PAPER • **OPEN ACCESS**

Pipelined implementation of the IP-block for direct tracking on configurable Zynq 7020 SoC platform

To cite this article: R A Seliverstov and I I Shagurin 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* **498** 012018

View the article online for updates and enhancements.



IOP ebooks™

Bringing you innovative digital publishing with leading voices to create your essential collection of books in STEM research.

Start exploring the collection - download the first chapter of every title for free.

This content was downloaded from IP address 223.68.190.130 on 20/09/2019 at 05:18

IOP Conf. Series: Materials Science and Engineering **498** (2019) 012018 doi:10.1088/1757-899X/498/1/012018

Pipelined implementation of the IP-block for direct tracking on configurable Zynq 7020 SoC platform

R A Seliverstov and I I Shagurin

National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Kashirskoe shosse 31, 115409 Moscow, Russian Federation

Abstract. The possibilities that occur in using specialized hardware for the accelerated execution of a number of computational steps in solving the problem of spatial localization of an object using the modern algorithm Semi-Dense SLAM are considered. As a hardware platform for implementing the localization system, a programmable system is used on a Xilinx Zynq-7020 chip, which consists of two ARM Cortex-A9 processor cores and an array of programmable logic (FPGA). An estimate of the amount of hardware resources required to perform spatial localization is presented. It is shown that the application of the developed computational pipeline allows to significantly increase the localization rate. The results of the research can be used to design navigation systems for unmanned aerial vehicles and autonomous vehicles, as well as augmented reality systems.

1. Introduction

One of the primary tasks in the development of AUVs and other mobile robotic systems is to provide and maintain their robust navigation in the surrounding space. To solve this problem are used methods and algorithms for Simultaneous Localization and Mapping SLAM [1]. The existing methods for performing SLAM can be divided into two groups:
- methods based on the usage of Reference Points (Feature-Based SLAM),
- direct methods which take advantage of the full image (Direct SLAM).

The major concept behind the first group of methods is to divide the geometric evaluation process into two consecutive steps: first, the extraction of a set of features from the input image and then calculation of the position of the camera as a function of the location of those features. This distinction imposes an important limitation: information about the straight or curved edges of the surrounding space, which does not contain data corresponding to a given set of features, is discarded, although it may be useful for improving the overall accuracy of the algorithm. The main disadvantages of these methods are the high requirements for computational resources for hardware implementation of the processes of finding, identifying and evaluating the position of landmarks in three-dimensional space.

SLAM methods based on full images or direct methods (Direct SLAM) allow to avoid the main drawback of Feature-Based SLAM because in Direct SLAM, the camera position and the geometry of the surrounding scene are determined based on information about all pixels of the input image. Direct methods also allow the construction of complete and accurate maps of the surrounding space [2]; however, this requires the use of high-performance graphics accelerators (GPU) for real-time operation.

In this paper, the problem of spatial localization of an object is solved using one of the newest modification of classical direct methods – the Semi-Dense SLAM, which is based on the use of incomplete images. This approach allows to track the movement of an autonomous vehicle with high

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

precision using the most informative parts of the image (a subset of pixels for which the intensity gradient is higher than a certain threshold value and for which it is convenient to track the position of the robot in space), and at the same time makes it possible to consistently build accurate and large-scale maps of the surrounding area [3].

2. The Semi-Dense SLAM algorithm for spatial localization

Figure 1 shows a block diagram of the Semi-Dense SLAM algorithm.

