UNIVERSITAT POLITÈCNICA DE CATALUNYA

Facultat d'Informàtica de Barcelona

MASTER THESIS

A Hardware Accelerator for ORB-SLAM

Author:
Raúl TARANCO

Supervisors:
Dr. Jose María ARNAU
Dr. Antonio GONZÁLEZ

Master in Innovation and Research in Informatics
High Performance Computing

ARCO Research Group
Computer Architecture Department

October 17, 2019

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstract

Facultat d'Informàtica de Barcelona
Computer Architecture Department

Master in Innovation and Research in Informatics
High Performance Computing

A Hardware Accelerator for ORB-SLAM

by Raúl TARANCO

Simultaneous Localization And Mapping (SLAM) is a key component of self-driving cars. It builds and updates a map of an unknown environment while keeping track of an agent's location within it. An effective implementation of SLAM presents important challenges due to 1) real-time inherent constraints and 2) energy consumption. In this project we study the state-of-the-art ORB-SLAM algorithms and investigate hardware techniques to accelerate its operation allowing to meet 1) and reduce 2). As a result we develop a hardware accelerator for ORB feature extraction that achieve 8x speedup and 2000x energy consumption reduction compared to the software implementation.

Keywords: orb, slam, orb-slam, Hardware Accelerator

Acknowledgements

I would like to thank my supervisors, Antonio González Colás and Jose María Arnau Montañés, for giving me the opportunity to work on this project and for all the great advise they gave me during this master thesis.

I thank my family for supporting me and for giving me the opportunity to study this master at UPC.

Finally thank you to María, for all her love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the motivation of the project, its objectives and the structure that follows the rest of the document.

1.1 Motivation

Automation in transport is increasing rapidly. This trend is supported, in part, by the relatively recent proliferation of Self-Driving Cars (SDCs) systems. Currently, 1.35 million people die each year as a result of road traffic crashes [1] and about 94% of these accidents are caused by human errors [2]. SDCs systems have the potential to radically change the automotive industry, offering safer vehicles with which to achieve a huge reduction in the number of accidents.

The recent advances in artificial intelligence and machine learning are leading to increasingly reliable and robust SDCs systems that could represent the solution to prevent traffic accidents in the future. However, such systems are based on algorithms that are extremely expensive from the computational point of view, with a demand for processing large amounts of data and inherent real-time restrictions. The hardware solutions that allow executing these algorithms are extremely expensive and exhibit high energy consumption, which affects both the economic cost and the autonomy of the vehicle [3], representing one of the main drawbacks for its mass commercialization.

One of the key tasks for a self-driving vehicle is to be able to localize itself in the surrounding environment. The use of external infrastructures, such as GPS, could be an option when solving this problem. However, the use of such systems can be expensive, not available or inaccurate. Therefore, it is convenient that localization is performed by processing information from on-board sensors similarly as animals and humans do.

On-board sensors can be classified as proprioceptive and exteroceptive sensors [4]. Proprioceptive sensors allow the agent to obtain measurements like velocity, position change and acceleration. Exteroceptive sensors provide information from the external world. Vision systems composed of one or more cameras are the most common examples of this type of sensors.

The simplest method to perform localization based on on-board sensors, consists on processing the information coming from motion sensor to estimate the position as part of a process called odometry. Although this method may be appropriate for

FIGURE 1.1: ORB-SLAM [7] system processing an image from the camera of a car (lef) and the representation of the environment created (rigth).

certain situations in small controlled environments, the estimations made end up accumulating errors making the trajectory to drift from the real one. For this reason, the need to use a map arises, since it allows drift-free localization.

Mapping is the process of creating a representation (map) of an unknown environment by using mainly exteroceptive on-board sensors. However this map may not be available or the environment may change. For all these reasons, it is desirable that an agent, for example a SDC, has the ability to create and update a map of the unknown surrounding environment while keeping track of its position within it. This challenging problem is called Simultaneous Localization and Mapping (SLAM) [4][5].

Among exteroceptive sensors, vision sensors are the most promising alternative because cameras are relatively inexpensive and compact providing a vast amount of information of the environment. Other alternatives employ lidar [6] (laser) technology. Although the systems based on lidar have great precision, they require expensive hardware.

