FPGA design of EKF block accelerator for 3D visual SLAM

Daniel Tortei, Jonathan Piat, Michel Devy

FPGA design of EKF block accelerator for 3D visual SLAM

Daniel Törtei Tertei[a,b,c], Jonathan Piat[a,b,*], Michel Devy[a,*]

[a] *CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*
[b] *Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France*
[c] *Faculty of Technical Sciences, Department of Computing and Automation, 21000 Novi Sad, Serbia*

Abstract

This paper deals with the evaluation of a dedicated architecture to be integrated in an embedded system mounted typically on a micro-aerial vehicle or on smart devices moved by an operator. This system performs an Extended Kalman Filter (EKF) based visual odometry algorithm: here we present an efficient hardware architecture conceived as a systolic array co-processor for EKF loop acceleration. Due to severe constraints on the power consumption, the real-time performance and the physical characteristics of the system (compacity, weight...), this algorithm is implemented entirely as a System On a programmable Chip (SoC) on the Zynq-7020 device. This heterogeneous platform[1] consumes less power than a standard microprocessor and provides powerful parallel data processing capabilities: applying hardware/software (hw/sw) co-design allows to guarantee real-time throughput with a very low power-per-feature rate.

Keywords: visual EKF-SLAM, SoC, FPGA, co-processor, AHP

1. Introduction

More and more applications require the mobility of a system (a robot, a vehicle, a drone, an operator with a tablet or a smartphone,...) equipped with several sensors. As soon as such a system moves in an environment, it must be endowed with several functions able to provide its position with respect to a known reference frame, to detect obstacles in order to avoid collisions (for robot navigation) or to recognize some known objects (for augmented reality). Our contribution concerns only the localization functionality.

Many solutions exists in the litterature, depending on the available sensory modalities, on the locomotion system, on the a priori knowledge learnt on the en-

*CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Email addresses: dtertei@laas.fr (Daniel Törtei Tertei), jpiat@laas.fr (Jonathan Piat), mdevy@laas.fr (Michel Devy)

[1] processor + reconfigurable hardware

vironment, and overall, on the available resources (energy, memory, computing power ...). Many robots (e.g. AGV for industrial environments) and vehicles (e.g. the Google car) are equipped with laser range finders in order to perceive the environment; here it is proposed to use only vision, i.e. a low-cost sensor that could be embedded both on autonomous robots and on smart devices moved by humans.

A standard Visual Odometry (VO) algorithm is exploited for the localization of our system moving in an initially unknown environment. Such a function builds a local map of its environment in the form of positions of *landmarks* observed in images, estimates the local motion made by the system, and then, cumulating these motions, estimates the system position with respect to a specific reference frame, generally the initial position when the VO algorithm has been started. Landmarks are presumed to be static. Our mobile system must be equipped with a camera and besides it may also have proprioceptive sensors in order to give other measurements on the motions: wheel odometry for a robot, Inertia Measurement Unit (IMU) for a device worn by an operator or for a drone...

A VO algorithm is very similar to a Visual Simultaneous Localization and Mapping approach (VSLAM), but the localization in VO is only estimated from a local map, i.e. a map built on a given horizon time, defined for example by the M last acquired images or by the N last detected landmarks; as the localization is computed from local information, it will be impacted by a drift growing with the time. With a Visual SLAM algorithm, it is possible to build a global map, even on large environments: if the mobile system returns in a known area and succeeds in matching observed features with mapped landmarks, cumulative errors may be corrected using a loop closure function. VO and VSLAM could be based either on Filtering - EKF, Rao-Blackwellized Particle Filter...- or on Optimization (global or incremental, iterative or direct...). It is known [1] that the localization given by Optimization methods is more accurate than for Filtering ones, but it requires more computing power, and moreover, for large environments, the map size could be huge, so that some map management strategy has to be applied.

It is the reason why, for systems of limited size, an EKF-based VO algorithm is most often applied for position estimation; moreover it is easier to take into account measurements acquired from asynchronous multiple modalities (several cameras, gyros, accelerometers...). Drifts could be corrected from time to time either fusing with a priori knowledge given to the system (e.g. Open Street Map in urban scenes, or a building map for indoor scenes) or with a global position provided by a GPS receiver in outdoor environments.

In spite of the increasing computing power on compact embedded system, EKF-based VO's computational complexity still may be too important with only a software implementation, mainly due to image processing in order to extract features from images and due to large scale matrix-matrix multiplications when applying the filter on large state vectors. The real-time constraint for typical applications of drones, robots or augmented reality on smart devices, involves to update the system position at least at $30Hz(33.33ms)$. Moreover improving

the frequency allows to enter a virtuous cycle: faster is the VO process, easier and more robust is the feature-landmark matching. Another issue is autonomy of the mobile system: standard processors, such as Intel's quad core i7, consume much power, which limits the level of autonomy.

As a solution to these issues, an embedded system has to be designed which would ensure high computational efficiency at low power consumption rates. Modern reconfigurable devices such as Field-Programmable Gate Arrays (FPGAs) may answer to that question. Their reconfigurable fabric consists of a large number of configurable slices, and besides they come with embedded Digital Signal Processing (DSP) blocks that can be used for floating point applications. Compared to a standard processor, computationally extensive algorithms may be parallelized and thus executed several times faster on FPGAs in floating point precision [2]. While Graphical Processing Units (GPUs) also make use of the parallelization techniques, the main advantage of FPGAs over both GPUs and general purpose central processing units (GPCPUs) remains in significantly lower power consumption.

Embedded GPUs have as well emerged on the market today with similar power consumption to an FPGA [3]. However, they are designed as to improve computational performances on highy demanding image processing applications. Thus FPGAs with their algorithm-specific memory management capabilities owing to reconfigurability still remain a better choice over embedded GPUs in terms of computational efficiency of a custom application.

So this paper presents mainly an efficient SoC architecture for EKF-based Visual Odometry applications. Using hw/sw co-design we intend to fully port VO functionality of our visual SLAM application into embedded domain. As this localization algorithm relies on heavy matrix multiplication computations, it is focused on a matrix multiplication accelerator conceived as a systolic array co-processor for a hard core ARM CortexA9 processor integrated in a Zynq-7020 FPGA.

