EFFICIENT MAPPING OF A KALMAN FILTER INTO AN FPGA USING TAYLOR EXPANSION

Yang Liu, Christos-Savvas Bouganis, Peter Y. K. Cheung

Department of Electrical & Electronic Engineering
Imperial College London
Email: {yang.liu, christos-savvas.bouganis, p.cheung}@imperial.ac.uk

ABSTRACT

The Kalman filter is widely used as an estimator in many modern applications. In the case where its implementation in hardware is required, the computational complexity of the algorithm dictates the use of many resources. This paper presents an approximation of the conventional Kalman filter by using Taylor expansion and matrix calculus in order to remove the hardware expensive part of the algorithm. The Bierman-Thornton algorithm, as the exact counterpart of our proposed Approximate Kalman filter algorithm, is also implemented for comparison purposes. Comparing to the Bierman-Thornton algorithm, the FPGA implementation results demonstrate that our proposed Approximate Kalman filter implementation achieves one order of magnitude higher throughput using less hardware resources, obtaining similar convergence rate and accuracy.

1. INTRODUCTION

The Kalman filter is a recursive estimator that provides the best estimation, in a least-square error sense, of the "hidden" information of interest from noisy measurements. It was firstly introduced in the seminal papers [1, 2]. One of the first public known application was made at NASA to guide the Apollo 11 lunar module to the moon's surface. Since then, the Kalman Filter has found its way to become an indispensable part of many applications ranging from signal processing to communication and control. In many recent applications, a real-time execution of Kalman filter is required. SLAM [3] for autonomous vehicle localization and spatiotemporal saliency detection in video processing [4] are two examples. In [4], thousands of feature points are extracted from a video sequence and each one of them needs to be tracked by a Kalman filter. This implies that thousands of Kalman filter operations must be completed between video frames. When the target system is an embedded system, the real-time performance requirement forces the system designer to have a dedicated Kalman filter module to cope with such demanding computation.

A number of Kalman filter implementations have been proposed in the literature from the perspective of VLSI design [5, 6, 7, 8]. Yeh [5] implemented the conventional Kalman filter on a trapezoidal array and used the Faddeev algorithm [9] to execute matrix operations including matrix multiplication, matrix addition and matrix inversion. Other works resorted to the alternative least squares formulation and belong to the class of square-root Kalman filters. Also, systolic architectures have been proposed and a variety of implementations [6, 7, 8] are distinguished by the difference in data flow and the order of computations. In [6, 7, 8], the computational intensive matrix inversion from the conventional Kalman filter is replaced by an orthogonal transformation such as Given transformation for QR triangularization. Recent FPGA implementations [10, 11] are based on the conventional formulation of the Kalman filter. They are restricted to limited small problems that either have one state or use a specific type of matrix whose inverse can be expressed in a simple form. However, these approaches have a limited range of applications.

All of the aforementioned work focus on the theoretical proposition of the algorithm and architecture and do not give enough information on the actual implementation in hardware, let alone in a modern FPGA. The major contributions of this paper are: 1) We propose the use of Taylor expansion and matrix calculus to derive an approximate formulation of the Kalman filter. By approximating the matrix inversion, which is the most computationally demanding part in Kalman filter, the computational complexity is reduced significantly; 2) The Bierman-Thornton [12, 13] Kalman filter is also implemented for the first time in an FPGA and compared to the proposed algorithm.

The paper is organized as follows. Section 2 describes the conventional Kalman filter. Section 3 presents the derivation of the proposed Approximate Kalman filter along with the corresponding implementation. Section 4 briefly outlines the Bierman-Thornton Kalman filter and its implementation. Section 5 compares the performance of the two algo-

This work was funded by the UK Research Council under the Basic Technology Research Programme Reverse Engineering Human Visual Processes GR/R87642/02.

1-4244-1060-6/07/$25.00 ©2007 IEEE.

rithms in terms of convergence rate, accuracy, throughput and hardware resource requirements. Finally, the paper concludes in Section 6.

