precomputes the linearizations for *k* sub-intervals within a given range of values of x. Therefore, when the CPU receives state data from the FPGA, all it has to do is send the linearized model corresponding to the value interval to which the linearized model belongs.

Importantly, the CPU only sends an updated state whenever the linearized model has *changed*, thus eliminating unnecessary communication. This method provides greater performance, and the accuracy can be fine-tuned through careful selection of the linearization intervals.

Unscented Kalman Filter (UKF)

Starting from the original nonlinear system, we generate a set of sigma points for the states in the vicinity of the mean, each attributed with a first-order (W_j^a) and second-order weight (W_j^c) , that we propagate the points through the transfer function $f(x)$, from which we calculate the mean and the covariance in the predict step by the following equations:

$$\hat{\mathbf{x}}_{k|k-1} = \sum_{j=0}^{2L} W_j^a \mathbf{x}_j$$

$$\mathbf{P}_{k|k-1} = \sum_{j=0}^{2L} W_j^c \left(\mathbf{x}_j - \hat{\mathbf{x}}_{k|k-1}\right) \left(\mathbf{x}_j - \hat{\mathbf{x}}_{k|k-1}\right)^\mathsf{T} + \mathbf{Q}_k$$

In the update step, we propagate the sigma points through the measurement function h and we calculate the mean and covariance by the following equations:

$$\hat{\mathbf{z}} = \sum_{j=0}^{2L} W_j^a \mathbf{z}_j$$

$$\hat{\mathbf{S}}_k = \sum_{j=0}^{2L} W_j^c (\mathbf{z}_j - \hat{\mathbf{z}})(\mathbf{z}_j - \hat{\mathbf{z}})^\mathsf{T} + \mathbf{R_k}$$

and the Kalman gain is calculated using the cross-covariance matrix:

$$\mathbf{C_{xz}} = \sum_{j=0}^{2L} W_j^c (\mathbf{x}_j - \hat{\mathbf{x}}_{k|k-1})(\mathbf{z}_j - \hat{\mathbf{z}})^\mathsf{T}$$

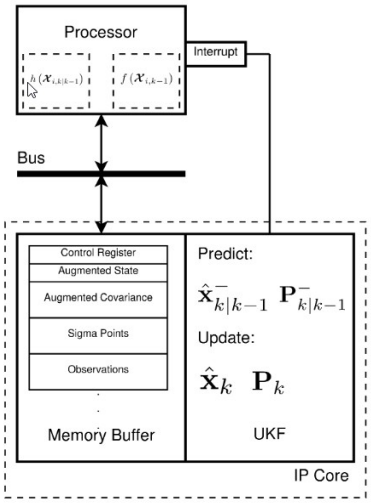
$$\mathbf{K}_k = \mathbf{C_{xz}} \hat{\mathbf{S}}_k^{-1}$$

and the aposteriori mean and covariance:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{z}})$$
$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k \hat{\mathbf{S}}_k \mathbf{K}_k^\mathsf{T}.$$

Similarly to the EKF, the Unscented Kalman Filter is also implemented through a dual hardware-software approach, in which the software holds the non-linear model functions [8]. Otherwise, implementing the rest of the UKF's functions poses no unique challenge compared to other Kalman filter algorithms, since its operations (such as computing the sigma points) are well suited to hardware acceleration.

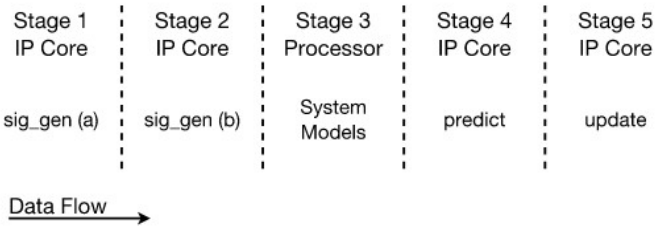
There is, however, a major point of divergence between hardware implementations of the UKF when it comes to the specific form of the covariance matrix being propagated through the algorithm's iterations. Whilst the standard UKF algorithm calls for the *P* matrix to be stored in its original form, there exist *square-root* variations of the UKF, which work directly with the square roots of the covariance matrices for better numerical stability [10].