1.1. State of the art and motivation

In literature, methods that use Visual SLAM for localization may be classified according to several criteria. Here, only Filtering-based (and specifically EKF-based) methods are considered, like in [4, 5], because they are more adapted for embedded systems with limited resources. Several real-time optimization-based methods have been proposed, but either they are not robust [6], or they require a huge computing power [7] or they are specific for drone applications [8].

Then according to features extracted from images, most methods exploit only interest points (Harris, SIFT, SURF, BRIEF, ORB...); [9] compares several methods proposed for point detection and characterization. Some authors [10] exploit heterogeneous landmarks; here only Harris points are used.

Finally according to the sensor configuration, the SLAM function could acquire images from monocular, bi-cam or stereovision sensors. These last one allows to acquire depth images, but only on the sensor vicinity e.g. about 8m.

For a 40cm stereo baseline; 3D landmarks are directly created as Euclidean points; points far from the sensor are never considered. Monocular or bearing-only SLAM uses a single camera, so that only 2D features can be observed; from such a feature it is not possible to directly add a 3D landmark in the map; either the initialization of this landmark is delayed [11], waiting for other observations from different view points so that a 3D point could be triangulated, or this initialization could be undelayed using a specific parametrization [12, 13], allowing to exploit landmarks far from the camera, e.g. a peak on the horizon line. Bicam-SLAM combines the two approaches, allowing to create landmarks both from features matched between the two images, and from features that are only perceived by one camera. Here only monocular SLAM with undelayed landmark initialization is used, so that our method could be executed on an embedded system limited in size and resources - such as a smartphone.

RT-SLAM [14] is a generic Open Source software implementation of our state of the art visual bearing-only SLAM algorithm, based only on one camera to observe the environment, and on an IMU to estimate motions. The map contains Anchored Homogeneous Point (AHP) landmarks [12] which is a seven-dimensional landmark parametrization. AHP landmark parameterization is shown to increase filter's consistency, achieving more robust and accurate localization and mapping. A predefined number of landmarks from the map is observed and positions are corrected after each frame acquisition. The choice is made by applying the one-point Random Sample Consensus (RANSAC) [15]. Active search strategy is used for initialization of new ones. Authors [14] report real-time performance at $60Hz$ when having 20 consecutive landmark corrections and 5 initializations per EKF loop on an Intel i7 processor (only one core is used).

C-SLAM [16] presents a C programming language version of RT-SLAM that implements a simplified SLAM application respecting the DO-178 norm required on airborn system. It uses a constant speed robot motion function (no inertia data) and Inverse-Depth Point landmark parameterization [13]. C-SLAM was implemented in [17], on an embedded architecture based on a Virtex5-Virtex6 FPGA couple [18]; it runs at 24Hz when having 20 IDP landmarks in the map and correcting 10 of them in each frame. The advantage gained by this architecture lies on the co-design possibility, with a hardware-level implementation of image processing functions at pixel frequency: Harris point extraction, active search strategy ...

Our SLAM is an upgraded version of C-SLAM. It is a monocular SLAM fusing IMU data for motion prediction, correcting 20 AHP interest points. Active search strategy is used as well to match the observed points and initialize new ones (5 per frame). This number is obtained by experimentation for one application on a robotic platform: C-SLAM behaviour was satisfactory when having set parameters in this manner. IMU data are used in order to reduce uncertainty on landmarks tracking during interest points matching. With this (algorithmic) optimization and for speed issues, one-point RANSAC is replaced by random choice when initializing new points in the map.

Our approach is as follows: a SLAM algorithm may be divided into two

parallel processes [19], namely vision (*front-end*) and filtering (*back-end*). These two processes are executed in pipeline, so that we have a latency of one image period, i.e. the system position is updated at $T+2\Delta T$, with landmarks observed in the image acquired at time T. In [20] it is shown that pose estimation can make up to 84% of execution time of an EKF-based SLAM algorithm: so, we concentrate on acceleration of the filtering part of the SLAM. We propose an EKF matrix multiplication SoC that is full-compliant to floating point IEEE 754 single precision standard [21] and which permits an EKF loop throughput of 30 Hz with a maximum of 52 AHP points in the map (while observing and correcting 20).

A power-efficient FPGA-based embedded EKF block accelerator is also proposed in [22]. Their SLAM is a 2D EKF-SLAM application that satisfies real-time constraints ($14Hz$ of execution time in their case) with approximately 1.8k landmarks contained in their map which are represented in Euclidean parametrization (two-dimensional state size). All of them are observed and corrected. They do not report on initialization and matching techniques and neither mention whether they use sensors besides a camera. It is the only FPGA-based solution (besides ours) in literature that proposes a visual EKF-SLAM block embedded accelerator to our knowledge. However, given the dimensionality of our problem (AHP parametrization) and system complexity (3D mapping, sensors) it is hardly comparable.

Finally, some authors proposed FPGA implementations of matrix multiplication or inversion, using different strategies [23, 24, 25]; the Parallel input/Multiple Output strategy proposed in this last paper, is very similar to our systolic array architecture.

1.2. Previous work and contributions

Our previous work on the subject is published in [26] - an EKF accelerator which is a PLB peripheral to PPC440 hard core embedded processor on a Virtex5 FPGA. It does not take into account the symmetry in cross-covariance matrix - related computations (authors in [22] do not mention that fact either). That impovement leads to significant improvements in design to both latency and required on-chip memory usage, which will be discussed here.

Contributions of this paper are:

- Genericity of the accelerator regarding system state size (other sensors may be added/removed), feature state size (up to seven-dimensional) and number of landmarks (up to 52 AHP points)
- Implementing the new design as an AXI4 bus peripheral on power-efficient Zynq-7020 FPGA
- Taking into account the cross-covariance matrix symmetry to reduce computational time (in both software and hardware) and on-chip memory storage

The paper is organized as follows: in Section 2 an analysis of visual EKF-SLAM complexity is given in order to deduce it's computational bottleneck. In Section 3 we address relevant designing issues as to theoretical lower bounds on the latency of any multiplication algorithm and the computational model mapping based on systolic array technique (with regard to their feasibility on an FPGA). Our SoC solution to EKF block acceleration is presented in Section 4. In the following section - 5 - it is put under evaluation in terms of latency, resource usage and power consumption. Experimental results (measured latencies and maximum supported number of AHP landmarks in the map) with future work are given in Section 6. Finally, Section 7 concludes the paper.