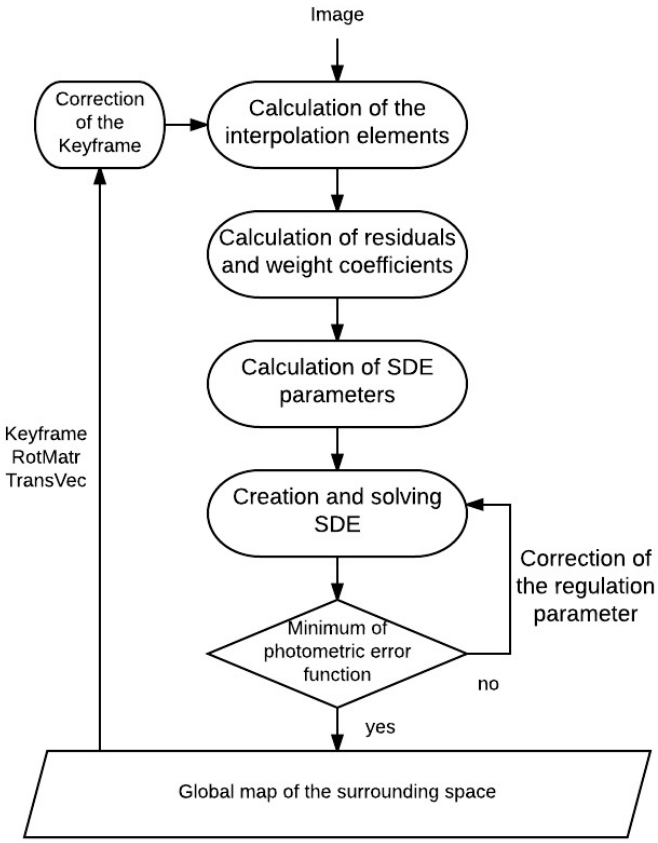


Figure 1. Block diagram of the Semi-Dense SLAM algorithm.

The algorithm is based on the use of a Keyframe, a special data structure that is stored in external memory and contains an image (with pixels determined by the x and y coordinates), as well as the image depth $\boldsymbol{D_i}$ and the depth variance σi [1]. A new Keyframe is formed after the camera is moved, when the distance between it and the current frame coming from the video camera exceeds a certain threshold value. At the stage of map construction, the previous Keyframe is decommissioned and added to the Keyframe graph. The global map of the surrounding space is represented in the form of a graph, the vertexes of which are KFs, whilst the edges are the parameters describing the rotation operations (the RotMat matrix) and the transition (TransVec vector) that the camera needs to take from the previous position to the next one, calculated by the algorithm (from one vertex of one graph to the other).

To describe the current position of the camera, an element of the Lie algebra is used - vector $\boldsymbol{\xi}_{ij}$, which contains the RotMat matrix and the TransVec vector, which determine the location of the current frame in relation to the previous frame. The value of this vector corresponds to the position of the minimum of the normalized photometric error function (1):

$$E_p(\boldsymbol{\xi}_{ij}) = \sum \left\| \frac{r_p^2(\boldsymbol{p},\boldsymbol{\xi}_{ij})}{\sigma_{r_p}^2(\boldsymbol{p},\boldsymbol{\xi}_{ij})} \right\| \quad (1)$$

where r_p is the residual, which is calculated as the difference in pixel intensities of the current frame $\boldsymbol{I_i(p)}$ and the corrected Keyframe $I_j\left(p, D_i(\boldsymbol{p}), \boldsymbol{\xi}_{ij}\right)$

$$r_p\left(\boldsymbol{p}, \boldsymbol{\xi_{ij}}\right) = I_i(\boldsymbol{p}) - I_j\left(p, D_i(\boldsymbol{p}), \boldsymbol{\xi}_{ij}\right) \quad (2)$$

$\sigma i -$ depth variance calculated with inverse depth $V_i = 1/Di$

$$\sigma_{r_p}^2\left(\boldsymbol{p}, \boldsymbol{\xi_{ij}}\right) = 2\sigma_I^2 + \left(\frac{\partial r_p(\boldsymbol{p}, \boldsymbol{\xi}_{ij})}{\partial D_i(\boldsymbol{p})}\right)^2 V_i(\boldsymbol{p}) \quad (3)$$

$D_i(p)$ – depth value for the described subset of pixels.

All operations are performed only for the selected subset of pixels $\boldsymbol{p}$ with an intensity value above a certain threshold.

At first, the Keyframe correction step is executed by changing the pixel projection in accordance with the current position of the camera. At the correction stage, the image contained in the KF is also calibrated taking into account the technical characteristics of the camera.