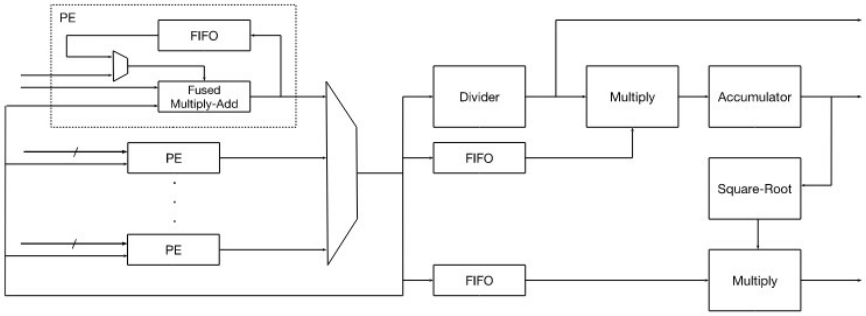
Architecture of the UKF implementation found in [9]

Of particular interest is the implementation proposed by [9]. In their paper, they suggest a mixed hardware-software approach in which communication between the processor and the FPGA is done through a shared bus interface, with an additional line designed for processor interrupts. The matrix square root calculation required by the algorithm is done through a standard Cholesky decomposition algorithm; these calculations are contained within the *trisolve* module of their architecture.

They expand on their work in [8], in which they conceptualize the algorithm through a data pipeline model (see below).



UKF data pipeline described in [9] (fig.2 in the original paper)



Datapath for the trisolve operation, fig. 4 in [8]

Our approach

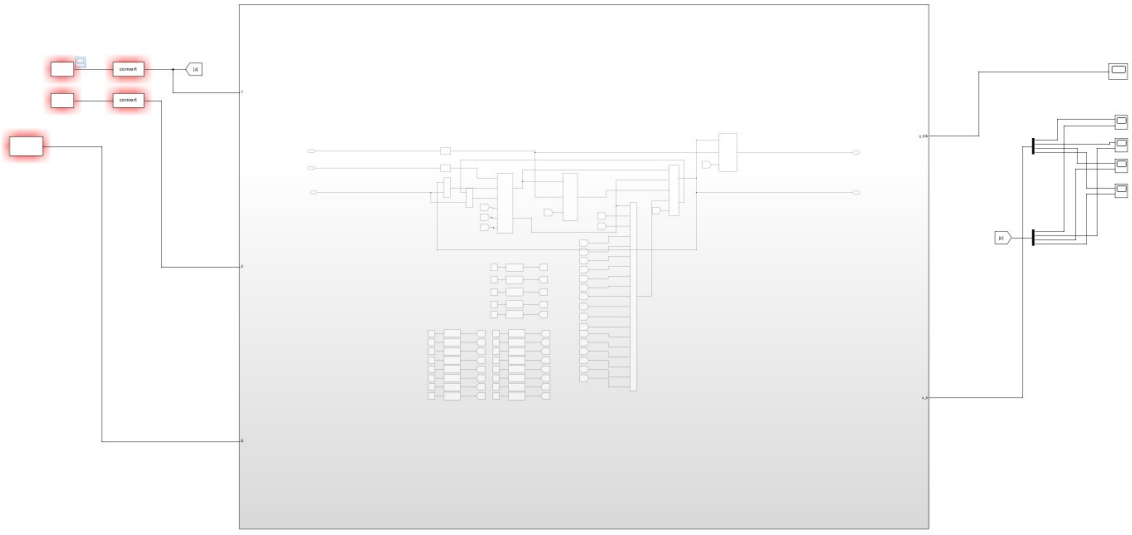
In the subsequent parts of the project, based on our review of the existing literature, we will seek to implement each of the three algorithms described above. Notably, we seek to implement each 3 of these entirely at the *hardware level, which also implies implementing a nonlinear system at the hardware level. Therefore:

- For the Standard Kalman filter, we will implement a pipelined systolic array structure that computes a Schur complement, based on the design first described in [3]. Despite the lower performance, such a structure will be easier to implement owing to the wealth of documentation available. Moreover, the flexibility of the design will allow us to test our design with systems of varying state sizes. This core structure we are going to maintain for the UKF and EKF implementations as well;
- For the Extended Kalman filter, we will opt for a Piecewise Affine implementation, based on the one described in [4]. Such an approach appeals to us owing to its high performance and ease of implementation. With a piecewise affine approach, we can precompute the required linearizations, forgoing the need for a direct hardware implementation of the derivatives and Jacobians of the *f* and *w* functions;
- For the Unscented Kalman Filter, we will also attempt a standard, hardware-only implementation. Aside from the UKF-specific components described in [8], our circuit will also contain a dedicated module that will execute the functions of a preset nonlinear system.

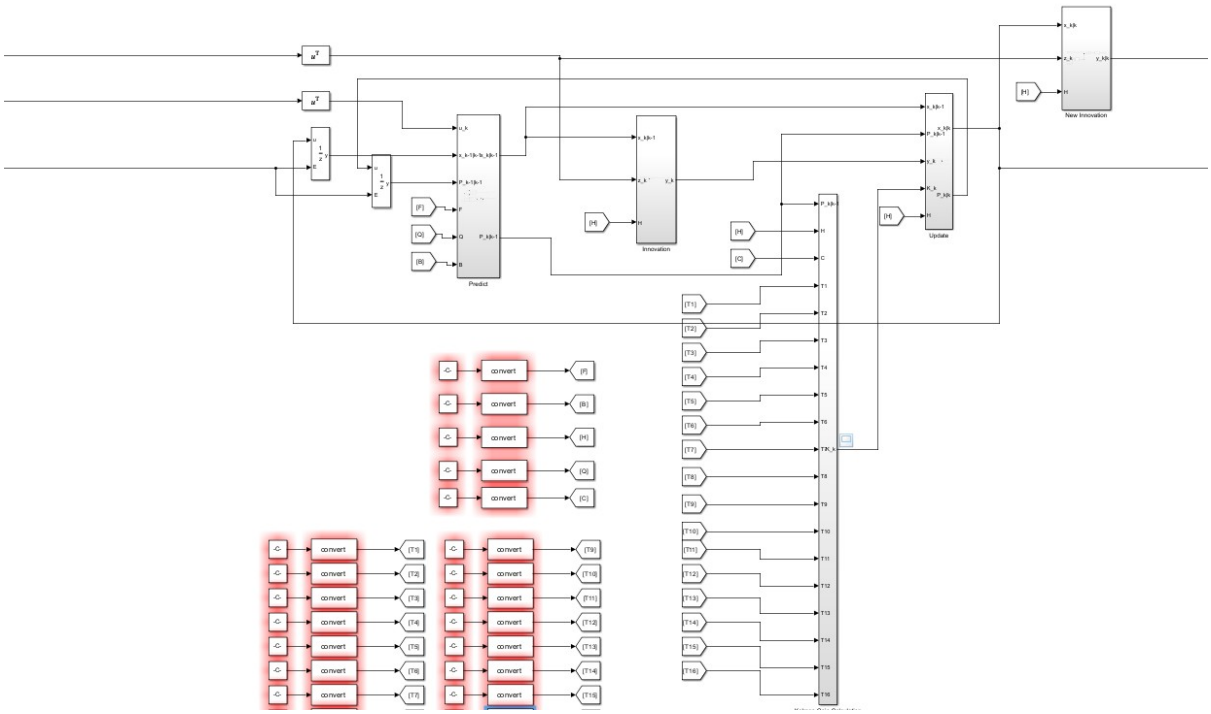
Evaluation Phase

Our initial attempt was to implement by using the pipelined systolic array structure, but we faced numerous difficulties regarding synchronization between the different nodes of the array, leading to garbage data being written at the array exit. Therefore, we decided to scale back our efforts and use a more straightforward approach for the Basic and Extended Kalman filters by using the linear Taylor approximation of the matrix inversion (as seen in [5]) and of the Jacobian in the case of the Extended Kalman Filter. Also, due to scheduling issues, we haven't implemented the Unscented Kalman Filter.

We implemented the 2 Kalman filters using subsystems in Simulink with the values of the Constant Blocks generated in the Workspace through separate MATLAB scripts. Then, we used the HDL Coder to generate the Verilog code from the subsystems, but we simulated the filters by using Simulink.