2. Visual EKF-SLAM analysis

Goal of this section is to establish a theoretical background in order to identify the computational bottleneck of an EKF function used for visual SLAM algorithm in our case. General conclusions may be induced by varying parameters in Table 1.

A state-space based predictive algorithm based on EKF [12] runs in a three-steps loop: i) prediction of robot's position from the motion estimates, ii) correction of the robot's and landmarks' positions with respect to the world reference frame (by tracking chosen landmarks), and iii) initialization of landmarks [27].

Let us recall that here that our SLAM is implemented without loop closure, so the system position is updated from a local map, with a limited size. Landmarks are removed and replaced by new ones in several situations. (1) They are never matched, due to environmental changes (illumination, occlusions ...). (2) They have been selected on the surface of a dynamic object. (3) Due to the camera motion, they are overpassed, so they could not be observed during the next motions. Possible losses of landmarks may severely influence the quality of motion estimation. Thus, it is preferable to have many landmarks memorized in the map after loop iii) and to choose a subset of these upon which position corrections are going to be made in loop ii). Also, parametrization (dimensioning) of landmarks plays a significant role in their quality [12], yielding robuster behaviour with higher landmark state size.

Introducing larger number of higher-dimensional landmarks increases visual EKF-SLAM's computational complexity, having as a consequence in loop ii) larger scale matrix-matrix multiplications.

It is shown that computational requirements for an EKF algorithm depend on the number of features N retained in the map: $L_{EKF} = O(N^2)$ [28]. Equations of the EKF are given below[2].

Prediction loop:

$$\hat{x}^{(t)} = f(\hat{x}^{(t-1)}, \varpi^{(t)}) \tag{1}$$

[2]In equations 2 and 7 we perform matrix calculations only on P_1 and P matrices' upper triangles respectively, as the cross-covariance matrix P is symmetrical

$$P_1^{(t)} = F_x^{(t)} P_1^{(t-1)} F_x^{(t)T} + F_\omega^{(t)} Q F_\omega^{(t)T} \tag{2}$$

$$P_2^{(t)} = F_x^{(t)} P_2^{(t-1)} \tag{3}$$

Correction loop:

$$Z^{(t)} = H^{(t)} P_3^{(t)} H^{(t)T} + R \tag{4}$$

$$K^{(t)} = P_4^{(t)} H^{(t)T} (Z^{(t)})^{-1} \tag{5}$$

$$\hat{x}^{(t)} = \widehat{x}^{(t)} + K^{(t)} (y^{(t)} - h(\hat{x}^{(t-1)})) \tag{6}$$

$$P^{(t+1)} = P^{(t)} - K^{(t)} Z^{(t)} K^{(t)T} \tag{7}$$

After each frame acquisition, the front-end process furnishes interest points found by a feature detector. Active search algorithm is used to find correspondences with observed landmarks in the past frame. In our case, we observe and search for correspondencies between 20 interest points. Matched points are passed through in the camera reference frame. They are afterwards reparameterized to richer AHP representation, which is the data type upon which our Kalman filter works. This information is stored in robot state vector $\hat{x}$ and is updated in the process within Jacobian (H) and cross-covariance (P) transformations - Table 1.

Given this data, an EKF loop is executed as follows: first, a prediction step is performed in order to update the corresponding elements in P : 1 - 3. Matrices P_1, P_2, P_3 and P_4 are sub-matrices of the cross covariance matrix P. Secondly, multiple correction loops are run, each with respect to a single observed landmark: equations 4 - 7 (in our case 20 times). In correction loop the filter's gain is calculated, upon which landmarks' and robot's positions are being updated (along with the the entire cross-covariance matrix). It is done in equation 6 where IMU odometry and GPS data are integrated in measurement process y^t (localization). Upon observed, matched and corrected landmarks all landmarks in the map are reparameterized from AHP to three-dimensional Euclidean representation (3D mapping). After correction loops, we are able to reduce the search space for landmarks which we are observing in the following frame.

In Table 1, N stands for the total number of landmarks retained in the local map, d is the dimension of AHP feature state size (which is seven), six is the number for GPS and IMU related variables and r is the state-space size of our robot (which is nineteen). All twenty landmarks in the map are observed. From that table we deduce Table 2 which shows the total amount of floating-point operations that are executed in an EKF block in function of the number of retained landmarks N in the visual map.

Highest share of the execution time belongs to equation 7. Here, the upper triangle of the square P matrix is updated during correction. This fact

Table 1: Description and matrices dimensions according to our implementation of the EKF algorithm.

Symbol	Dimension	Description
$\hat{x}$	$(dN + r) \times 1$	Robot and feature positions
f	-	Prediction function
P	$(dN + r) \times (dN + r)$	Cross covariance matrix
P_1	$r \times r$	Cross covariance matrix with respect to robot position
P_2	$r \times dN$	Cross covariance matrix with respect to all landmarks
P_3	$2d \times 2d$	Cross feature-robot covariance matrix
P_4	$(dN + r) \times 2d$	Cross feature-feature and robot-robot covariance matrix
F_x	$r \times r$	Jacobian to system state
F_ω	$r \times 6$	Jacobian to system state perturbation
Q	6×6	Permanent perturbation
Z	2×2	Covariance innovation
H	$2 \times 2d$	Jacobian
R	2×2	Measurement noise
K	$(dN + r) \times 2$	Filter gain
y	2×1	Measured output
h	-	Observation function

Table 2: Number of floating-point operations in an EKF loop taking into consideration symmetry of the cross-covariance matrix. Equations 4 - 7 are executed c = 20 times, for each observation

Equation no.	FLOP	Execution time [%]
1	361	0.02
2	17000	1.21
3	4921N	6.99
4	c × 868	1.23
5	c × (420N + 1140)	13.54
6	c × (49N + 135)	1.58
7	c × ($102N^2$ + 576N + 811)	75.43
Total	$2040N^2$ + 25821N + 76441	100

is confirmed after having run a code profiling tool [29] over the implemented EKF-SLAM code (which will be more detailed in section 6). Thus, a significant speed-up can be made by leveraging the $P - KZK^T$ operation in hardware. As the P matrix of a Filtering-based visual SLAM application is frequently evoked for pose estimation and state update, special care needs to be made on data communication with the rest of the system. It is why we propose to store it on high-speed on-chip memory. In that way we could keep the FLOPs number in equation 7 close to a constant value and thus rendering it independent of N^2 in terms of processor execution time.