The minimum of the error function is calculated using an iterative optimization process using the Levenberg-Marquardt method [4]. At each iteration, the value of the position vector $\boldsymbol{\xi_{ij}}$ is determined, at which the local minimum of the error function [3] is reached. The increment of the vector $\delta\boldsymbol{\xi}^{(n)}$ for which $E_p\left(\boldsymbol{\xi}_{ij}\right)$ reaches a local minimum is calculated for the n-th iteration by a system of differential equations:

$$\delta\boldsymbol{\xi}^{(n)} = -(\boldsymbol{H} + \lambda\boldsymbol{I})^{-1}\boldsymbol{J}^T r\left(\boldsymbol{\xi}^{(n)}\right) = -(\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I})^{-1}\boldsymbol{J}^T r\left(\boldsymbol{\xi}^{(n)}\right) \quad (4)$$

where $\boldsymbol{J}$ is the matrix of the first derivatives of the error function (Jacobian), $\boldsymbol{H}$ is the matrix of the second derivatives of the error function (Hessian), $\boldsymbol{I}$ is the identity matrix, $\lambda \geq 0$ is the regulation parameter that improves the convergence of the algorithm by increasing the monotonicity of the function $E_p\left(\boldsymbol{\xi}_{ij}\right)$. Elements of the Jacobian are calculated in accordance with expression:

$$\boldsymbol{J} = \left.\frac{\partial r(\epsilon \circ \boldsymbol{\xi}^{(n)})}{\partial \epsilon}\right|_{\epsilon=0} \quad (5)$$

where ϵ is the linearization parameter. To calculate Hessian, we use the approximation $\boldsymbol{H} = \boldsymbol{J^T J}$. The necessary data for the formation of the Jacobian are determined at the stage of calculating the interpolation elements, where the pixel intensity increments dIx, dIy are calculated for the x, y coordinates and interpolated intensity value $\boldsymbol{C}$.

In the presence of substantial fluctuations of intensity, large values of the error function arise, which slow down the process of finding the minimum. To reduce the effect of such differences, normalization is obtained with the use of the matrix of such weighting coefficients $\boldsymbol{W}$ that limit the intensity values [5].

Taking into account the weighting coefficients, the SDE for finding the minimum of the error function is:

$$\delta\boldsymbol{\xi}^{(n)} = -(\boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J} + \lambda\boldsymbol{I})^{-1}\boldsymbol{J}^T\boldsymbol{W} r\left(\boldsymbol{\xi}^{(n)}\right) = -\boldsymbol{A}/(\boldsymbol{B} + \lambda\boldsymbol{I}) \quad (6)$$

where the parameters $\boldsymbol{A}, \boldsymbol{B}$ on the right-hand side of equations are determined by the expressions

$$\boldsymbol{A} = \boldsymbol{J}^T\boldsymbol{W} r\left(\boldsymbol{\xi}^{(n)}\right), \boldsymbol{B} = \boldsymbol{J}^T\boldsymbol{W}\boldsymbol{J} \quad (7).$$

To form the SDE, the Jacobian and the parameters $\boldsymbol{A}, \boldsymbol{B}$ are calculated, which are determined in the previous step by the values of interpolation elements, residuals and weighting coefficients.

The system of equations is solved with different values of the regulation parameter λ. The number of iterations depends on the size of the image. In the event of a global minimum of the photometric error function, the new position of the camera $\boldsymbol{\xi_{ij}}$ is fixed and added to the global map of the surrounding space. Otherwise, the value of λ is corrected and the solution of the system of equations is recalculated. For optimal choice of the values of λ, the technique described in [6] is used.

The iterative process of finding $\delta\boldsymbol{\xi}^{(n)}$ is finished in the following cases:

1) After performing the maximum number of iterations N (in this work, for the resolution of 320x240 pixels, the value N = 50 is assumed);

2) If the increment $\delta\boldsymbol{\xi}^{(n)}$ obtained at the last iteration turns out to be less than the predefined threshold value;

3) If the change in the error function obtained at the last iteration is less than a certain value (for example, 0.1%).

Fulfilment of these conditions ensures good convergence of the algorithm and reduces the number of iterations required to calculate the next camera position.

Calculations performed at various stages of the algorithm can be made using a universal processor (computer) or using specialized hardware blocks. In [7], an implementation of the Semi-Dense SLAM algorithm is described, which utilizes a processor with specialized hardware blocks for calculating the Jacobian and residuals. In the presented article, the possibility and efficiency of using the hardware-implemented pipeline for calculating all necessary parameters for the formation of a linear system of equations are considered, the solution of which allows to calculate the position of the object in the surrounding space.

3. The architecture of the spatial localization system

For the hardware implementation of the spatial localization algorithm, it is necessary to use a system with the following architecture:

- high-performance processor for solving the system of differential equations (SDE) with correction of the regulation parameter, general control of the system units, performance of necessary service functions;
- an array of programmable logic for the implementation of specialized blocks that perform the calculation of the parameters for the system of equations;
- sufficient memory to store a global map of the environment and necessary service information;
- a set of interface devices that provides interconnection between the system units and external devices.