2. CONVENTIONAL KALMAN FILTER

It is not always feasible to measure all the variables of a dynamic system. Kalman filter provides a way to infer the missing variables, or states, from noisy measurements. It can also be used to predict future states taking into account all the measurements in the past. It must be noticed that Kalman filter is a *recursive* filter. The model is updated when a new measurement is available. However, it does not need to store all the measurements. A formulation of the Kalman filter is shown in (1).

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}\mathbf{x}_k + \mathbf{w}_k \\ \mathbf{z}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \end{aligned} \tag{1}$$

$\mathbf{x}_k$ is the n-dimensional state vector and $\mathbf{z}_k$ is the m-dimensional measurement vector at time k. The state $\mathbf{x}_k$ evolves in time and the transition is governed by the transition matrix $\mathbf{F}$, which is $n \times n$ in size. The $m \times n$ matrix $\mathbf{H}$ is called the measurement matrix and relates the measurements to the states. The system is corrupted by two types of noise. $\mathbf{w}$ is the n-dimensional process noise vector and $\mathbf{v}$ is the m-dimensional measurement noise vector. The noise vectors are assumed to be independent Gaussian processes with zero mean and covariance matrices $\mathbf{Q}$ and $\mathbf{R}$, respectively.

The task of a Kalman filter is to estimate or predict the state vector $\mathbf{x}_k$ from noise-perturbed measurements. At time k, given a new measurement vector $\mathbf{z}_k$, a Kalman filter computes the optimal prediction of the state vector $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_{k+1|k}$ based on known measurement vectors $\{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_k\}$.

Kalman filter can be interpreted as having two stages: *measurement update* and *prediction*. In the *measurement update* stage, the parameters of the Kalman filter are updated using (2) to (4).

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

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

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

$\mathbf{K}_k$ is the Kalman gain, $\hat{\mathbf{x}}_k^-$ is the *a priori* state vector, $\hat{\mathbf{x}}_k$ is the *a posteriori* state vector, $\mathbf{P}_k^-$ is the *a priori* state estimation error covariance matrix and $\mathbf{P}_k$ is the *a posteriori* state estimation error covariance matrix. The Kalman gain in (2) is the statically optimal gain. This Kalman gain minimizes $E\{\|\mathbf{x}_k - \hat{\mathbf{x}}_k\|^2\}$, which is the expected value of the square of the magnitude of the error in the posterior state estimation. By using the optimal gain in (2), the update of the state estimation error covariance matrix in (4) can be simplified as in (5).

$$\mathbf{P}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}]\mathbf{P}_k^- \tag{5}$$

In the *prediction* stage, the future states are predicted by using the projection equations:

$$\text{Project state:} \quad \hat{\mathbf{x}}_k^- = \mathbf{F}\hat{\mathbf{x}}_{k-1} \tag{6}$$

$$\text{Project covariance:} \quad \mathbf{P}_k^- = \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{Q} \tag{7}$$

More details on the derivations in this section can be found in [14].

3. APPROXIMATE KALMAN FILTER

To implement the conventional Kalman filter described in Section 2, one has to compute the matrix inversion in (2) in order to calculate the optimal gain. Matrix inversion constitutes the biggest obstacle in the integration of the Kalman filter on a silicon chip. That is because it is considerably more expensive than any other operations in the conventional Kalman filter. Hence, arises the motivation to avoid the matrix inversion with the main objective to decrease the implementation requirements. In [15, 16], the matrix inversion process is also avoided by manually computing the Kalman gain in advance. During the Kalman filtering operations, the pre-computed Kalman gains are fed into the process. However, this approach disregards all the information contained in the new measurement with respect to the gain.

In this section, we derive a solution that approximates the optimal Kalman gain in (2) by resorting to the Taylor expansion and matrix calculus. This solution significantly reduces the computational complexity by relieving us from performing the demanding matrix inversion. The drawback is a slower convergence rate than the conventional algorithm since the Kalman gain is not optimal anymore. However, the error due to the approximation is not accumulated since the Kalman filter parameters are properly updated.

