Journal of VLSI Signal Processing manuscript No.
(will be inserted by the editor)

A Floating-point Extended Kalman Filter Implementation for Autonomous Mobile Robots

Vanderlei Bonato[1], Eduardo Marques[1], George A. Constantinides[2]

[1] Institute of Mathematical and Computing Sciences
The University of São Paulo
São Carlos - BR
e-mail: vbonato@icmc.usp.br
[2] Department of Electrical and Electronic Engineering
Imperial College London
London - U.K.

Received: 24 November 2007 / Revised version: 17 July 2008

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), a three-fold improvement over a Pentium M 1.6GHz, and a 13-fold improvement over an ARM920T 200MHz. The proposed architecture also consumes only 1.3% of the Pentium and 12.3% of the ARM energy per feature.

Keywords Mobile Robotics, SLAM, EKF, FPGA

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 Bayesian filters [2] [3] involving high computational complexity and a large volume of data.

In most cases, these algorithms are implemented on personal computers, and cannot be directly applied to mobile robots [4]. 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 [5].

This paper presents an FPGA-based implementation of the EKF algorithm [6] related to simultaneous localization and mapping (SLAM) problem. The main contributions of this work are:

- as far we know, it is the first FPGA-based architecture for the EKF-based SLAM problem;
- it presents an analysis of the computational complexity and memory bandwidth requirements for FPGA-based EKF;
- the results show that our proposed architecture updates feature-based maps with three times more features than a Pentium M 1.6GHz processor is capable of in real time, and thirteen times more than an ARM920T 200MHz, while consuming only 1.3% of the Pentium and 12.3% of the ARM energy per feature.

The paper is organized as follows. Section 2 briefly describes the EKF algorithm and a complexity analysis is derived. Section 3 presents some related work. The proposed architecture is presented in Section 4, and then Section 5 shows some experimental results. Finally, Section 6 concludes the paper.

2 Extended Kalman Filter Algorithm

This section briefly describes the EKF algorithm along with a complexity analysis related to the number of

floating-point operations. A complete EKF algorithm description can be found in [1]. EKF is composed of two phases: prediction and update. In the SLAM context, the prediction phase estimates the robot position $\mu_v^{(t)}$, at time t, based on a prior believed position $\mu_v^{(t-1)}$ and on a movement control $u^{(t)}$, while the update phase integrates robot sensor observations $z^{(t)}$ in order to update a map of the robot environment and to again estimate the robot position. These two steps are repeated for each EKF iteration, where the data estimated at one iteration are used as input to the next one.

In this paper the robot position comprises of two-dimensional planar coordinates (x, y) relative to some external coordinate frame, along with its angular orientation θ. The map consists of a set of feature positions $(\mu_{f_1}^{(t)}, \mu_{f_2}^{(t)}, ..., \mu_{f_n}^{(t)})$ detected from the robot navigation environment by a sensor, where each feature is represented in the same way by its (x, y) coordinates. Thus the robot state size is three and the feature state size is two; these parameters are represented in this paper by r and s, respectively, for generalization. As these robot and feature states are estimated, they have an associated covariance matrix in order to represent their uncertainty, which is represented by $\Sigma^{(t)}$ at time t. In the EKF algorithm these data are organized as in (1), where $\mu^{(t)}$ is composed of the estimated robot position and the feature set at time t.

$$\mu^{(t)} = \begin{bmatrix} \mu_v^{(t)} \\ \mu_{f_1}^{(t)} \\ \mu_{f_2}^{(t)} \\ \vdots \\ \mu_{f_n}^{(t)} \end{bmatrix}, \Sigma^{(t)} = \begin{bmatrix} \Sigma_{vv}^{(t)} & \Sigma_{vf_1}^{(t)} & \dots & \Sigma_{vf_n}^{(t)} \\ \Sigma_{f_1v}^{(t)} & \Sigma_{f_1f_1}^{(t)} & \dots & \Sigma_{f_1f_n}^{(t)} \\ \Sigma_{f_2v}^{(t)} & \Sigma_{f_2f_1}^{(t)} & \dots & \Sigma_{f_2f_n}^{(t)} \\ \vdots & \vdots & \ddots & \\ \Sigma_{f_nv}^{(t)} & \Sigma_{f_nf_1}^{(t)} & \dots & \Sigma_{f_nf_n}^{(t)} \end{bmatrix} \quad (1)$$

Table 1 describes the variables used in the prediction and update EKF equations along with their dimensions. Equations (2) and (3) are used to estimate the new robot position given the belief vector $\mu_v^{(t-1)}$ and the covariance matrix that correspond to the robot position $\Sigma_{vv}^{(t-1)}$ and the current motion control $u^{(t)}$. $F^{(t)}$ and $G^{(t)}$ are Jacobian matrices containing derivatives of the prediction function α with respect to the motion command variables at time t. Equation (4) estimates the covariance between the robot and feature position Σ_{vf} given the corresponding covariance from time $(t-1)$ and the matrix $F^{(t)}$.