As a hardware platform for the implementation of the proposed architecture, a programmable System on Chip Zynq-7020 FPGA SoC from Xilinx [8] is used, which meets the specified requirements to the greatest extent. Zynq-7020 contains two high-performance 32-bit Cortex-A9 processor cores with a maximum operating frequency of up to 766 MHz and an FPGA-type programmable logic array containing 85,000 logic cells (LUTs) and 106400 triggers. The SoC additionally contains interface blocks that allow the implementation of the high-speed AXI bus, 220 DSP blocks for performing multiply-accumulated operations, 4.9 Mb memory blocks and some other specialized units. The used SoC is implemented as part of the Digilent's ZedBoard development board, which also contains a dynamic RAM of 1 MB. This RAM is used as the external memory for the implemented spatial localization system.

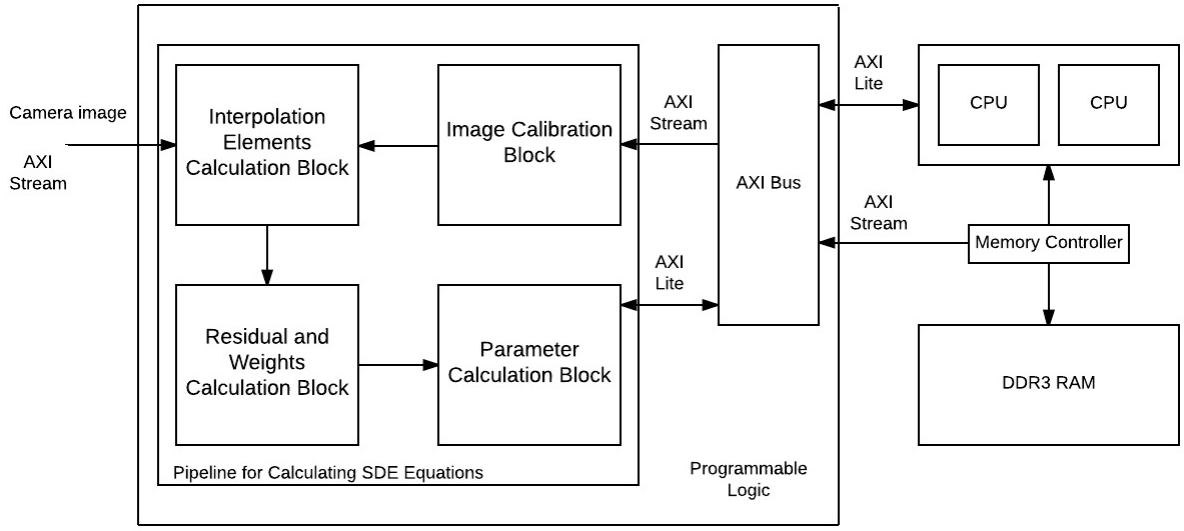


Figure 2. Architecture of the spatial localization system.

Figure 2 shows the architecture of the spatial localization system. Two Cortex-A9 processor cores perform the functions of a central processing unit (CPU), providing the overall control over the operation of the localization system and the solution of the SDE. A computational pipeline is implemented on programmable logic to calculate the parameters of the SDE. Separate stages of the pipeline perform correction of the selected Keyframe, calculation of interpolation elements, determination of residual and weighting coefficients, and calculation of the values of parameters necessary for solving the SDE. Transmission of information inside the system is provided by two types of AXI interface blocks: AXI Stream for streaming of video data and AXI Lite for transmission of control signals from the processor. The control of the external DDR3 DRAM, where the global environment map as well as the service information used during the system operation are stored, is provided by one of the CPUs with the help of a memory controller that is part of the SoC.