3.1. Derivation

The matrix function we want to approximate in the conventional Kalman filter is $[\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + \mathbf{R}]^{-1}$ in (2). It has only one variable matrix $\mathbf{P}_k^-$, which is updated at each time step, whereas the other matrices are constant. Our goal is to approximate this matrix function using Taylor expansion. The Taylor series of a differentiable matrix-valued function $g(\mathbf{Y})$ around value $\mathbf{X}$ is given in (8), where $\overrightarrow{dg(\mathbf{X})}^{\mathbf{Y}-\mathbf{X}}$ is the directional derivative [17].

$$g(\mathbf{Y}) = g(\mathbf{X}) + \overrightarrow{dg(\mathbf{X})}^{\mathbf{Y}-\mathbf{X}} + \frac{1}{2!}\overrightarrow{dg^2(\mathbf{X})}^{\mathbf{Y}-\mathbf{X}} + ... \tag{8}$$

The first order approximation of (8) is given in (9).

$$g(\mathbf{Y}) \approx g(\mathbf{X}) + \overrightarrow{dg(\mathbf{X})}^{\mathbf{Y}-\mathbf{X}} \tag{9}$$

The directional derivative can be calculated from (10)

$$\overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg(\mathbf{X})} = \sum_{u,l} \frac{dg(\mathbf{X})}{d\mathbf{X}_{u,l}} (\mathbf{Y}-\mathbf{X})_{u,l} \quad (10)$$

where $\mathbf{X}_{u,l}$ denotes the scalar element on the u^{th} row and l^{th} column of matrix $\mathbf{X}$. Let $g(\mathbf{X}) = [\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R}]^{-1}$ where $\mathbf{X}$ is the variable matrix and $\mathbf{H}$ and $\mathbf{R}$ are both constant matrices. Thus,

$$\begin{aligned} \frac{dg(\mathbf{X})}{d\mathbf{X}_{u,l}} &= \frac{d([\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R}]^{-1})}{d\mathbf{X}_{u,l}} \\ &= -[\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R}]^{-1} \frac{d(\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R})}{d\mathbf{X}_{u,l}} [\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R}]^{-1} \\ &= -[\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R}]^{-1} \mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T [\mathbf{H}\mathbf{X}\mathbf{H}^T + \mathbf{R}]^{-1} \quad (11) \end{aligned}$$

where

$$\mathbf{E}_{u,l} = \frac{d\mathbf{X}}{d\mathbf{X}_{u,l}} = \begin{pmatrix} 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \\ & & l & \end{pmatrix} \quad u$$

is a zero matrix with 1 on the u^{th} row and l^{th} column. For $\mathbf{X} = \mathbf{I}$, where $\mathbf{I}$ is an identity matrix, (11) becomes

$$\left.\frac{dg(\mathbf{X})}{d\mathbf{X}_{u,l}}\right|_{\mathbf{X}=\mathbf{I}} = -[\mathbf{H}\mathbf{H}^T + \mathbf{R}]^{-1} \mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T [\mathbf{H}\mathbf{H}^T + \mathbf{R}]^{-1} \quad (12)$$

Let $\mathbf{C} = [\mathbf{H}\mathbf{H}^T + \mathbf{R}]^{-1} = g(\mathbf{X})|_{\mathbf{X}=\mathbf{I}}$. By substituting (12) in (10), we get (13).

$$\overset{\rightarrow \mathbf{Y}-\mathbf{X}}{dg(\mathbf{X})} = -\sum_{u,l} \Big[\big(\mathbf{C}\mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T\mathbf{C}\big)(\mathbf{Y}-\mathbf{X})_{u,l} \Big] \quad (13)$$

which can be interpreted as a weighted sum of matrices. The scalar weights are $(\mathbf{Y}-\mathbf{X})_{u,l}$ and the matrices are $\mathbf{C}\mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T\mathbf{C}$. Hence, the first-order Taylor approximation in (9) is given by (14).

$$g(\mathbf{Y}) = \mathbf{C} - \sum_{u,l} \Big[\big(\mathbf{C}\mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T\mathbf{C}\big)(\mathbf{Y}-\mathbf{I})_{u,l} \Big] \quad (14)$$