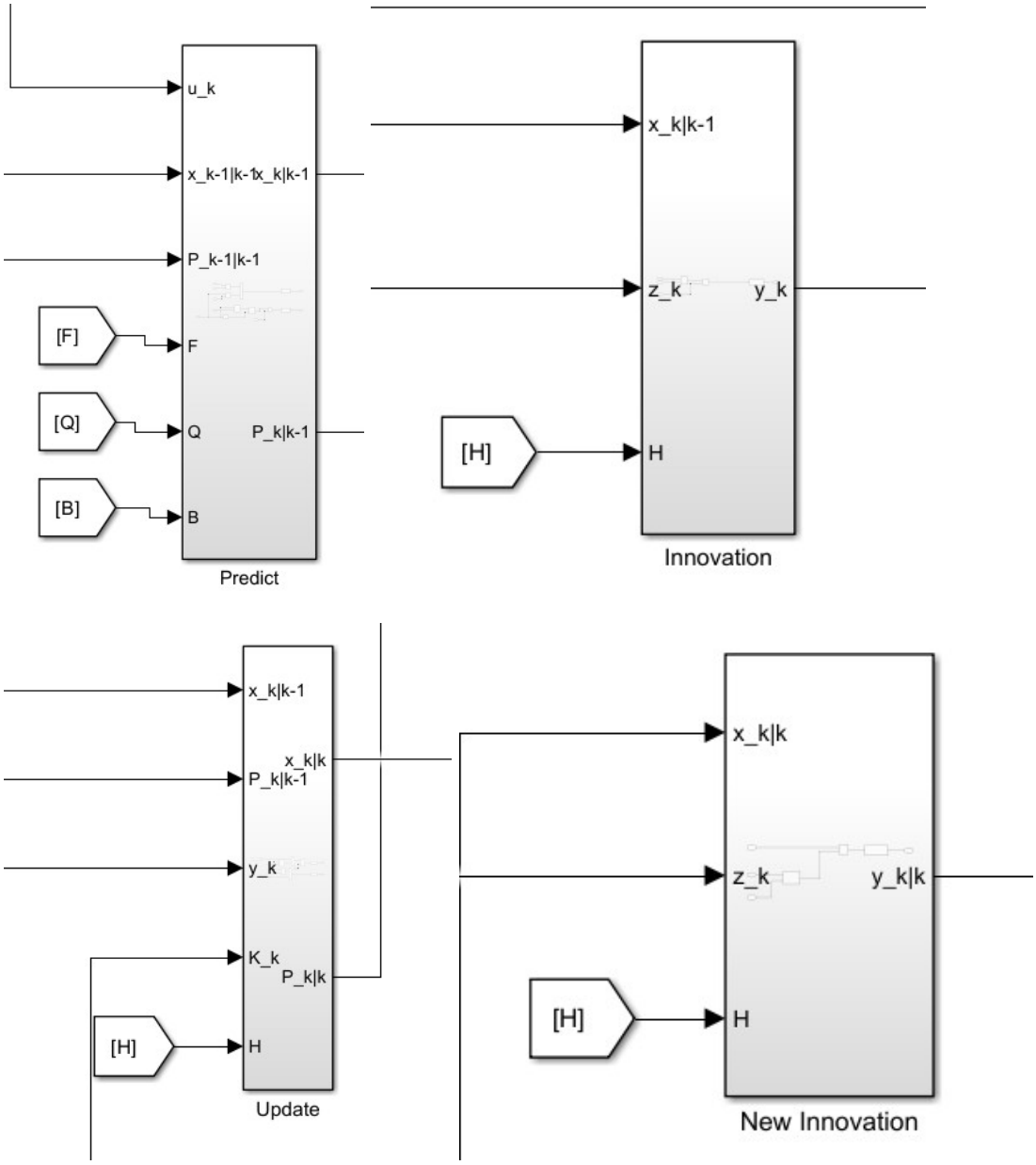


The main diagram of the basic Kalman Filter. The output is generated by the last 4 Scope blocks.



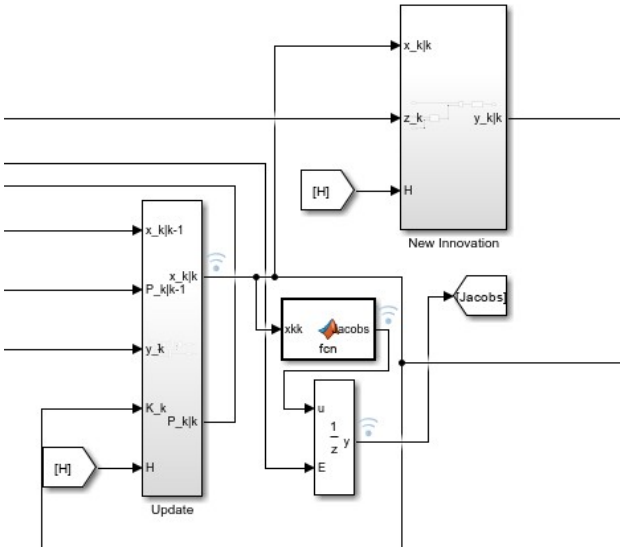
The subsystem of the Kalman Filter

The low-left blocks convert the values into FixedPoints, so that they are not susceptible to numerical errors. The bigger subsystems in the upper half follow a certain phase of the Kalman filter: Predict, Innovation, Update and another Innovation (*New Innovation*) before computing the final value. Also, the long subsystem in the low-right part computes the approximation of the matrix inverse by using the linear Taylor approximation.