After computing the prediction equations, the update step starts by predicting the sensor measurement through the measurement function equation (9) using the estimated robot position $\mu_v^{(t)}$ and the detected feature $\mu_{fi}^{(t-1)}$. Then equations (10) and (11) calculate the innovation related to the measurement $\nu^{(t)}$ and covariance $S^{(t)}$, respectively. $\nu^{(t)}$ is the difference between the real and the estimated sensor measurement and $S^{(t)}$ is

Table 1 The description and dimension of the EKF symbols, where s and r represent the feature and robot state size, v and f the robot and feature position, i the feature number and n the total number of features.

Sym.	Dimension	Description
μ	$(r+sn)\times 1$	Both robot and feature positions
μ_v	$r \times 1$	Elements of μ related to robot position
μ_f	$sn \times 1$	Elements of μ related to 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 with respect to v
H_{fi}	$s \times s$	Measurement Jacobian with respect to f_i
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

the new information added to the system covariance given the current matrix $H^{(t)}$ and the covariance from the previous prediction phase. $H^{(t)}$ is a matrix that compounds two Jacobian matrices $H_v^{(t)}$ and $H_{fi}^{(t)}$ from (8) which are derivatives of the prediction function γ with respect to the estimated robot position and the detected feature at time t. Then, equation (12) computes the filter weight. Finally, equations (5) and (6) update all data that corresponds to the estimated map.

Prediction:

$$\mu_v^{(t)} = \alpha(\mu_v^{(t-1)}, u^{(t)}) \quad (2)$$

$$\Sigma_{vv}^{(t)} = F^{(t)} \Sigma_{vv}^{(t-1)} F^{(t)T} + G^{(t)} Q G^{(t)T} \quad (3)$$

$$\Sigma_{vf}^{(t)} = F^{(t)} \Sigma_{vf}^{(t-1)} \quad (4)$$

Update:

$$\mu^{(\mathbf{t})} = \bar{\mu}^{(t)} + W^{(t)} \nu^{(t)} \quad (5)$$

$$\Sigma^{(t)} = \bar{\Sigma}^{(t)} - W^{(t)} S^{(t)} W^{(t)T} \quad (6)$$

where:

$$\bar{\mu}^{(t)} = \begin{bmatrix} \mu_v^{(t)} \\ \mu_f^{(t-1)} \end{bmatrix}, \bar{\Sigma}^{(t)} = \begin{bmatrix} \Sigma_{vv}^{(t)} & \Sigma_{vf}^{(t)} \\ \Sigma_{vf}^{(t)T} & \Sigma_{ff}^{(t-1)} \end{bmatrix} \quad (7)$$

$$H^{(t)} = \begin{bmatrix} H_v^{(t)} \; Z_1 \; H_{fi}^{(t)} \; Z_2 \end{bmatrix} \quad (8)$$

$$z_{pred}^{(t)} = \gamma(\mu_v^{(t)}, \mu_{fi}^{(t-1)}) \quad (9)$$

$$\nu^{(t)} = z^{(t)} - z_{pred}^{(t)} \quad (10)$$

$$S^{(t)} = H^{(t)} \bar{\Sigma}^{(t)} H^{(t)T} + R \quad (11)$$

$$W^{(t)} = \bar{\Sigma}^{(t)} H^{(t)T} S^{(t)^{-1}} \quad (12)$$

$$\mathbf{F} = \frac{\partial f}{\partial x_v} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \quad (13)$$

$$\mathbf{G} = \frac{\partial f}{\partial u} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_m} \end{bmatrix} \quad (14)$$

$$\mathbf{H_v} = \frac{\partial h}{\partial x_v} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_m} \end{bmatrix} \quad (15)$$

$$\mathbf{H_f} = \frac{\partial h}{\partial x_f} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_m} \end{bmatrix} \quad (16)$$

2.1 Computation Complexity Analysis

It is well known that the computational requirements for EKF algorithm for SLAM is $\Theta(n^2)$, where n represents the number of features [1]. However to better understand how this computational complexity is distributed between the equations, we present in Table 2 an analysis of the number of floating-point operations for each EKF equation used in our proposed architecture. As can be noticed, the highest complexity is located in equation (6), 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 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 (8), $H^{(t)}$ is a sparse and 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 2.