Thus, the approximate Kalman gain in (2) is given in (15).

$$\mathbf{K}_k^{\text{approx}} = \mathbf{P}_k^- \mathbf{H}^T \Big(\mathbf{C} - \sum_{u,l} \Big[\big(\mathbf{C}\mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T\mathbf{C}\big)(\mathbf{P}_k^- - \mathbf{I})_{u,l} \Big] \Big) \quad (15)$$

It is important to point out that due to the approximation, the state estimation error covariance matrix must be updated using (4) instead of the simplified (5). The derivation from (12) to (15) takes the Taylor expansion around $\mathbf{I}$. In practice, we can insert any *a priori* information matrix. As can be seen from the approximate Kalman gain in (15), we eliminate the need for matrix inversion that has to be computed at each step. $\mathbf{C} = [\mathbf{H}\mathbf{H}^T + \mathbf{R}]^{-1}$ is constant and can be precomputed, as well as the terms $\mathbf{C}\mathbf{H}\mathbf{E}_{u,l}\mathbf{H}^T\mathbf{C}$ in (15).

3.2. Hardware Implementation

Essentially, the Approximate Kalman filter system consists of the implementation of equations (15), (3), (4), (6) and (7). The high-level overview of the Approximate Kalman filter implementation is shown in Figure 1. It comprises the Matrix Multiplication module, the Matrix Addition/Subtraction module and the Weighted Sum of Matrices module. At the initialization stage, all the parameter matrices such as $\mathbf{F}$ and $\mathbf{H}$ are read from the blockRAM and written to a register array. Also, the matrix $\mathbf{C}$ and the constant terms in (15) are downloaded. This increases flexibility because the hardware does not need to be rebuilt for a different problem. Instead, new parameter matrices can be updated in the blockRAM and re-initialize the system for the corresponding register array to reflect the update. During one operation of the Approximate Kalman filter block, the measurement vector is fed into the block and the state vector is returned.

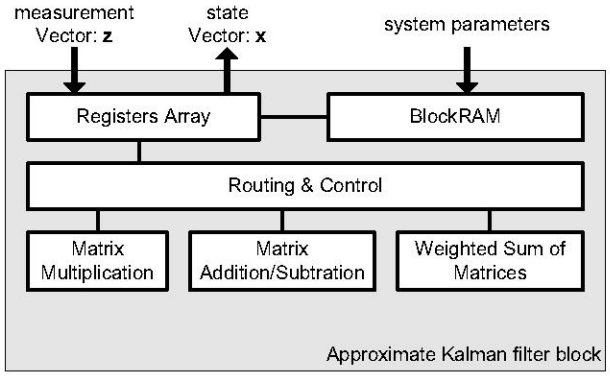

Fig. 1. Overview of the Approximate Kalman filter

The architecture of a vector multiplication module, which is the building block of the Matrix Multiplication module, is described below. To multiply a row vector $\mathbf{a} = [a_1, a_2, a_3, a_4]$ and a row vector $\mathbf{b} = [b_1, b_2, b_3, b_4]$, a vector multiplication module that compute the scalar $c = \mathbf{a}\mathbf{b}^T$ is shown in Figure 2.

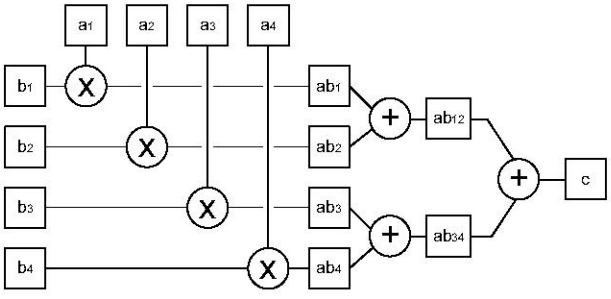

Fig. 2. Vector Multiplication module

The Matrix Multiplication module consists of many Vector Multiplication modules. It is important to execute ma-

trix multiplication rapidly because this operation is predominant in the Approximate Kalman filter. All the vector multiplications are executed in parallel in order to maximize speed. Similarly, the Matrix Addition/Subtraction module is formed by an array of adder/subtractors. Each adder/subtractor performs matrix element addition/subtraction in parallel to the others.