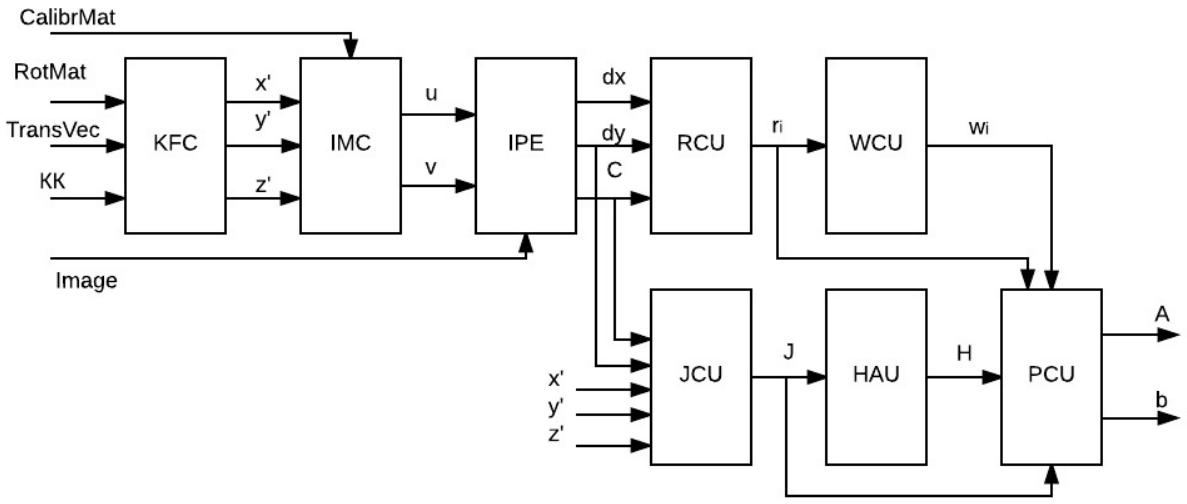


Figure 3. Microarchitecture of the computational pipeline.

Figure 3 shows the microarchitecture of the computational pipeline, which contains 8 operating units. The current Keyframe selected from DDR3, the RotMat matrix and the TransVec vector (the initial position vector $\boldsymbol{\xi}_{ij}$), which is set by the CPU, enter the input of the Keyframe Correction Block (KFC).

In this block, the projection of the Keyframe is changed by multiplying the pixels of the Keyframe by the RotMat matrix and adding the TransVec vector. The output of the KFC generates the corrected values of the coordinates x', y' and depth z', which are fed to the image calibration block IMC. In accordance with the pinhole camera model, this block divides the coordinates x' and y' by the value of the depth z' and the resulting values are multiplied by the calibration matrix CalibrMatr. This matrix takes into account the technical characteristics of the used video camera and is stored as the service information in the system memory. A set of corrected pixel coordinates of the Keyframe (u, v), corresponding to the camera position at the current time, are sent to the output of the IMC.

Using the data calculated on the previous stage of the pipeline, the block for calculating the interpolation elements IPE [7] determines the intensity increments dIx, dIy with respect to the coordinates x and y and the interpolated intensity value C, which are then fed to the block for calculating the residual coefficients RCU. This block calculates the value of the residual coefficients $\boldsymbol{r_p}$, which is determined by the expression (2). After that, the sums of intensities and the sum of intensity squares ($\sum c$ и $\sum c^2$) of the current frame as well as the sum of the intensities and the sum of squares of the intensity of the Keyframe ($\sum c_2$ и $\sum c_2^2$) are calculated for finding auxiliary parameters of Affine transformations.

The calculation unit for the weighting coefficients WCU generates a matrix of weights $\boldsymbol{W}$, the elements of which are determined depending on the value of the corresponding residual coefficient:

$w_i = 5.0/r_i$, if $r_i < 5.0, w_i = 1$ otherwise.

The corrected values of coordinates x', y' and depth z', obtained at the output of the IMC, as well as the calculated pixel increments $dIx,\ dIy$ calculated by the IPE block are fed to the block of calculation of Jacobian JCU. This block calculates the elements of the Jacobian in accordance with the expressions:

$J(1) = \frac{dx}{z}, J(2) = \frac{dy}{z}), J(3) = -\frac{x}{z^2}dx - \frac{y}{z^2}dy, J(4) = -\frac{xy}{z^2}dx - \frac{z^2+y^2}{z^2}dy, J(5) = \frac{z^2+x^2}{z^2}dx + \frac{xy}{z^2}dy, J(6) = -\frac{y}{z}dx + \frac{x}{z}dy$ (7)

The Hessian approximation block HAU computes the approximate value of the Hessian by transposing the Jacobian and multiplying its direct and transpose matrixes. The calculated values of the Jacobian and Hessian, as well as previously derived residual and weighting coefficients, are used by the PCU parameter calculation block to determine the parameters $A,\ B$, which enter one of the CPUs where the SDE is generated and solved. The results of the SDE solution, which determine the new position of the object, are added to the global map of the surrounding space, stored in the system memory.