3 Related Work

The EKF has been widely used in several areas, such as for image depth recovery [7], artificial neural network training [8] and for autonomous satellite navigation [9]. In probabilistic robotics it has also been intensively used to tackle the SLAM problem as it is considered one of the most efficient solutions that we have up-to-date [2].

In [10] one of the earliest implementations of the KF on an FPGA is presented (EKF and KF versions have the same core where the main differences are in the models since the EKF is just one extension to deal with nonlinear problems). It proposes a co-processor for target tracking in radar systems, which achieves performance of $1.8\mu s$ per filter iteration. The paper states a performance gain from two to three orders of magnitude over other implementations from that time, however it does not inform how many targets can be tracked at this speed. Since then several other implementations on FPGA have been proposed [11] [12] [13], despite that none of them is specific to the SLAM problem. Although the EKF core remains basically the same among the applications its computational complexity depends on how the internal state information is represented, updated and expanded. The main difference between our EKF hardware architecture and the existing EKF hardware proposals is that the architecture for SLAM needs to update in each filter iteration the whole covariance matrix for every element being estimated by the filter, as in the SLAM the elements/features and the robot position are estimated simultaneously. Consequently, any new information added to the filter needs to be propagated through all elements, independent of being or not at the moment detected by the sensor. Thus, the proposed architecture demonstrated in Section 4 is focus on, in addition to an efficient core implementation, reducing the external memory bottleneck by exploiting on-chip data reusability and data distribution between four external memories.

Over the past few years, there has been substantial interest in reducing the EKF computational cost for SLAM in software-based implementations by reducing the effort to update its covariance matrix. A common approach is to split the global map into a set of local maps and to update the covariance matrix locally [14] [15]. Another method adopted is to stop updating the map data for those features of low variance [16]. Some research has also concentrated on optimizing the EKF memory requirement [17].

Although significant advances have been made in software, very little attention appears to have been given in relation to the hardware architecture. However, it is possible to increase the system performance by creating hardware specialised for the EKF algorithm applied to SLAM. Furthermore, dedicated hardware tends to consume less power than general purpose solutions, which is very important for mobile robots powered by batteries.

Table 2 Number of floating-point addition (ADD), subtraction (SUB), multiplication (MUL) and division (DIV) operations for each EKF equation when $r = 3$ and $s = 2$, for n features.

Equation	FLOP			
	ADD&SUB	MUL	DIV	**Total per Equation**
(2)	5	4	–	9
(3)	47	58	–	105
(4)	$12n$	$18n$		$30n$
(10)	2	–	–	2
(11)	$16n + 44$	$20n + 50$	4	$36n + 98$
(12)	$16n + 35$	$20n + 52$	4	$36n + 91$
(5)	$4n + 6$	$4n + 6$	–	$8n + 12$
(6)	$8n^2 + 28n + 24$	$8n^2 + 32n + 30$	–	$16n^2 + 60n + 54$
Total per FLOP	$8n^2 + 76n + 163$	$8n^2 + 94n + 200$	8	$16n^2 + 170n + 371$

4 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. This architecture has been developed to be integrated with a light embedded processor to compute the prediction and measurement models from equations (2) and (10) along with their Jacobian matrices. This configuration allows easy update of models, since they can change according to the sensor and robot types. Thus, the proposed hardware can be re-used for any kind of sensor models since the robot and feature state size are kept in the same dimensions. For each EKF iteration the hardware receives $\mu_v^{(t)}$, $\nu^{(t)}$, $F^{(t)}$, $G^{(t)}$ and $H^{(t)}$ from the software and sends back $v^{(t-1)}$ and $\mu_{fi}^{(t-1)}$. As there is data dependency between software and hardware, the hardware starts its execution only after the software has finished its computation and the next EKF iteration starts only after both computations from hardware and software are concluded. In this implementation, the proposed hardware architecture has its own memory banks, and an additional memory bank is used for the software processor. Such a configuration provides an efficient hardware/software partition, as the computational complexity associated with the prediction and measurement models is generally small (less than 1k FLOPS), and is independent of the number of features. In addition, according to Table 1, the amount of data involved in the hardware/software communication is small and consequently does not compromise the overall system latency.