The Weighted Sum of Matrices module is shown in Figure 3 and computes $\sum_{u,l}\left[(\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C})(\mathbf{P}_k^- - \mathbf{I})_{u,l}\right]$ in (15). The rest of the operations in (15) are performed by the matrix multiplication module and matrix addition/subtraction module. In Figure 3, $\mathbf{T}_{u,l}$ represents $\mathbf{CHE}_{u,l}\mathbf{H}^T\mathbf{C}$, which is an array of constant matrices. These matrices are precomputed and loaded to registers from blockRAM at initialization. $\mathbf{t}_{u,l}$ represents the variable scalar weight $(\mathbf{P}_k^- - \mathbf{I})_{u,l}$. The $\mathbf{T}_{u,l}$ and the corresponding $\mathbf{t}_{u,l}$ are multiplexed to the Matrix scalar multiplication module, which multiply the $\mathbf{t}_{u,l}$ to every element of $\mathbf{T}_{u,l}$. The summation is achieved through accumulating the result of matrix additions to variable $\mathbf{T}_{\text{out}}$, which eventually contains the weighted sum of the matrices.

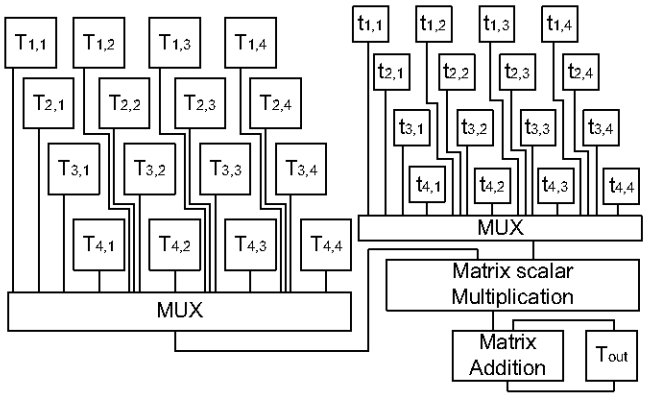

Fig. 3. The Weighted Sum of Matrices module

4. BIERMAN-THORNTON UD KALMAN FILTER

Although the conventional Kalman filter in Section 2 can be implemented directly in hardware, it is clear that using an iterative method for matrix inversion is particularly sensitive to round-off errors and likely to cause numerical problems. Square-root forms of the Kalman filter are more robust numerically because the triangular square roots are propagated to preserve the symmetry of the covariance matrices in the presence of round-off errors [14]. A square-root Kalman filter takes the "square-root", the Cholesky factor, of the state estimation error covariance matrix $\mathbf{P} = \mathbf{L}\mathbf{L}^T$. A set of equations is derived to update the factor $\mathbf{L}$ instead of the covariance matrix $\mathbf{P}$ itself. In practice, mathematically equivalent implementation methods can have very different numerical stability at the same precision. Bierman-Thornton UD Kalman filter [12, 13] belongs to the square-root Kalman filter framework and it consists of a pair of algorithms, the Bierman algorithm and the Thornton algorithm. It should be noted that the Bierman-Thornton algorithm computes the *optimal* Kalman gain, thus it is not an approximation to the conventional Kalman filter.

4.1. Bierman-Thornton algorithm

The Bierman algorithm executes the *measurement update* of the modified Cholesky factor $\mathbf{U}$ and $\mathbf{D}$ of the covariance matrix $\mathbf{P} = \mathbf{U}\mathbf{D}\mathbf{U}^T$, where $\mathbf{U}$ is a unit[1] upper triangular matrix and $\mathbf{D}$ is a diagonal matrix. It takes in a scalar element of m-dimensional measurement vector $\mathbf{z}$ one at a time, and uses the corresponding vector from the measurement matrix $\mathbf{H}$ to update the state vector $\mathbf{x}$, as well as $\mathbf{U}$ and $\mathbf{D}$. Therefore, it needs m iterations to execute the measurement update.

