A FLOATING-POINT EXTENDED KALMAN FILTER IMPLEMENTATION FOR AUTONOMOUS MOBILE ROBOTS

Vanderlei Bonato, Eduardo Marques [*]

Institute of Mathematical and Computing Sciences
The University of São Paulo
São Carlos, BR.
email: vbonato, emarques@icmc.usp.br

George A. Constantinides [†]

Department of Electrical and Electronic Engineering
Imperial College London
London, U.K.
email: g.constantinides@imperial.ac.uk

ABSTRACT

Localization and Mapping are two of the most important capabilities for autonomous mobile robots and have been receiving considerable attention from the scientific computing community over the last 10 years. One of the most efficient methods to address these problems is based on the use of the Extended Kalman Filter (EKF). The EKF simultaneously estimates a model of the environment (map) and the position of the robot based on odometric and exteroceptive sensor information. As this algorithm demands a considerable amount of computation, it is usually executed on high end PCs coupled to the robot. In this work we present an FPGA-based architecture for the EKF algorithm that is capable of processing two-dimensional maps containing up to 1.8k features at real time (14Hz) and is two orders of magnitude more power efficient than a general purpose processor.

1. INTRODUCTION

Mobile robotics is a very active research field that has been investigated for more than two decades. Its goal is develop intelligent machines capable of acting autonomously in complex environments. Localization and mapping, which are used to calculate the robot position inside its navigation environment and to create a representation of this environment, are two of the most important tasks to be performed by mobile robots [1]. Most solutions for these problems are based on probabilistic inferences derived from Bayes filters [2] involving high computational complexity and a large volume of data.

In most cases, these algorithms are implemented on personal computers, and can not be directly applied to mobile robots [3]. Embedding these algorithms on chip is usually desired when the typical solution of a laptop mounted on the robot is unsatisfactory, such as when the robotic system has power constraints [4].

This paper presents an FPGA-based implementation of the EKF algorithm [5] related to simultaneous localization and mapping (SLAM) problem. The main contributions of this work are: it is the first FPGA-based architecture for the EKF-based SLAM problem; presents an analysis of the computational complexity and memory bandwidth requirements for the FPGA-based EKF and shows that our proposed architecture updates feature-based maps in real time with $3\times$ more features than a Pentium M 1.6GHz processor, while consuming only 1.3% of the processor power.

The paper is organized as follows. Section 2 presents the EKF equations and a complexity analysis is derived. The proposed architecture is presented in Section 3, and then Section 4 shows some experimental results. Finally, Section 5 concludes the paper.

2. EKF COMPUTATION COMPLEXITY ANALYSIS

This section presents the EKF equations along with a complexity analysis related to the number of floating-point operations. A complete EKF algorithm description can be found in [1].

It is well known that the computational requirements for EKF algorithm in SLAM is $\Theta(n^2)$, where n represents the number of features [1]. However to better understand how this computational complexity is distributed between its equations, we present in Table 1 an analysis of the number of floating-point operations for each EKF equation used in our proposed architecture (See in Table 2 the variables used in the prediction and update EKF equations along with their dimensions). As can be noticed, the highest complexity is located in equation (5), since as all elements of the covariance matrix, which has a high dimension given by $(r+sn)\times(r+sn)$, must be evaluated and updated for each iteration. Consequently in this particular equation there is

[*]Ack. to CAPES (Ref. BEX2683/06-7)
[†]Ack. to EPSRC (Grant EP/C549481/1 and EP/C512596/1)

Table 1. Number of floating-point operations for each EKF equation when $r = 3$ and $s = 2$, where n corresponds to the number of features and r and s are the robot and feature state size, respectively.

Equations	FLOP
(1); (2); (9); (10); (3)	9; 189; $30n$; 2; $36n + 97$
(4); (11); (5)	$36n + 93$; $8n + 21$; $16n^2 + 60n + 54$
Total	$16n^2 + 170n + 465$

Table 2. The description and dimension of the EKF symbols, where s and r represent the feature and robot state size, i the feature number and n the total number of features.