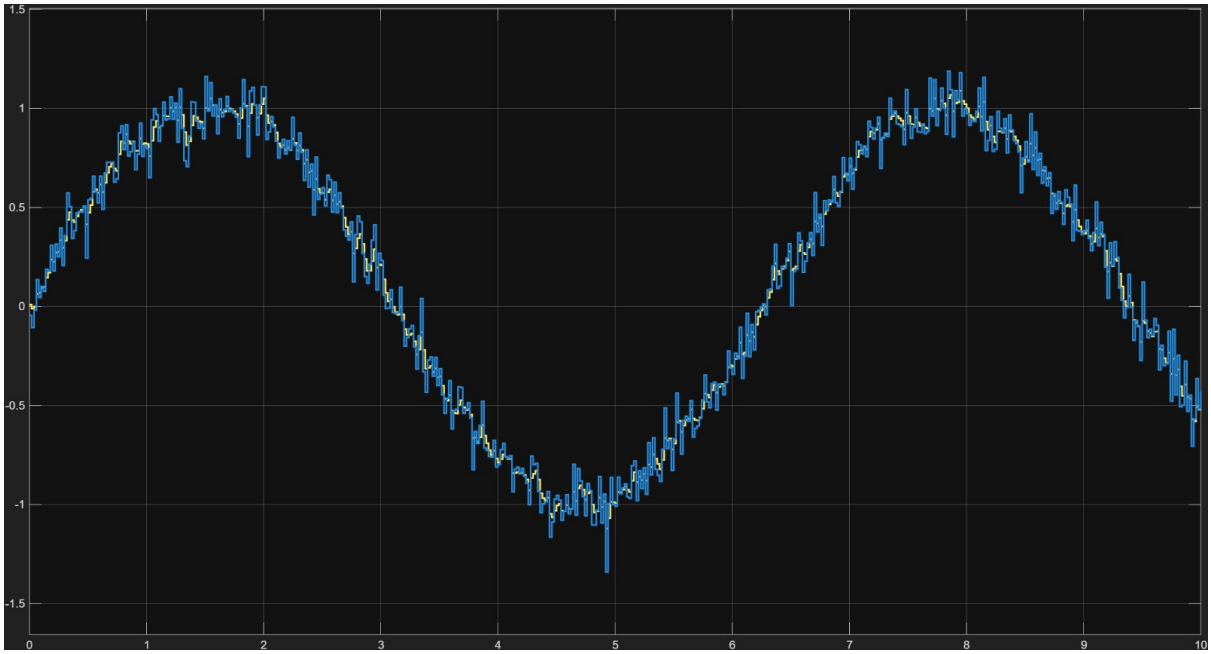


The Extended Kalman Filter was implemented in a similar approach with the main difference being computing the Jacobian matrix in the estimated state.