The Thornton algorithm [13] is responsible for the *prediction* of $\mathbf{x}$, $\mathbf{U}$ and $\mathbf{D}$, which is equivalent to the *prediction* stage of the conventional Kalman filter. A detailed description of the Bierman-Thornton algorithm can be found in [14].

4.2. Hardware implementation

The interaction between the Bierman module and the Thornton module is shown in Figure 4. The scalar values from the measurement vector $\mathbf{z}$, along with the associated system parameters from $\mathbf{H}$ and measurement error covariance matrix $\mathbf{R}$, are multiplexed into the Bierman module. After the Bierman module processes the entire measurement vector, the results are ready to be read by the Thornton module. These results are $\mathbf{x}$, $\mathbf{U}$ and $\mathbf{D}$. The Thornton algorithm returns the *predicted* $\mathbf{x}$,$\mathbf{U}$,$\mathbf{D}$, which would be the input for the Bierman module at the next time step.

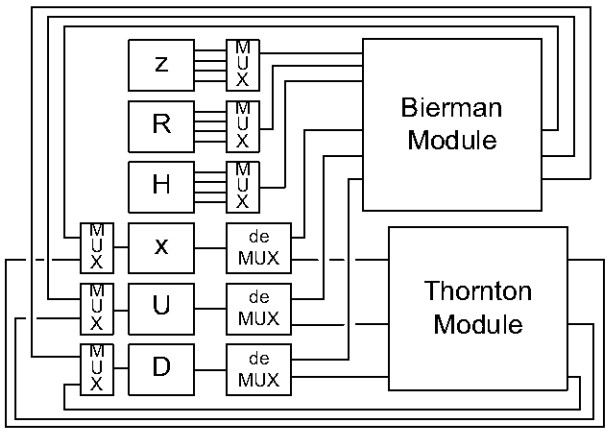

Fig. 4. Overview of the Bierman-Thronton Kalman filter system

[1] A unit triangular matrix is a triangular matrix with 1 on the diagonal

5. PERFORMANCE EVALUATION

The proposed Approximate Kalman filter and the Bierman-Thornton Kalman filter are implemented and their performance is compared. The target device is the Xilinx® Virtex-4 FPGA chip XC4VFX140 with package FF1517 and speed grade 11. The hardware is designed using Handel-C by Celoxica®. The dimension of the state vector is referred to as n and the dimension of the measurement vector is referred to as m. Unless stated otherwise, $n = 4$ and $m = 4$ throughout the experiment section. This problem size can be found in many practical applications [3, 4].

5.1. Convergence Rate

To test the convergence rate of the two Kalman filter implementations, a Kalman filter system with a constant state is used. This setup can show how fast the *estimated* states from the two Kalman filters converge to the *true* state from the noisy measurements. Figure 5 shows that the Approximate Kalman filter does converge to the true value but not as fast as the Bierman-Thornton algorithm. This is expected as the Kalman gain in our algorithm is not optimal due to the approximation. However, it follows the Bierman-Thornton algorithm quite closely.

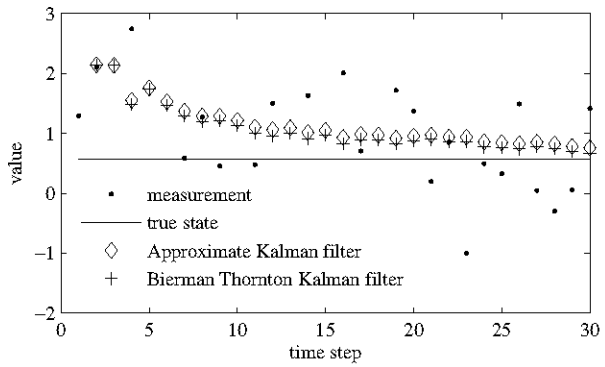

Fig. 5. The convergence rate of the two Kalman filters

5.2. Accuracy

To test the accuracy of the two implementations, a dynamic system is used, in which the state vector changes with time. A uniform wordlength is used throughout the system. Without loss of generality, half of the bits were used for the sign and integer part, and the other half were used for the fractional part. Figure 6 shows the impact on the average square error of the estimated state vectors by varying the wordlength. The square error of a state vector is defined as in (16).