3. Design considerations

This section will start with floating-point matrix multiplication considerations on an FPGA device (latency, I/O bandwidth and storage). Afterwards,

after having introduced a Systolic Array (SA) - based processing system on an FPGA, a processing element (PE) - based computational flow for EKF's equation 7 will be given.

3.1. Matrix multiplication tradeoffs on FPGAs

On a reconfigurable computing system the main tradeoff is between optimal speed and resource utilization. Higher design speeds may be achieved at the cost of a higher resource usage. On the other hand, the limited number of configurable slices should be used for parallel logic implementation. Considering applications of matrix-matrix multiplication algorithms on programmable hardware of limited size, such as FPGAs, we have to take into account specific constraints on latency L, total storage size in words M and memory bandwidth requirement B denoted by number of I/O operations performed in each cycle. Based on a multiplication of two n-squared matrices, authors in [30] explain these tradeoffs in reference to lower bound on achievable latency:

$$L \geq max(L_1, L_2) \geq max\left(\frac{n^3}{p}, \frac{n^3}{\sqrt{M}B}\right), (M \leq n^2) \tag{8}$$

where p is the number of theoretical Processing Elements (PE) that may be executed in parallel, L_1 is the latency according to computation time and L_2 is the latency dependent on I/O bandwidth. Furthermore, they conclude that an optimal latency is in this case of order $O(n^2)$ when having $p = O(n)$ PEs and $M = O(n^2)$ available storage size in words. However, for large scale matrices the latency is of order $O(\frac{n^3}{p})$. This is due to the fact that having FPGAs with limited resources it is hardly possible to instantiate that many PEs in parallel. A recent work [31] describes an architecture of linear array PEs, similar to those in [30], but achieving an optimal latency of order $O(n^2)$ by exploiting full-duplex communication with the host processor (no on-chip memory storage) and at the cost of having it involved during addition of intermediary values. Our design will focus on minimal latency of matrix multiplications in EKF equation 7 and on-chip memory storage.

3.2. Notion of systolic arrays on an FPGA

As previously explained, and especially due to available resources on the Zynq-7020 device, we chose to optimize the tri-matrix multiplications. Pipelined designs present a suitable solution when it comes to allocating resources given the specific problem size of our computational model. By identifying computational kernels (i.e. floating point multiplication, addition and subtraction) a problem-specific computational unit called PE may be conceived. Systolic array (SA) is a form of a pipelined design consisting of a multi-dimensional grid of interconnected PEs. Main sequential operation is being optimized (e.g. equation 7 in our case) by restructuring its executional flow so that it may be executed on parallel PEs. Besides, a central logic, called *Sequencer*, controls the data flow in the grid. Advantage of a SA based computation over sequential or multi-threaded is that the its grid is able to generate new results after each clock

cycle. In consequence, by exploiting structural properties of an SA we are able to lower the dimension of the computational model (which is three-dimensional for n-squared matrices multiplication) and obtain a better throughput at the cost of more complex control logic of data flows.

Authors in [32] present useful mapping techniques inherent to an FPGA-based design. In order to avoid using large number of buffers due to routing issues in a standard matrix-matrix multiplication algorithm, they propose a modified locally recursive version upon which they map control logic onto their two-dimensional SA. However, because of speed issues, PEs are often organized in a linear list structure - a special case of a one-dimension systolic array in which they interconnect by short connection wires. In general, broadcasting data on a larger grid in an FPGA using a higher dimensional SA structure is not recommended because of speed degradation due to the size and routing complexity of FPGA-based floating point units. In order to achieve efficient low latency design we focus our efforts on implementing a 1-D SA.

3.3. Computational model mapping

The computation model of a one-dimensional SA with four PEs adapted to equation 7 of the EKF block may be described with the following pseudo-code:

Phase I: $K \times Z$

1: **for** $i = 0; i < 7N + 19; i+ = 4$ **do**
2: (load K matrix, in parallel for x = 0 : 3)
3: $PE_{in}(x) = \{K_{i+x,0}, K_{i+x,1}\}$
4: (double-buffer Z matrix)
5: $PE_{in}(0) = \{Z_{0,0}, Z_{1,0}\}$
6: $PE_{in}(1) = \{Z_{0,1}, Z_{1,1}\}$
7: $PE_{in}(2) = \{Z_{0,0}, Z_{1,0}\}$
8: $PE_{in}(3) = \{Z_{0,1}, Z_{1,1}\}$
9: (output PE function)
10: $PE_{out}(0) = K_{i+x,0} \times Z_{0,0} + K_{i+x,1} \times Z_{1,0}$
11: $PE_{out}(1) = K_{i+x,0} \times Z_{0,1} + K_{i+x,1} \times Z_{1,1}$
12: $PE_{out}(2) = K_{i+x,0} \times Z_{0,0} + K_{i+x,1} \times Z_{1,0}$
13: $PE_{out}(3) = K_{i+x,0} \times Z_{0,1} + K_{i+x,1} \times Z_{1,1}$
14: **end for**

Phase II: $P - KZ \times K^t$

1: **for** $i = 0, y = 0; i < 7N + 19; i+ = 4, y+ = 4$ **do**
2: (load KZ matrix, in parallel for x = 0 : 3)
3: $PE_{in}(x) = \{KZ_{i+x,0}, KZ_{i+x,1}\}$
4: **for** $j = y; j < 7N + 19; j+ = 4$ **do**
5: (load K^t matrix, in parallel for x = 0 : 3)
6: $PE_{in}(x) = \{{K^t}_{0,j+x}, {K^t}_{1,j+x}, P_{i+x,j+x}\}$
7: (output PE function, in parallel for x = 0 : 3)
8: $PE_{out}(x) = P_{i+x,j+x} - (KZ_{i+x,0} \times {K^t}_{0,j+x} + KZ_{i+x,1} \times {K^t}_{1,j+x})$
9: **end for**
10: **end for**