Symbols	Dimension	Description
v	$r \times 1$	Robot position
f	$s \times 1$	Feature position
μ	$(r + sn) \times 1$	Both robot and feature positions
μ_v	$r \times 1$	Elements of μ for robot position
μ_f	$sn \times 1$	Elements of μ for feature position
Σ_{vv}	$r \times r$	Robot position covariance
Σ_{vf}	$r \times (sn)$	Cross robot-feature covariance
Σ_{ff}	$(sn) \times (sn)$	Cross feature-feature covariance
Σ	$(r + sn)\times$ $(r + sn)$	Cross robot-feature and feature-feature covariance
α	-	Prediction function
γ	-	Measurement function
u	$r \times 1$	Robot motion command
F	$r \times r$	Robot motion Jacobian
G	$r \times r$	Robot motion noise Jacobian
Q	$r \times r$	Permanent motion noise
H_v	$s \times r$	Measurement Jacobian; respect to μ_v
H_{fi}	$s \times s$	Measurement Jacobian; respect to μ_{fi}
H	$s \times (r + sn)$	Compounded measurement Jacobian
R	$s \times s$	Permanent measurement noise
W	$(r + sn) \times s$	Filter gain
ν	$s \times 1$	Mean innovation
z	$s \times 1$	Sensor measurement
z_{pred}	$s \times 1$	Sensor measurement prediction
S	$s \times s$	Covariance innovation
Z_1	$s \times (s(i-1))$	Zero Matrix
Z_2	$s \times (s(n-i))$	Zero Matrix

not only a large number of floating-point operations, but also a large memory bandwidth requirement to access the covariance matrix. Therefore, both aspects must be considered in order to develop a high performance system. Another important consideration is related to the matrix $H^{(t)}$ that has dimension $s \times (r + sn)$ and is used to multiply the covariance matrix that has dimension $(r+sn)\times(r+sn)$. As given in equation (7), $H^{(t)}$ is a sparse structured matrix. Taking advantage of this structure, the total EKF complexity can be reduced from $48n^2 + 202n + 255$ to $16n^2 + 170n + 465$, as given in Table 1.

Prediction:

$$\mu_v^{(t)} = \alpha(v^{(t-1)}, u^{(t)}) \quad (1)$$

$$\Sigma_{vv}^{(t)} = F^{(t)}\Sigma_{vv}^{(t-1)}F^{(t)T} + G^{(t)}QG^{(t)T} \quad (2)$$

$$\Sigma_{vf}^{(t)} = F^{(t)}\Sigma_{vf}^{(t-1)} \quad (3)$$

Update:

$$\mu^{(t)} = \bar{\mu}^{(t)} + W^{(t)}\nu^{(t)} \quad (4)$$

$$\Sigma^{(t)} = \bar{\Sigma}^{(t)} - W^{(t)}S^{(t)}W^{(t)T} \quad (5)$$

where:

$$\bar{\mu}^{(t)} = \begin{bmatrix} \mu_v^{(t)} \\ \mu_f^{(t-1)} \end{bmatrix}, \bar{\Sigma}^{(t)} = \begin{bmatrix} \Sigma_{vv}^{(t)} & \Sigma_{vf}^{(t)} \\ \Sigma_{vf}^{(t)T} & \Sigma_{ff}^{(t-1)} \end{bmatrix} \quad (6)$$

$$H^{(t)} = \begin{bmatrix} H_v^{(t)} & Z_1 & H_{fi}^{(t)} & Z_2 \end{bmatrix} \quad (7)$$

$$z_{pred}^{(t)} = \gamma(\mu_v^{(t)}, \mu_{fi}^{(t-1)}) \quad (8)$$

$$\nu^{(t)} = z^{(t)} - z_{pred}^{(t)} \quad (9)$$

$$S^{(t)} = H^{(t)}\bar{\Sigma}^{(t)}H^{(t)T} + R \quad (10)$$

$$W^{(t)} = \bar{\Sigma}^{(t)}H^{(t)T}S^{(t)-1} \quad (11)$$

3. HARDWARE ARCHITECTURE

This section presents a hardware architecture specially developed to compute the prediction and update phases of the EKF algorithm applied to the SLAM problem. To implement this architecture we consider that the prediction and measurement functions along with their Jacobian matrices are computed separately and their results are stored in the on chip memories. As can be noticed from the previous section, the main characteristics of this algorithm are: most operations are matrix multiplication, addition and subtraction, and particularly in equation (5) a large quantity of data must be read and updated in the covariance matrix. Guided by these dominant characteristics and also by the desired system performance, we propose an FPGA-based architecture illustrated in Fig. 1, which is composed of four external memory banks, a set of on chip memories, a state machine and four Processing Elements (PEs).

To define the required performance for this architecture we consider its application in a SLAM system that has only monocular cameras as the exteroceptive sensors, which is the case in our mobile robotics project [6]. In this SLAM system the depth information of the features are obtained by using the well-known triangulation technique [7]. Therefore, the cameras, which are fixed on the robot base, must be moved to capture images from different positions. Considering that the robot navigates in a straight line with maximum speed of 1m/s and that the triangulation works with images captured every 70mm, we have the EKF epoch frequency defined by: $1000/70 = 14.28$Hz. Hence, the required performance can be estimated by combining this information with the requirement of 1.5k features to be processed by the EKF algorithm in real time.