$$\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^{\text{true}} - \mathbf{x}_i^{\text{estimated}}\right)^2 \qquad (16)$$

where $\mathbf{x}_i^{\text{true}}$ and $\mathbf{x}_i^{\text{estimated}}$ are the i^{th} scalar element in the *true* and *estimated* state vectors. For each wordlength value, both Kalman filter implementations were run with 1000 measurement vectors. As reference, the error from a conventional Kalman filter algorithm implemented in software with double floating point precision is also shown in Figure 6. The large error of the Bierman-Thornton implementation below 18-bits is due to overflow/underflow. The input data are the same for both implementations. It can be concluded that the the approximation Kalman filter implementation requires less wordlength to avoid overflow/underflow than the Bierman-Thornton algorithm.

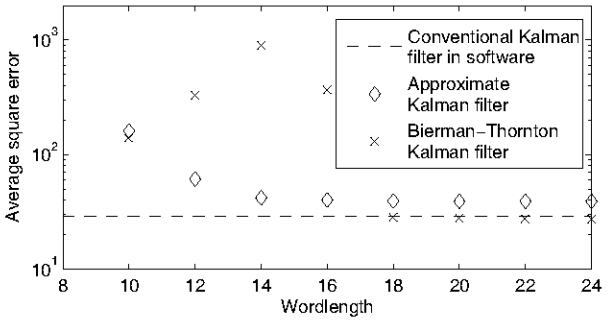

Fig. 6. Accuracy versus wordlength

5.3. Area

Figure 7 provides the information on how the area, in terms of FPGA slices, varies with wordlength. In this comparison, the DSP48 blocks are not used. Both the Bierman-Thornton and the approximate Kalman filter implementations use 9 blockRAMs in the FPGA. As illustrated in Figure 7, both implementations scale linearly with wordlength.

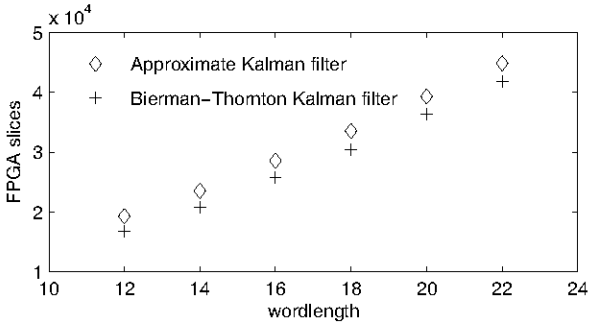

Fig. 7. Area comparison in FPGA slices

Table 1 shows the hardware resource usages for Kalman filters for different problem sizes, to which both n and m are equal. The "no DSP" column corresponds to the number of occupied slices when the DSP48 blocks in Virtex-4 were not used. Only at the problem size of 4, the approximate Kalman filter implementation is slightly larger than the Bierman-Thornton one, which is also reflected in Figure 7. It can be concluded that the Approximate Kalman filter implementation scales better than the Bierman-Thornton Kalman filter implementation as the dimension of the measurement and state vector increases.

Problem size	Approximate			Bierman-Thornton		
	with DSP48		no DSP	with DSP48		no DSP
	slices	DSP	slices	slices	DSP	slices
3	3848	24	5848	4728	40	6854
4	12302	96	19323	9019	110	16783
5	13668	150	25972	18357	190	32575
6	18707	162	34583	32897	192	47407

Table 1. Hardware resource usage for 12-bits systems. Available resources: 192 DSP48s, 63168 slices.