4. The results of modelling the spatial localization system

To evaluate the hardware resources required for the implementation of the spatial localization system using the Semi-Dense SLAM method, the system was synthesized on the Zynq-7020 SoC platform. The programmable logic provides 85000 logical cells, 4.9 MB of available BRAM memory, and 220 DSP cores. Synthesis and verification of the system model were carried out by means of EDA software Vivado HLS [9].

The model was verified by comparing the results of the work of the synthesized model and the reference model written in C++. As the benchmark test actions, the real images from the video camera and the corresponding Keyframes formed during the test run of the software algorithm on the ROS operating system were used. Test actions were applied to the inputs of the synthesized system as well as to the reference model, which performed calculations in the C Co-simulation mode (в оригинале, только Co-simulation). The results of the calculations were written to the corresponding data sets and compared to an accuracy of 10^{-4}. The testing carried out for several frame sequences confirmed the correctness of the work of the synthesized model of the spatial localization system.

Table 1 shows the number of BRAM memory cells, DSP blocks, FF triggers, and LUT logical cells used in the synthesis of the computational pipeline and its main blocks IPE, (RCU +WCU), (JCU + HAU + PCU). The distribution of resources for the implementation of synthesizable constructions that were used in the high-level description of blocks in the C++ language (mathematical operations - Expression, buffers with sequential access - FIFO, sub-blocks Instance, obtained as a result of synthesis at a given hierarchy level, memory cells, - MUX, triggers - Register) is presented.

Table 1. Hardware resources for the pipeline implementation.

	Spatial Localization Block				**JCU + HAU + PCU**				**IPE**				**RCU +WCU**			
	BRAM	DSP	FF	LUT	BRAM	DSP	FF	LUT	BRAM	DSP	FF	LUT	BRAM	DSP	FF	LUT
Expression				2748				2205				313				230
FIFO		25	126	1046							126	1046				
Instance	4	44	13958	20613	2	14	5573	5514		26	2287	5224		30	5498	9875
Memory	128				2								128			
MUX								2271				605				2696
Register			10424	834			6069	320			2470	40				474
Total	132	69	24508	30813	4	14	11642	10310	0	26	5483	7228	128	30	1855	13275
Utilization	47%	31%	23%	57%	1%	6%	11%	19%	0%	11%	5%	13%	46%	14%	7383	25%

The spatial localization system receives new input data from the video camera on each clock cycle. The clock speed of the system $\boldsymbol{F_t}$ is 100 MHz. According to estimates made using the Vivado HLS EDA, the localization delay determined by the signal transit time along the most critical path of the computational pipeline $\boldsymbol{T_p}$ is 10,660 cycles (about 1 ms for a given $\boldsymbol{F_t}$ value).

5. Conclusion

The use of high-speed AXI Stream interfaces and hardware pipeline for the formation of SDE makes it possible to significantly improve the speed of performing the spatial localization of objects. According to the estimates, the localization delay associated with the formation of the CDS is reduced approximately 3-fold, compared to the solution described in [7]. The proposed architecture of the computational pipeline for determining the parameters of the CDS can be effectively used in the navigation systems of unmanned aerial vehicles, autonomous vehicles, as well as in the augmented reality systems.

References

[1] Engel J, Schops T and Cremers D 2014 LSD-SLAM: Large-scale direct monocular SLAM in *European Conference on Computer Vision (ECCV)*
[2] Newcombe R, Lovegrove S and Davison A 2011 *IEEE International Conference on. IEEE* pp 2320–2327
[3] Schops T, Engel J and Cremers D *Mixed and Augmented Reality (ISMAR), International Symposium on. IEEE* pp. 145–150
[4] More J 1978 *Numerical analysis* pp 105–116.
[5] Kerl C, Sturm J and Cremers D 2013 *Proc. of IROS* pp 2100-2106
[6] Marquardt D 1963 *Indst. Appl. Mat.* **11** 2 pp 431-441
[7] Boikos K and Bouganis C-S 2016 *International Conference on Field-Programmable Logic and Applications (FPL)*
[8] Xilinx 2012 Vivado Design Suite Tutorial *UG871 (v2012.2)*
[9] Xilinx 2015 ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide *UG850 (v1.5)*