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 (6) 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).

Table 3 The PE operations, where A,B,C and R are block partitions of matrices.

Code	Operation
00	$R = AB$
01	undefined
10	$R = C + AB$
11	$R = C - AB$

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 [18]. In this SLAM system the depth information of the features are obtained by using the well-known triangulation technique [19]. 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 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. As can be seen in Fig. 1, the covariance matrix is distributed between the four memories, while

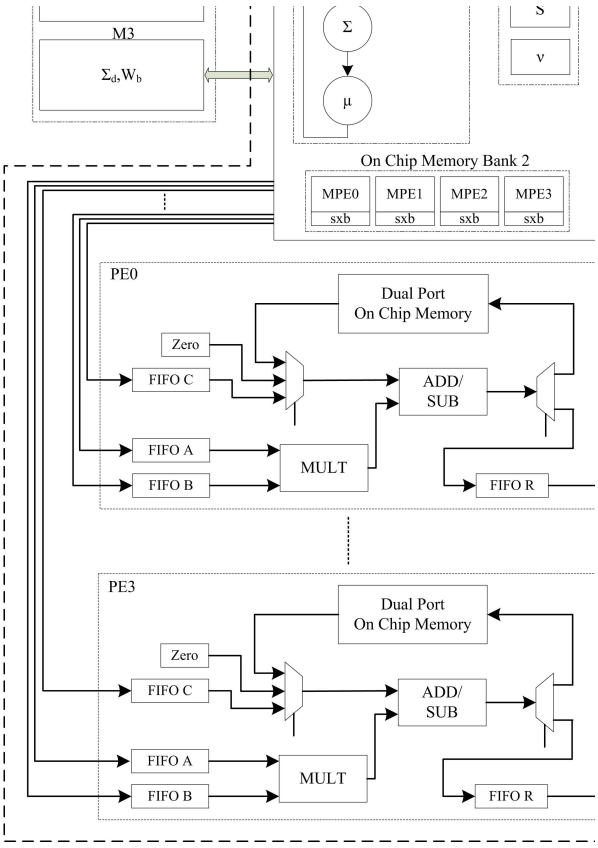

Fig. 1 EKF architecture; **MPEx** are on chip memories for PEs data reusability and **b** the PEs buffer size; other symbols are presented in Table 1.

the mean vector is in memory 0 and the filter gain matrix is distributed between memory 2 and 3. Such data organization allows an efficient bandwidth balance between these memory banks, since the covariance matrix is by far the most accessed data. 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.

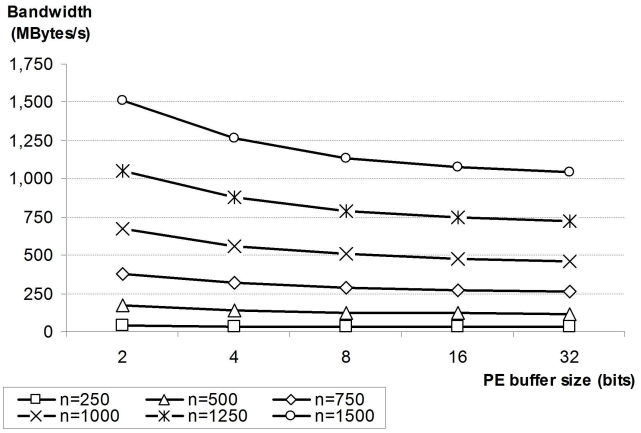

Fig. 2 External memory access bandwidth, where n represents the number of features.

4.1 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 presented in Table 3, in a such way that data reuse in the code can be exploited, mitigating the off chip memory access bottleneck, and considering that FPGA parallelism can be efficiently exploited.