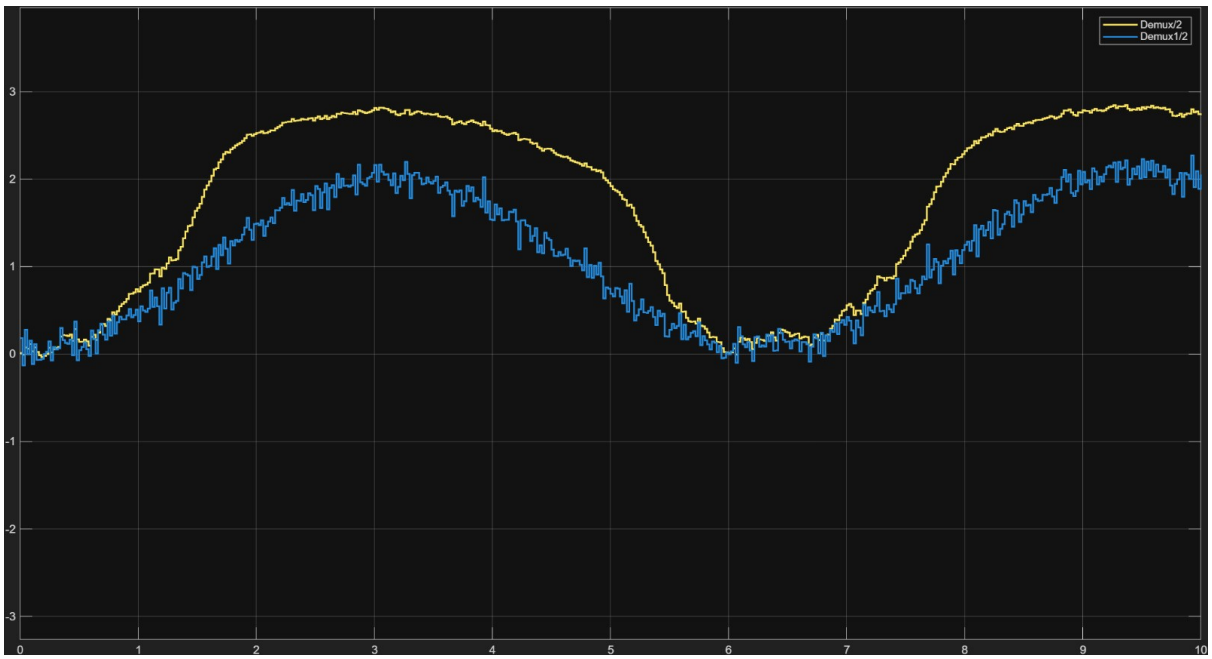
Bratosin Andrei - grupa 333AB
Pătrașcu Gabriel-Alexandru - grupa 333AB
Bicu David-Mihai - grupa 334AB