For 1.5k features and r and s equal to 3 and 2, respectively, the covariance matrix dimension $\Sigma^{(t)}$ is 3003×3003 and the mean vector $\mu^{(t)}$ is 3003×1. In this implementation the data are represented in single precision floating-point format (32bits), so these matrices store a total equal to 36MB. These data are stored in an external memory and that

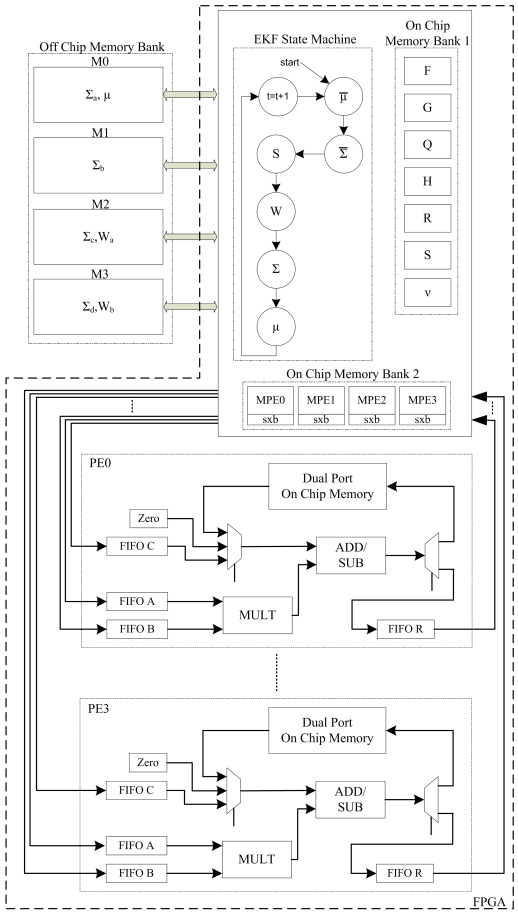

Fig. 1. EKF architecture; **MPEx** are on chip memories for PEs data reusability and **b** the PEs buffer size; other symbols are presented in Table 2.

they must be read and written at a frequency of 14Hz. As a result, the external memory bandwidth is 1GB/s. In order to avoid bottlenecks the proposed architecture distributes the data between 4 external memories. The access control to these memories is implemented by a state machine, where each memory bank is accessed in parallel. This state machine also controls the data flow between the on chip memories and the PEs, exploits the data reuse inside the FPGA using the on chip memory bank (MPEs), computes the inversion of matrix S and controls the iteration among the EKF equations.

3.1. Processing Element

The purpose of the PEs in the EKF architecture is to compute the three most common operations of the EKF algorithm, which are $R = AB$; $R = C + AB$ and $R = C - AB$, where A,B,C and R are block partitions of matrices. These operations are implemented in a such way that data reuse in the code can be exploited, mitigating the off chip memory access bottleneck, and FPGA parallelism can be efficiently explored.

In this architecture matrices of size $N \times M$ are partitioned into blocks. Consider three matrices A, B and C where A multiplies B and the result is added to C. Each block is composed of $1 \times M$ elements of matrix A, $N \times 8$ elements of matrix B and 1×8 elements of matrix C. The constant 8 corresponds to the number of words that can be stored in the on chip memory of each PE for data reuse. In our proposed PE the size 8 was chosen as it results in a good tradeoff between on chip memory size and off chip memory bandwidth requirement for our target platform. In the proposed architecture each matrix block is processed by a single PE, hence 4 blocks can be concurrently processed.

In the PE architecture the data of matrix A is reused 8 times. However, whenever matrix B has fewer than 8 columns the computation is re-ordered in order take full advantage of pipeline depth. In the EKF algorithm, this happens in the operations used to calculate the filter gain W; the transformation is represented by $R = (B^T A^T)^T$, $R = (C^T + B^T A^T)^T$ or $R = (C^T - B^T A^T)^T$, and they are easily done in the state machine by changing the data order sent to the FIFOs. Finally, in situations where matrix A has fewer than 8 rows and matrix B fewer than 8 columns, it is necessary to fill this gap with dummy data and then reject the corresponding results. In the EKF algorithm this occurs only in equation (2).

4. RESULTS

This section presents some experimental results related to the EKF architecture described in this paper, in particular the hardware resources employed, its performance and power consumption. The architecture is described in the Handel-C language and it has been validated in the Celoxica RC250 development kit, featuring an EP2S90F1020C4 FPGA and 4 external SRAM memory banks.