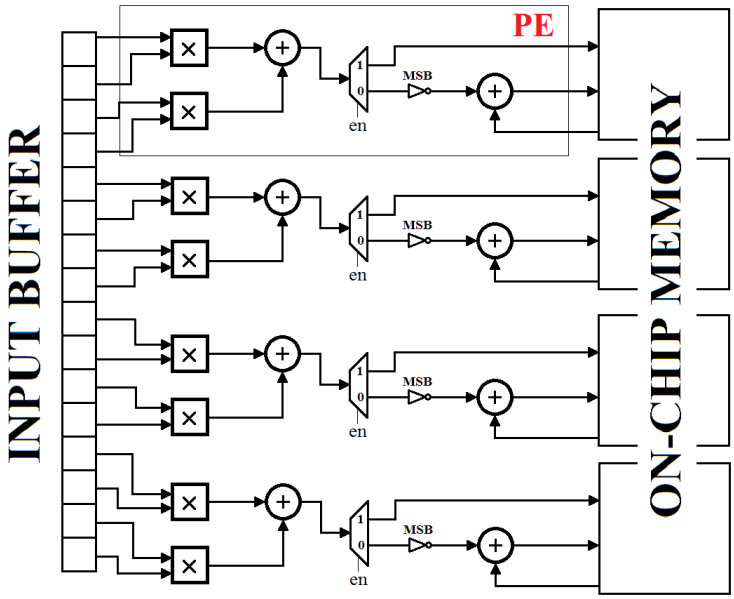

Figure 1: Our tiling of the 1-D systolic array for visual EKF-SLAM matrix multiplications

Each PE consists of two floating-point multipliers and two floating-point adders - Fig. 1. During KZ multiplication, the second adder is disabled ($en = 1$). For the Phase II, floating-point subtraction is performed by negating the most significant bit (MSB) of the second operand (in IEEE floating-point format this negates the given number value) in addition operation. In this way, the whole matrix subtraction process takes a latency of a single floating-point adder. In phase II in the inner loop the P matrix symmetry is taken into account ($j = y$). This structure yields four floating-point matrix multiplication results in parallel every clock cycle. As the innovation matrix Z is always square-two dimensional for a given visual EKF-SLAM application, we argue that a PEs conceived in this manner are optimal for those kinds of applications: there is no need for storage of intermediary values during matrix-matrix multiplications (in both Phase I and II).

In the next section we explain in detail all the structural aspects of our EKF block accelerator with its functional description.

4. Visual EKF-SLAM block accelerator

Accelerator's hardware architecture is given on Fig. 2.

The host processor (dual core ARM CortexA9) performs equations 1 - 6. Accelerator's main logical component is the *Main sequencer*, a state machine generic to parameters N and r. The tri-matrix multiplication in equation 7 is

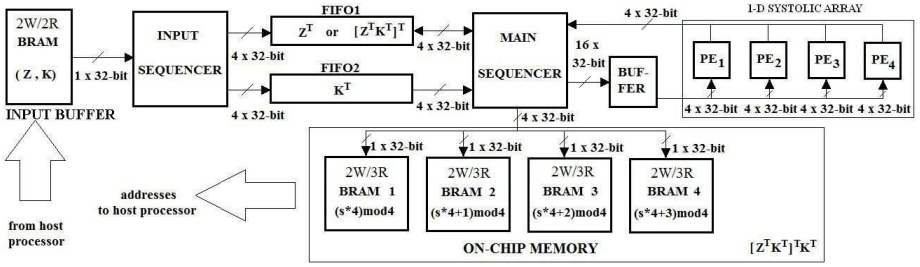

Figure 2: Tri-matrix multiplication architecture.

performed by taking into account the identity:

$$KZK^T = (Z^T K^T)^T K^T \tag{9}$$

as in that way the larger matrix K has to be transposed only once during the computation. We instantiate[3] two floating-point multipliers and two floating point adders that form our PE kernel. Computational flow is mapped onto four PEs that are not interconnected between themselves due to dimensionality of a visual EKF-SLAM. The lack of any additional internal buffer in our SA allows for higher overall speed of the design. Although we may have instantiated more PEs in the 1-D SA, the throughput already achieved of this design conforms to our real-time constraints, which will be more detailed in section 6. Each First In First Out (FIFO) structure contains 4 FIFOs of depth of $ceil((7N + 19)/4)$ 32-bit words[4]. P matrix is stored in four tri-port 2W/3R Block Random Access Memories (BRAMs)[5].

We refer to the computational model as given in pseudocodes in subsection 3.3 :

Phase I. Main sequencer pre-buffers the input stream onto PEs. Its task is to provide the SA with input data at each clock cycle in order to maximize the operational throughput. For phase I, Z^T and K^T matrices are being popped from FIFO1 and FIFO2, respectively. While popping, K^T elements are also being pushed in a circular fashion using asynchronous read/write operations in FIFO2 structure, as they are reused later in phase II (which can be also seen from equation 9). After initial SA latency, main sequencer pushes the resulting $[Z^T\ K^T]^T$ matrix in FIFO1.

Phase II. $[Z^T\ K^T]^T$ and K^T matrices are fetched from relevant FIFOs and loaded onto the SA. Multiplication result is stored directly into 2W/3R BRAMs. It is necessary to make use of tri-port BRAMs, because in phase II we have

[3]We make use of Xilinx IP Core Generator v14.2 to instantiate the floating-point arithmetic units.

[4]FIFOs are instantiated using Xilinx IP Core Generator v14.2 and physically implemented as dual-port BRAMs.

[5]2W/3R BRAMs are obtained by multiplexing a read port of one of 2W/2R (true dual-port) BRAM's ports