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. Fig. 2 presents how the data reuse inside the PEs influences the external memory access bandwidth. This graph shows the bandwidth related to all elements of equation (6) for a set of different numbers of features computed according to equation (17), where ω is the memory bandwidth in MB/s, u the update frequency, m the matrix dimension, j the PE buffer size and s the feature state size. As can be noticed from this equation, the larger is j the more dominant is the $2m^2$ term, which corresponds to the reading and writing operations of the whole covariance matrix from and to the external memory. 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.

$$\omega = u(m^2 s/j + ms + 2m^2)4 \tag{17}$$

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 to take full advantage of pipeline depth. In the EKF algorithm,

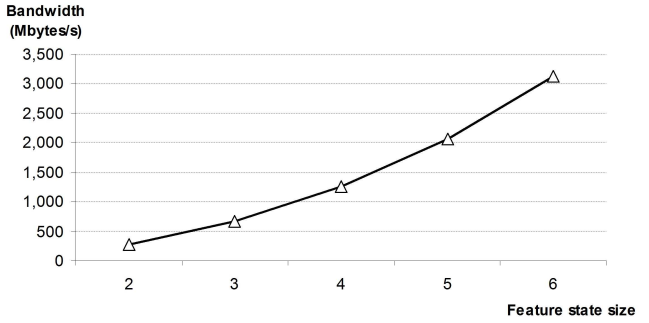

Fig. 3 External memory access bandwidth as a function of the feature state size using four PEs ($j = 8$), $1.5k$ features and a 14Hz update frequency.

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 (3).

4.2 Alternative System Configurations

Having presented the PE architecture and the methodology to implement the matrix operations, this subsection demonstrates the impact in the external memory bandwidth and on the PE operational frequency for alternative system configurations by changing the feature state size and the numbers of PEs. These projections are carried out using $1.5k$ features and a 14Hz update frequency. Fig. 3 demonstrates the impact on external memory bandwidth of changing the feature state size using four PEs ($j = 8$). Considering that each PE has one dedicated external memory and that each memory is able to transfer up to 400MB/s, the proposed system can support a feature dimension up to four without reducing the $1.5k$ features and the update frequency. However, to achieve such performance it is necessary to adjust the PE processing frequency according to the processing demand.

Fig. 4 shows the PE pipeline frequency required for the EKF in relation to the number of PEs, which has been computed based on the EKF computational complexity (FLOP), considering the feature state size equal to two and that each PE computes two floating-point operations per clock cycle. According to this graph, with four PEs the required frequency is around 50MHz demonstrating that it is possible to improve the system performance for higher feature state dimensions as in FPGAs it is feasible to have PEs operating over 100MHz.

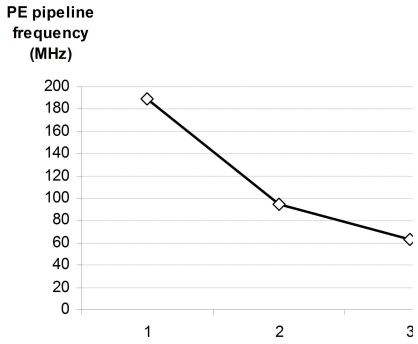

Fig. 4 PE pipeline frequency required for the EKF in relation to the number of PEs, considering feature state size equal to two and that each PE computes 2FLOP per clock cycle.

Table 4 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

5 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 [20] and it has been validated in the Celoxica RC250 development kit [21], featuring 4 external SRAM memory banks and an EP2S90F1020C4 FPGA.

The resource utilization and the maximal clock frequency of a single PE is shown in Table 4. 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 5 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 large as it has to repeatedly partition each matrix between the PEs and combine the results again. Moreover, it contains a floating-point di-

Table 5 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%)

Table 6 The performance and power comparison between the FPGA proposed architecture and the Pentium M 1.6GHz and ARM920T 200MHz processors.

	FPGA	Pentium M	ARM9
Device power (W)	1.3	31.1	0.75
Max. features in real time (k)	1.8	0.57	0.13
Power per feature (mW)	0.7	54	5.7

vision 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.

5.1 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 2, 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 the Quartus II tool. The estimated power using signal activities generated by probabilistic methods is 1.3W.

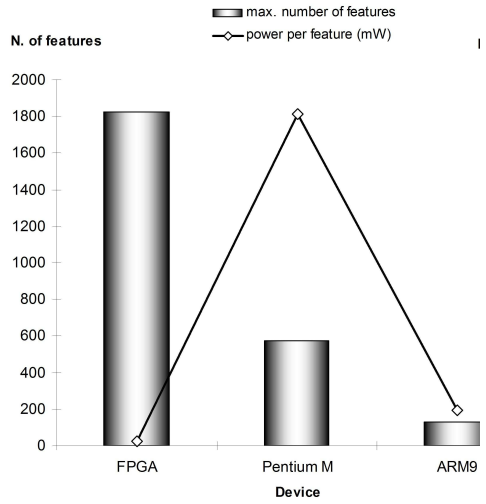

Fig. 5 Maximum number of features and the power consumption for three processing devices.