The resource utilization and the maximal clock frequency of a single PE is shown in Table 3. As can be noticed, 70% of the PE resources are used by the floating-point units (MULT, ADD) and the remaining 30% by the control logic. These floating-point units are from the Celoxica library, and are single precision based on the IEEE754 standard. Although the MULT unit has an operating frequency higher than the ADD unit, the overall PE frequency is limited by the ADD as they are working from a common clock. The minimal PE latency is 4 and the maximum depends on the PE operations and the matrix size.

Table 4 presents the resources for the whole architecture, which includes the external memory controller, EKF state machine, embedded memories and the 4 PEs. The most part of the resources in column $SM + EMC$ is used to implement the EKF state machine. The state machine is relatively

Table 3. FPGA resources for a single PE.

EP2S90F1020C4	MULT	ADD/SUB	Entire PE
Clock (MHz)	156	97	94
Latency	3	3	4(minimal)
ALUTs	599	1078	2368
Registers	517	546	1492
DSP blocks (9bit)	8	0	8

Table 4. FPGA resources for the whole EKF architecture; where SM is the EKF State Machine and EMC the External Memory Controller.

EP2S90F1020C4	4 PEs	SM+EMC	Total
Clock (MHz)	90	70	70
ALUTs	9252	8868	18120 (25%)
Registers	5332	2985	8317
DSP blocks (9bit)	32	0	32 (8%)
Memory bits	9216	3392	12608 (1%)

large as it has to repeatedly partition each matrix between the PEs and combine the results again. Moreover, it contains a floating-point division unit used to calculate the S matrix inversion. Although the PE has higher frequency than the state machine, the whole system clock rate is limited by the state machine clock as it is responsible for sending data to and reading from the PE FIFOs.

4.1. Performance and Power Analyses

We can analyse the system performance at the achieved clock frequency of 70MHz, the EKF prediction and update frequency 14Hz and the algorithm complexity presented in Table 1, where n represents the number of features. In this implementation each PE computes 2 floating-point operations (MULT and ADD) per clock cycle and as there are 4 PEs the system has peak floating-point performance of 560MFLOPs. However, the average performance is slightly inferior to these figures for the following reasons: first, the FIFOs are both flushed whenever there is a transition from one matrix operation to another and second, in some clock cycles the state machine does not send data to the FIFOs as a consequence of internal loop controls. The first overhead is a constant and the bigger the matrices are the smaller the influence average performance. However, the second overhead is a proportional value that reduces the average performance by approximately 3%. Thus the maximum number of features that can be processed in real time (14Hz) is approximately 1.8k. The power consumption of the EKF system was estimated by the PowerPlay Power Analyzer from Quartus II tool. The estimated power using signal activities generated by probabilistic methods is 1.3W.

To compare these results with another technology we consider the Pentium M 1.6GHz processor as it is commonly found in on-board mobile robot cards, such as in the Pioneer 3DX platform [8]. The performance achieved running the EKF implemented in C on this processor is 572 features in real time and the power consumption of this processor, according to its datasheet, is 31.1W. Thus, comparing these solutions, we can notice that the FPGA processes $3\times$ more features in real time and also consumes two orders of magnitude less energy than the Pentium M.

5. CONCLUSION

We have presented in this paper a computational complexity analysis and an FPGA-based architecture for the EKF algorithm applied to the SLAM problem. The analysis demonstrated that for hardware implementation, both an efficient floating-point matrix multiplication and a high external memory bandwidth are required. On the Celoxica RC250 platform, we have demonstrated that FPGAs are a suitable technology to solve this problem as the matrix multiplication can be accelerated by exploiting parallelism, while the off chip memory bandwidth can be improved through access to parallel memory banks. As a result, the architecture we implemented on the FPGA has both a higher performance and a significantly less energy consumption than the general purpose processor commonly adopted to solve this problem.

6. REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.

[2] D. Fox, J. Hightower, L. Liao, D. Schultz, and G. Borriello, "Bayesian filters for location estimation," *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 24–33, 2003.

[3] P. Pirjanian, N. Karlsson, L. Goncalves, and E. D. Bernardo, "Low-cost visual localization and mapping for consumer robotics," *Industrial Robot*, vol. 30, no. 2, pp. 139–144, 2003.

[4] B. Thomas, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Berlin and Heidelberg: Springer-Verlag, 2003.

[5] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*. New York: Springer-Verlag, 1990, pp. 167–193.

[6] V. Bonato, J. A. d. Holanda, and E. Marques, "An embedded multi-camera system for simultaneous localization and mapping," in *Proceedings of Applied Reconfigurable Computing, Lecture Notes on Computer Science - LNCS 3985*. Springer-Verlag, 2006, pp. 109–114.

[7] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[8] ActivMedia Robotics, "Technical specifications," 2006. [Online]. Available: http://www.activrobots.com/ROBOTS/p2dx.html