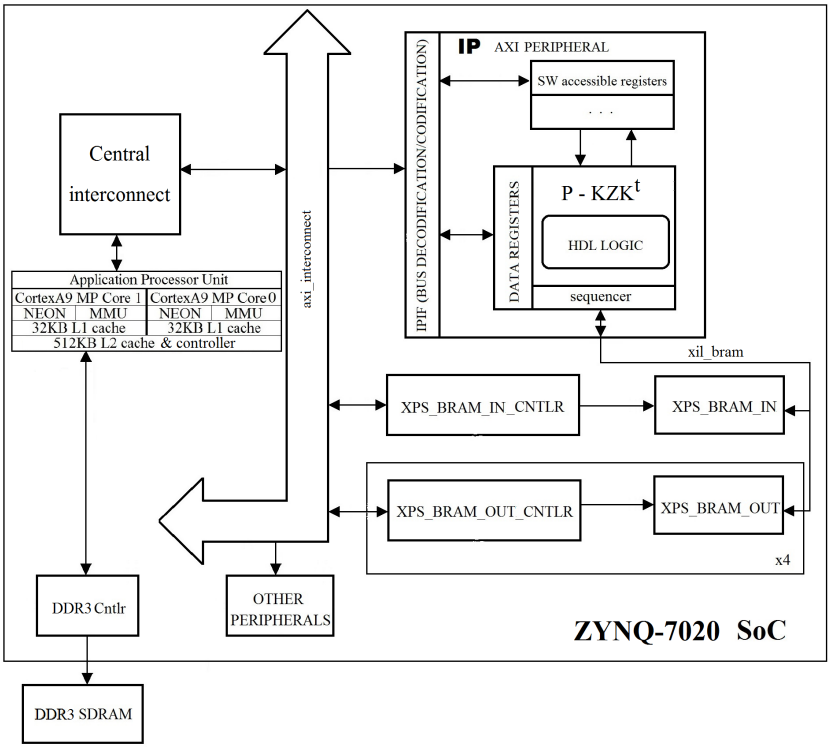

Figure 3: Our SoC implementation of the 3D EKF-SLAM application

simultaneous read and write operations with different addresses from/to on-chip memory (e.g. the elements of P are simultaneously fetched and stored the inner for loop in pseudocode lines 6 and 8). As one 1W/1R port is used for the store operation, the multiplexed 1R port is used for fetch operation. The third 1W/1R address line functioanlity rests for the host processor (execution of other equations in EKF block that require parts of P matrix).

On Figure 3 we present the entire Zynq-7020 SoC with our co-processor referred to as Intellectual Property (IP). It is attached as an Advanced eXtensible Interface (AXI) peripheral to a dual-core ARM processor[6]. The cross-covariance matrix P makes in total $((7N + r)(7N + r) + (7N + r))2$ bytes. Because it is stored entirely in the on-chip memory, pointer values are pre-set in program memory on each corresponding element in the on-chip memory so the host may access elements in P.

Communication with the host processor unfolds in three simple steps:

1. ARM loads matrices K and Z onto a fast on-chip memory (input buffer) via $axi_interconnect$ bus from the external DDR3 SDRAM and sends a

[6]The system is designed using Xilinx Platform Studio v14.2

"go" signal to the IP.

2. Our co-processor performs operations in equation 7, during which host is able to perform other tasks which don't evolve state estimation or update. During phase II, main sequencer stores the subtracted four results into four BRAMs, each containing the $(4s + j)mod4$ th element, where s is the current store cycle count and j is the number of the current row in a $(7N + 19)mod4$ cycle.

3. Co-processor asserts a "done multiplication" signal. By polling, the host verifies end of the execution of the operations in equation 7.

5. Design evaluation

In this section we give a theoretical estimate on developed multiplier's latency which is later going to be compared with experimental results (in Section 6) in order to evaluate the efficiency of the proposed SA-based design in terms of speed of execution. Afterwards we present an overview over resource and power consumption of the entire co-processor on the FPGA.

5.1. Multiplier Latency

After the analysis in section 3 we have:

$$L > L_0 + L_1 + L_2 \tag{10}$$

$$L_1 > max\left(\frac{n}{p}, \frac{n}{\sqrt{n}B}\right) \tag{11}$$

$$L_2 > max\left(\frac{n^2}{2p}, \frac{n^2}{\sqrt{2n^2}B}\right) \tag{12}$$

L_1 and L_2 are latencies after $Z^T K^T$ and $(Z^T K^T)^T K^T$ matrix multiplications respectively. L_0 is the latency due to processor-IP communication bandwidth. Concerning the matrix multiplication part, we have $M_1 = \sqrt{n}$ and $M_2 = \sqrt{n^2}$ because we have loaded all the corresponding matrices into two FIFO structures so that they could be fetched at each clock cycle. Furthermore, $B = 20$ owing to buffered input data and parallel storing of the results. Thus we are able to deduce that $L_1 > \frac{7N+r}{4}$ and $L_2 > \frac{(7N+r)^2}{8}$ as we have $p = 4$ processing elements. All delays greater than L_1+L_2 clock cycles are due to L_0 and because of pushing the two input matrices onto FIFOs.

Table 3: 1-D systolic array resource usage

Resources	**Add**	**Mult**	**Array**	**% IP**
LUTs	254	121	3000	41
Slice registers	271	72	2744	43
DSP48E1s	2	2	32	100
Latency	9	5	14	-
$F_{MAX}[MHz]$	380	368	311	-

Table 4: Accelerator resource usage with on-chip memory: comparison with the previous version

Resources	**Virtex5**	**Zynq-7020**
LUTs	8686	7313
Slice registers	10198	6366
DSP48Es/DSP48E1s	32	32
BRAMs [Kb]	1548	756
F_{MAX}[MHz]	119	198

5.2. Resource usage

In literature, efficient floating point units are implemented with several embedded DSP blocks and Look Up Tables (LUTs). In Table 3 an overview over resource usage of the SA-based multiplier versus IP is given. The one-dimensional systolic array consists of 8 deeply pipelined multipliers and 8 adders. In terms of logical resources, the systolic array makes use of 41% of the entire IP while the rest is used by input sequencer, main sequencer, buffer and other internal buffers with 32-bit delay lines. For speed issues, embedded multipliers (DSP48E1s) are instantiated in each PE of the SA.

In Table 4 we show the ratio of resource usage of the accelerator (with instantiated on-chip memory) versus all the available resources on Zynq-7020 device. Our map containts in total 20 AHP landmarks. A 36k BRAM is used for the input BRAM. Eight 18k BRAMs are used for instantiation of the FIFO structures (4 $\times$ 18k per FIFO) and sixteen 36k BRAMs are used to form the four memory banks[7] for storing 51kB of resulting data.

We conclude that the accelerator is efficient in terms of resource usage as it consumes only 11% (6368 out of 53200) of the configurable logic on the FPGA, leaving enough resources for other modules that might be required for the rest of SLAM (acceleration of vision process). BRAM usage (23%) for storing the cross-covariance matrix is minimal due to symmetry-related optimization. The speed issues are going to be detailed in Section 6. Comparing to our previous work on Virtex5 device, the new version is more efficient in terms of resource usage.