To compare these results with other technologies we consider the Pentium M 1.6GHz and the ARM920T 200-MHz (Cirrus Logic - EP9315) processors, as the first one is commonly found in on-board mobile robot cards, such as in the Pioneer 3DX platform [22], and the second one is widely used in low power embedded applications. The performance achieved in real time (14Hz) on these devices, running the EKF implemented in C, are 572 features for the Pentium and 131 for the ARM processors. The codes for both ICs were compiled using GCC 3.4.6 with the switches configured as standard and no special features used. The software is mainly limited by the external memory bandwidth and floating-point unit performance, which could be significantly improved by using multi-core processor technology, dedicated floating-point units, provided by high speed external memory banks.

According to the processor datasheets, Pentium M consumes 31.1W and ARM 750mW when running at full speed. Table 6 summarizes the comparison among these three approaches, which is also illustrated in Fig. 5. As can be seen, despite the ARM processor having the lowest power consumption, the FPGA processes more features in real time and hence also consumes less power per feature than both processors. It is also important to consider that the EKF computational complexity is $\Theta(n^2)$, where n represents the number of features. Thus, to achieve the same FPGA performance the Pentium M must increase its processing power by 9.9× and the ARM920T by 189×.

6 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 considerably higher performance and significantly less energy consumption than the Pentium M 1.6GHz and ARM920T 200MHz processors.

Although the adopted FPGA would support more than four PEs, the current system performance of 1.8k features at 14Hz, which is suitable for most applications in indoor environments, is limited by the external memory bandwidth. The recent approach of splitting the global map into a set of local maps so as to update the covariance matrix locally would be a solution to overcome this limitation when higher performance is need. Another possibility would be to reduce the floating-point precision or even use a fixed-point format. However, these approaches require special attention as the result precision can compromise the filter convergence.

Finally, as 75% of the FPGA resources are still free a softcore processor with floating-point unit, such as NIOS II, can be added in this space in order to compute the EKF prediction and measurement models discussed in section 4. Furthermore, it is possible to implement many other robot functionalities in the same chip avoiding the necessity of using personal computers commonly found in mobile robots.

Acknowledgments

The authors would like to thank CAPES (Ref. BEX2683/06-7) and EPSRC (Grant EP/C549481/1 and EP/C512596/1) for the financial support given to develop this research project. We also thank Carlos R.P. Almeida Jr. for his important help to generate the EKF software benchmarks.