5.4. Throughput

Throughput is a metric that indicates how fast the Kalman filter hardware can process measurements and return state vectors. In this paper, throughput is defined as the number of state vectors estimated per second. The FPGA implementations use 16-bits in this subsection. For comparison, a software Kalman filter employing the conventional formulation described in Section 2 is included. The conventional Kalman filter was implemented in C++ at double floating point precision using OpenCV library [18] and was run on a Pentium 4 PC with 2.4GHz CPU and 1GB of RAM. The results are summarized in Table 2. It can be concluded that the Approximate Kalman filter FPGA implementation has around 10 times larger throughput than the Bierman-Thronton Kalman filter FPGA implementation which is in turn faster than the software implementation by one order of magnitude. Since the Approximate Kalman filter uses less hardware resources as demonstrated in Section 5.3, it is evident that our proposed algorithm is a better design choice.

Kalman Filter Algorithm	Latency (Clock cycle)	Maximum frequency	Throughput
Approximate	78	41.6MHz	532966
Bierman-Thornton	578	39.1 MHz	63087
Conventional in C++	N/A	N/A	2567

Table 2. Throughput comparison for 16-bits implementations

6. CONCLUSION

In this paper, we propose a new architecture for the Kalman filter, which reduces the computational complexity of a conventional Kalman filter by approximating the matrix inversion function using Taylor expansion. We implemented both the proposed Approximate Kalman filter and the Bierman-Thornton Kalman filter in a modern FPGA. The results demonstrate that the proposed Approximate Kalman filter implementation has significantly larger throughput using less hardware resources, achieving at the same time similar accuracy and convergence rate.

7. REFERENCES

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[2] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," *Transactions of the ASME - Journal of Basic Engineering*, vol. 83, pp. 95–107, 1961.

[3] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229 – 241, 2001.

[4] Y. Liu, C.-S. Bouganis, and P. Cheung, "A Spatiotemporal Saliency Framework," in *2006 IEEE International Conference on Image Processing*, 2006, pp. 437–440.

[5] H.-G. Yeh, "Systolic implementation on Kalman filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 36, no. 9, pp. 1514 – 1517, Sept. 1988.

[6] T. Sung and Y. Hu, "Parallel VLSI implementation of the Kalman filter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-23, no. 2, pp. 215 – 224, 1987.

[7] S.-Y. Kung and J.-N. Hwang, "Systolic array designs for Kalman filtering," *IEEE Transactions on Signal Processing*, vol. 39, no. 1, pp. 171 – 182, 1991.

[8] G. Irwin and F. Gaston, "A systolic architecture for square root covariance Kalman filtering," *Proceedings of the International Conference on Systolic Arrays*, pp. 255 – 263, 1989.

[9] D. K. Faddeev, V. N. Faddeeva, and R. C. Williams, *Computational methods of linear algebra*. Freeman, 1963.

[10] C. Lee and Z. Salcic, "High-performance FPGA-based implementation of Kalman filter," *Microprocessors and Microsystems*, vol. 21, no. 4, pp. 257 – 265, 1997.

[11] R. Turney, A. Reza, and J. Delva, "FPGA implementation of adaptive temporal Kalman filter for real time video filtering," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 2231 – 2234, 1999.

[12] G. Bierman, *Factorization methods for discrete sequential estimation*. Academic Press, 1977.

[13] C. L. Thornton, "Triangular covariance factorizations for kalman filtering," Ph.D. dissertation, University of California at Los Angeles, School of Engineering, 1976.

[14] M. S. Grewal and A. P. Andrews, *Kalman filtering : theory and practice using MATLAB*, 2nd ed. New York ; Chichester: Wiley, 2001.

[15] S. Kiaei and U. B. Desai, "Independent data flow wavefront array processors for recursive equations." in *Proceedings of the Twentieth Conference on Information Sciences and Systems.*, Princeton, NJ, USA, 1986, pp. 429–435.

[16] D. Massicotte, "A systolic VLSI implementation of Kalman-filter-based algorithms for signal reconstruction," in *Proceedings of the 1998 IEEE ICASSP.*, vol. 5, Seattle, WA, USA, 1998, pp. 3029 – 3032.

[17] J. Dattorro, *Convex Optimization & Euclidean Distance Geometry*. Meboo Publishing USA, 2005.

[18] http://opencvlibrary.sourceforge.net.