[7] A 32-bit 4096 word memory bank is implemented as 4 $\times$ 36k BRAM, instantiated in Xilinx Platform Studio v14.2. Each memory bank holds one fourth of the P matrix

Table 5: Visual EKF-SLAM accelerator's power distribution: comparison with the previous version

On-chip resource	**Virtex5 [W]**	**Zynq-7020 [W]**
Clocks	0.165	0.029
Logic	0.003	0.019
Signals	0.015	0.031
BRAMs	0.149	0.065
DSPs	0.001	0.01
PLLs	0.132	0.059
CPU	0.229	1
Leakage	1.469	0.079
Total	2.164	1.292

5.3. Power consumption

The FPGA power consumption is estimated using Xilinx XPower Analyzer tool - Table 7. Most of the power drain is due to dual core ARM processors (which run in low power mode at $667MHz$) but even with that fact in terms of power efficiency there is almost a one to two ratio in favor of Zynq-7020 implemented design.

6. Experimental results

After having synthesized our design, it's maximal operating frequency is 125MHz. Thus its peak throughput is $((p \times 4))FLOP \times 125MHz = 2GFLOPs$.

Using Gprof we obtain Table 6 and Figure 4 which show the cycles duration in each EKF block equation. We differentiate between the case when using and when not our accelerator in order to measure the impact of its use purely in terms of execution speed. Below the horizontal line are the equations related to the correction loop, which is run twenty times in an EKF loop. Some equations take longer to execute with acceleration, which is due to the sw/hw co-design; as when using the co-processor, the entire P matrix is stored onto the on-chip memory in order to leverage the most demanding operations. We then pay the price (though small) by having to transfer the data accros *axi_interconnect* bus. On the other hand, when not using the accelerator, the 51kB matrix is stored in program memory and all the operations are done by the host processors which takes a lot more time as it can be seen in this table. Thus, it is a profitable trade-off. With our design we made a 36-fold acceleration of EKF equation 7. On Figure 4 the EKF block throughput is given in function of corrected landmarks when using and when not the co-processor: for smaller maps (containing less than 21 AHP points) all landmarks are observed and corrected; for bigger maps there were made only 20 correction loops. We conclude that our design supports maps of up to 52 AHP landmarks with 20 corrections for an EKF block execution under real-time constraints.

Table 6: EKF equations cycles measured with 125 MHz counter

Equation no°	Accelerator	No accelerator
1	6700	6700
2	89000	86600
3	507000	500000
4	270	270
5	18800	18000
6	7500	7500
7	6100	220000

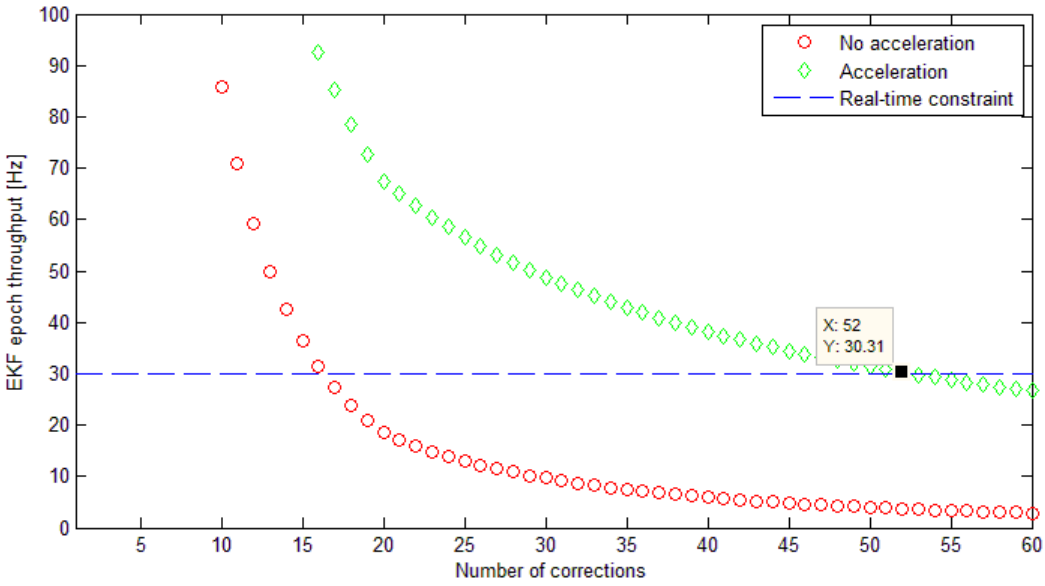

Figure 4: Measured EKF block throughput.

In Table 7 power per feature benchmarking is presented for our 3D EKF SLAM application (with 20 landmarks corrections). *Maximum number of landmarks* is the number of AHP landmarks retained in the SLAM's map that permit $30Hz$ EKF block period.

6.1. Measured multiplication latencies

For twenty landmark observations: the input sequencer takes $2\times2+159\times2 = 322$ clock cycles and the computational latency is $L_1 + L_2 \sim 3752$ clock cycles[8] which leaves us for the host processors-accelerator communication latency $L_0 \sim$

[8] Validated by ChipScope Pro Analyzer

Table 7: Power per feature metric for visual EKF-SLAM with AHP points

Hardware platform	**max. no. landmarks**	**Power [W]**	**Power/feature**
Intel i3 (2×1.8 GHz, 4GB RAM)	~ 100	34	0.34
Virtex5 SoC (400 MHz)	45	2.164	0.048
Zynq-7020 SoC (2×667 MHz)	52	1.292	0.024

$6100 - 4074 = 2026$ clock cycles. Computational latency is close to the predicted value in subsection 5.1 from which we confirm that the desing performs close to its theoretical optimum.

6.2. Future work

Having reduced the cycles number in EKF correction loop, new bottlenecks of the design become equations in the prediction loop : Eq. 2 and Eq. 3 (see Table 6). Besides VHDL code optimization for higher execution speeds (F_{MAX}), the focus of the future work is to exploit the existing processing power (the PEs) and FIFO structures (FIFO1 and FIFO2) so as to upgrade:

1. Structure of the SA to become a two-dimensional array by interconnecting existing PEs and
2. Its control logic.

7. Conclusion

In this paper we proposed an accelerator for a visual 3D EKF-SLAM application (without loop closure) that uses high-dimensional landmark parametrization for mobile platform localization. It is designed as an SoC on a Zynq-7020 device, yielding high performance metrics: minimal latency (close to theoretical optimum), low resource usage (11% of LUTs and 23% of on-chip memory) and low power per feature (24mW per single seven-dimensional landmark state size). It allows $30Hz$ throughput of an EKF block while containing 52 AHP parameterized points in SLAM's local map, which is more than a three-fold improvement in comparison with the application running purely on ARM hosts.