We used the 2 Kalman filters to estimate a sine wave. As can be seen from the graphs below, the basic Kalman filter follows the wave pretty closely, while the Extended Kalman Filter doesn't approximate the peaks very well, as it reaches a higher amplitude with a flat zone around the maximum points.



The approximation of the basic Kalman Filter (blue - original wave; yellow - the approximation)



The approximation of the Extended Kalman Filter (blue - original wave; yellow - the approximation)

Bibliography

[1] X. Zhu, R. Jiang, Y. Chen, S. Hu, și D. Wang, „FPGA implementation of Kalman filter for neural ensemble decoding of rat's motor cortex", *Neurocomputing*, vol. 74, nr. 17, pp. 2906-2913, oct. 2011, doi: 10.1016/j.neucom.2011.03.044.

[2] D. Pritsker, „Hybrid implementation of Extended Kalman Filter on an FPGA", în *2015 IEEE Radar Conference (RadarCon)*, Arlington, VA, USA: IEEE, mai 2015, pp. 0077-0082. doi: 10.1109/RADAR.2015.7130974.

[3] A. Bigdeli, M. Biglari-Abhari, Z. Salcic, și Y. Tin Lai, „A New Pipelined Systolic Array-Based Architecture for Matrix Inversion in FPGAs with Kalman Filter Case Study", *EURASIP J. Adv. Signal Process.*, vol. 2006, nr. 1, p. 089186, dec. 2006, doi: 10.1155/ASP/2006/89186.

[4] A. Mills, P. H. Jones, și J. Zambreno, „Parameterizable FPGA-Based Kalman Filter Coprocessor Using Piecewise Affine Modeling", în *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, IL, USA: IEEE, mai 2016, pp. 139-147. doi: 10.1109/IPDPSW.2016.101.

[5] Y. Liu, C.-S. Bouganis, și P. Y. K. Cheung, „Efficient Mapping of a Kalman Filter into an FPGA using Taylor Expansion", în *2007 International Conference on Field Programmable Logic and Applications*, Amsterdam, Netherlands: IEEE, aug. 2007, pp. 345-350. doi: 10.1109/FPL.2007.4380670.

[6] C. L. Thornton, J. Bierman, și S. Ieee, „Filtering and Error Analysis via the UDU' Covariance Factorization".
[7] G. H. Golub și C. F. Van Loan, *Matrix computations*, Fourth edition. în Johns Hopkins studies in the mathematical sciences. Baltimore: The Johns Hopkins University Press, 2013. (p. 293)
[8] J. Soh și X. Wu, „A Five-Stage Pipeline Architecture of the Unscented Kalman Filter for System-on-Chip Applications", *IEEE Trans. Ind. Electron.*, vol. 65, nr. 3, pp. 2785-2794, mar. 2018, doi: 10.1109/TIE.2017.2740844.
[9] J. Soh și X. Wu, „An FPGA-Based Unscented Kalman Filter for System-On-Chip Applications", *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 64, nr. 4, pp. 447-451, apr. 2017, doi: 10.1109/TCSII.2016.2565730.
[10] R. Dutt și A. Acharyya, „Low-Complexity Square-Root Unscented Kalman Filter Design Methodology", *Circuits Syst. Signal Process.*, vol. 42, nr. 11, pp. 6900-6928, nov. 2023, doi: 10.1007/s00034-023-02437-9.