References

1. S. Thrun, W. Burgard and D. Fox, *Probabilistic Robotics*, Cambridge, MA, USA: MIT Press, 2005.
2. S. Thrun, "Robotic Mapping: A Survey", Carnegie Mellon University (Technical Report - CMU-CS-02-111), 2002, p. 31.
3. D. Fox, J. Hightower, L. Liao, D. Schultz and G. Borriello, "Bayesian Filters for Location Estimation", *IEEE Pervasive Computing*, vol. 2, no. 3, 2003, pp. 24–33.
4. 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, 2003, pp 139–144.
5. T. Bräunl, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*, Berlin and Heidelberg: Springer-Verlag, 2003, pp. 111–129.
6. R. Smith, M. Self and P. Cheeseman, "Estimating uncertain spatial relationships in robotics", *Autonomous robot vehicles*, 1990, pp. 167–193.
7. P. Chalimbaud, F. Marmoiton and F. Berry, "Towards an Embedded Visuo-Inertial Smart Sensor", *The International Journal of Robotics Research*, vol. 36, no. 6, 2007, pp. 537-546.
8. K.R. Anne, K. Kyamakya, F. Erbas, C. Takenga and J.C. Chedjou, "GSM RSSI-based positioning using extended Kalman filter for training artificial neural networks", *IEEE Vehicular Technology Conference*, vol. 6, 2004, pp. 4141- 4145.
9. J. Albukerque, J.L. Lair, B. Govin, G. Muller and P. Riant, "Autonomous satellite navigation using optico-inertial instruments", *Automatic control in space*, 1985, pp. 183-188.
10. C.R. Lee and Z. Salcic, "A fully-hardware-type maximum-parallel architecture for Kalman tracking filter in FPGAs", *IEEE International Conference on Information, Communications and Signal Processing*, 1997, pp. 1243-1247.
11. Y. Liu, C. Bouganis and P.Y.K. Cheung, "Efficient Mapping of a Kalman Filter into an FPGA using Taylor Expansion", *IEEE International Conference on Field Programmable Logic and Applications*, 2007, pp. 345–350.
12. Z. Zhangxing and Z. Xinhua, "Design of fpga-based kalman filter with CORDIC algorithm", *Recent trends in multimedia information processing*, 2002, pp. 191–199.
13. Z. Salcic and C.R. Lee, "FPGA-based adaptive tracking estimation computer", *IEEE Transactions on Aerospace and Electronic systems*, vol. 37, no. 2, 2001, pp. 699–706.
14. L.M. Paz and J. Neira, "Optimal Local Map Size for EKF-based SLAM", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5019–5025.
15. C. Estrada, J. Neira and J.D. Tardós, "Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments", *IEEE Transactions on Robotics*, vol. 21, no. 4, 2005, pp. 588–596.
16. G.H. Kim, J.S. Kim and K.S. Hong, "Vision-based Simultaneous Localization and Mapping with Two Cameras", *IEEE International Conference on Intelligent Robots and Systems*, 2005, pp. 3401–3406.
17. J.E. Guivant and E.M. Nebot, "Solving Computational and Memory Requirements of Feature-Based Simultaneous Localization and Mapping Algorithms", *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, 2003, pp. 749–755.
18. V. Bonato, J.A Holanda and E. Marques", "An Embedded Multi-Camera System for Simultaneous Localization and Mapping", *Applied Reconfigurable Computing*, Lecture Notes on Computer Science - LNCS 3985, 2006, pp. 109-114.
19. R.I. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision", Cambridge University Press, Ed. 2, 2004.
20. Celoxica, "Handel-C Language Reference Manual (User Guide)", 2005, http://www.celoxica.com.

21. Celoxica, “RC250 - High Performance System Development & Evaluation Board”, 2005, http://www.celoxica.com/products/.
22. ActivMedia Robotics, “Technical Specifications”, 2006, http://www.activrobots.com/ROBOTS/p2dx.html.

Vanderlei Bonato has MSc(2004) and BSc(2002) degrees in Computer Science. He obtained the PhD in 2008 by the Institute of Mathematical and Computing Sciences at University of Sao Paulo, Brazil, which was partially developed at the Department of Electrical and Electronic Engineering of Imperial College London. His interest areas are image processing and probabilistic algorithms applied to mobile robotics. Recently, he has concentrated his work on developing customized architectures based on reconfigurable computing for navigation and mapping problems. Finally, he has also seven years of industrial experience in automation systems and four years in visual electronic communication projects.

Eduardo Marques has a MSc degree in Computer Science, and a PhD degree in Digital System Engineering, both from the University of São Paulo, Brazil. He is a faculty member of the ICMC-USP since 1986.

Dr. Marques is Associate Editor of the International Journal of Reconfigurable Computing. Recently, he has been involved in the organization of some international events in the area of reconfigurable computing, such as ARC´2007 and the 23rd Annual ACM Symposium on Applied Computing (track Robotics: Hardware, Software, and Embedded Systems). He also serves on the technical program committees of FPL, SPL and JCIS.

During the first years of his academic career his main research interest was on Parallel Machines and Dataflow Computing. Afterwards, he has focused on Reconfigurable Computing, currently working on embedded systems and framework development applied to mobile robotics and education.

George A. Constantinides graduated in 1998 from Imperial College (U.K.) with an M.Eng.(Hons.) in Information Systems Engineering. He then joined the Circuits and Systems group in the Electrical and Electronic Engineering department at Imperial College, where he obtained his Ph.D. in 2001 and joined the faculty in 2002.

Dr. Constantinides is Associate Editor of the IEEE Transactions on Computers and the Springer Journal of VLSI Signal Processing. He was programme co-chair of FPT in 2006 and FPL in 2003, and serves on the technical program committees of FPL, FPT, ISCAS, CODES+ISSS and ARC, for which he is also a member of the steering committee.

His research interests include the theory and practice of reconfigurable computing, electronic design automation, and customised computer architectures. He is particularly focused on numerical algorithms from digital signal processing and optimization. He is the author of over 70 peer-reviewed publications in this area.