As it consumes small amounts of resources, other accelerators (i.e. for feature detection and/or feature matching) or soft-core processors (i.e. MicroBlaze) may be implemented on the FPGA.

These characteristics make our design suitable for implementation on embedded systems of limited size, memory and computational resources which require reliable localization functionality, such as micro-aerial vehicles and smartphones.

Acknowledgement

This work has been co-authored by Daniel Törtei Tertei, paid by the FUI-AAP14 project AIR-COBOT, co-funded by BPI France, FEDER and the Midi-Pyrenees region.

References

[1] H. Strasdat, J. Montiel, A. Davison, Real-time monocular slam: Why filter?, in: Proc. of IEEE International Conference on Robotics and Automation (ICRA), 2010.

[2] K. Underwood, K. Hemmert, Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance, in: Proc. IEEE Symp. Field-Programmable Custom Computing Machines, 2004.

[3] NVIDIA, NVIDIA Jetson TX1 Supercomputer-on-Module Drives Next Wave of Autonomous Machines (2015).
URL http://devblogs.nvidia.com/parallelforall/nvidia-jetson-tx1-supercomputer-on-module-drives-next-wave-of-autonomous-machines/

[4] G. Zhang, P. Vela, Optimally observable and minimal cardinality monocular slam, in: Proc. IEEE International Conference on Robotics and Automation (ICRA), 2015.

[5] G. Bresson, T. Feraud, R. Aufrere, P. Checchin, R. Chapuis, Real time monocular slam with low memory requirements, IEEE Intelligent Transportation Systems Transactions and Magazine.

[6] G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in: Proc. 6th IEEE and ACM Int. Symp. on Mixed and Augmented Reality, IEEE Computer Society, 2007.

[7] R. Mur-Artal, J. Montiel, J. Tardos, Orb-slam: a versatile and accurate monocular slam system, IEEE Transactions on Robotics (ITRO).

[8] C. Forster, M. Pizzoli, D. Scaramuzza, Svo: Fast semi-direct monocular visual odometry, in: Proc. IEEE International Conference on Robotics and Automation (ICRA), 2014.

[9] G. Zhang, P. Vela, Good features to track for visual slam, in: Proc. IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[10] T. Vidal-Calleja, C. Berger, J. Solà, S. Lacroix, Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain, Robotics and Autonomous Systems 59 (9) (2011) 654–674.

[11] A. J. Davison, Real-time simultaneous localisation and mapping with a single camera, in: Proc. Ninth IEEE International Conference on Computer Vision - Volume 2, ICCV '03, 2003.

[12] J. Solà, T. Vidal-Calleja, J. Civera, J. Montiel, Impact of landmark parametrization on monocular EKF-SLAM with points and lines, International Journal on Computer Vision.

[13] J. Montiel, Unified inverse depth parametrization for monocular slam., in: Proc. Robotics: Science and Systems (RSS), 2006.

[14] C. Roussillon, A. Gonzalez, J. Solà, J. Codol, N. Mansard, S. Lacroix, M. Devy, RT-SLAM : A Generic and Real-Time Visual SLAM Implementation., in: 8th International Conference on Computer Vision Systems, Sophia Antipolis (France), 2011.

[15] M. Fischler, R. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, Communications of the ACM.

[16] A. Gonzalez, J. Codol, M. Devy, A C-embedded Algorithm for Real-Time Monocular SLAM., in: 18th International Conference on Electronics, Circuits and Systems, Beyrouth, Liban, 2011.

[17] D. Botero, J. Piat, M. Devy, J. Boizard, An fpga accelerator for multispectral vision-based ekf-slam, Proc. IROS Workshop on Smart CAmeras for roBOTic applications (SCaBot), Vilamoura (Portugal).

[18] Xilinx, All Programmable FPGAs (2011).
URL `http://www.xilinx.com/content/xilinx/en/products/silicon-devices/fpga/`

[19] J. Piat, J. Piat, D. Marquez-Gamez, M. Devy, Embedded vision-based SLAM: A model-driven approach, in: Proc. Conference on Design and Architectures for Signal and Image Processing (DASIP), 2013.

[20] B. Vincke, A. Elouardi, A. Lambert, Implementation of EKF-SLAM on a Multi-Core Embedded System, in: IECON, Montreal, Canada, 2012.

[21] IEEE Standard for Binary Floating-Point Arithmetic, IEEE.

[22] V. Bonato, E. Marques, G. Constantinides, A Floating-Point Extended Kalman Filter Implementation for Autonomous Mobile Robots, Journal of VLSI Signal Processing.

[23] E. Sarraf, M. Ahmed-Ouameur, D. Massicotte, Fpga design and implementation of direct matrix inversion based on steepest descent method, in: Proc. 50th Midwest Symposium on Circuits and Systems (MWSCAS), 2007.

[24] S. Qasim, A. Telba, A. AlMazroo, Fpga design and implementation of matrix multiplier architectures for image and signal processing applications, International Journal of Computer Science and Network Security (IJCSNS) 10 (2).

[25] S. Tiwari, S. Singh, N. Meena, Fpga design and implementation of matrix multiplication architecture by ppi-mo techniques, International Journal of Computer Applications 80 (1) (2013) 19–22.

[26] D. T. Tertei, J. Piat, M. Devy, FPGA design and implementation of a matrix multiplier based accelerator for 3D EKF SLAM, in: Proc. Conference on ReConFigurable Computing and FPGAs (ReConFig), 2014.

[27] J. Solà, A. Monin, M. Devy, T. Lemaire, Undelayed initialization in bearing only slam., in: Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2005, pp. 2499–2504.

[28] S. Thrun, W. Burgard, D. Fox, Probabilistic Robotics, MIT Press.

[29] J. Spivey, Fast, accurate call graph profiling, Oxford University Computing Laboratory.

[30] L. Zhuo, V. Prasanna, Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on Reconfigurable Computing Systems, in: IEEE Transactions on Parallele and Distributed Systems, Vol. 18, No. 4, 2007.

[31] Z. Jovanovi, V. Milutinovi, FPGA accelerator for floating-point matrix multiplication, IET Computers and Digital Techniques.

[32] A. Castillo-Atoche, D. Torres-Roman, Y. Shkvarko, Towards real time implementation of reconstructive signal processing algorithms using systolic array coprocessors, Journal of Systems Architecture (2010) 327–339.