SLAM systems that mainly use vision are known as Visual SLAM. A Visual SLAM system can use an infrastructure with several configurations among which are monocular and stereo [4][7]. In addition, these configurations can be enriched with data from proprioceptive sensors [8][9]. Figure 1.1 shows a running Monocular Visual SLAM system.

There are two main approaches to perform Visual SLAM [10]: feature-based and direct methods. Feature-based methods process images to extract distinctive interest points (keypoints) that can be reliably and repeatedly detected in images of the same scene, ideally under different viewpoints and illumination conditions [9]. In contrast to feature-based methods, direct methods directly use input images from a set of cameras without any abstraction, accessing the pixel intensities directly and estimating its associated depth using optimization techniques to minimize the photometric error [10][11].

In this project, we aim to develop a hardware accelerator for one of the state-of-the-art Monocular Visual SLAM solutions: ORB-SLAM [12][7]. We perform a characterization of this solution to detect the main bottlenecks to focus our efforts. We found that the pre-processing stage, in which the images coming from the cameras are handled, constitutes one of the most compute-intensive parts. In this stage, Oriented FAST and Rotated BRIEF (ORB) [13] features are extracted from the images of the camera. Thus, we choose to develop a hardware accelerator specifically designed to extract ORB features in the same way software-based implementations of

ORB-SLAM extract them.

The accelerator has been evaluated to find the solution that offers the best compromise between performance, cost and energy consumption. Although we could have opted for an Field-Programmable Gate Array (FPGA) platform, this evaluation has been carried out considering an Application-Specific Integrated Circuit (ASIC) solution.

1.2 Objectives

The main objective of the project is to reduce the cost and energy consumption of the hardware used for ORB-SLAM, while improving the performance. This major main objective can be broken down into several objectives:

1. Perform a characterization of ORB-SLAM, detecting the critical parts and main bottlenecks. It is also necessary to estimate the cost, performance and energy consumption of these parts.
2. Propose a hardware architecture to accelerate ORB-SLAM based on the characterization performed.
3. Quantify the performance and energy consumption of the proposed hardware architecture, establishing the improvement they represent with respect to the state-of-the-art.

1.3 Document structure

The remainder of this document is divided into five more chapters:

- Chapter 2 provides a basic description of ORB-SLAM emphasizing on the localization part, the extraction of keypoints and their description.
- Chapter 3 describes the hardware development and evaluation methodology followed during the project.
- Chapter 4 contains the characterization of ORB-SLAM with detailed information about the critical parts that affect the performance.
- Chapter 5 contains the description of the proposed hardware architecture to accelerate the ORB features extraction. Furthermore, an evaluation of the performance, energy and area of the solution based on the experimental results is presented.
- Chapter 6 provides a summary of the work done in the project and some promising paths to follow in the future, in order to enhance the proposed accelerator.

Chapter 2

Background

This chapter provides a basic description of ORB-SLAM emphasizing on the localization part, the extraction of keypoints or features, and their description.

2.1 ORB-SLAM

ORB-SLAM [12][9] has been ranked top of the available open source algorithms for monocular localization under the KITTI dataset [14] among others benchmarks. This system is able to estimate the real trajectory that an agent equipped with a camera traces while building a sparse representation of the surroundings in a wide variety of situations. We are particularly interested in its performance in the context of self-driving cars.

Figure 2.1 shows and overview of the system. The algorithm divides the work into three threads for tracking, local mapping and a loop closing. Frames coming from the visual system are first processed by the tracking thread. These threads operate into a map data structure that keeps the system state. The map is composed of the following elements:

- Map points, that stores the 3D position in the world coordinate system of the points detected to build the map, its corresponding ORB descriptor, the mean viewing direction vector and a range of distance from which the point can be observed.
- Keyframes, that stores the camera pose, camera intrinsic parameters and all the ORB features extracted from that frame.
- Co-visibility graph, a weighted unidirectional graph that relates keyframes (graph nodes) whit other keyframes creating and edge, if there are a certain amount of shared map point observations between them.
- Spanning tree, a connected subgraph of the co-visibility graph with a minimal number of edges (the number of shared map point to create and edge between two frames is more strict).

In addition, a place recognition system is embedded into the system to perform loop detection and relocalization.

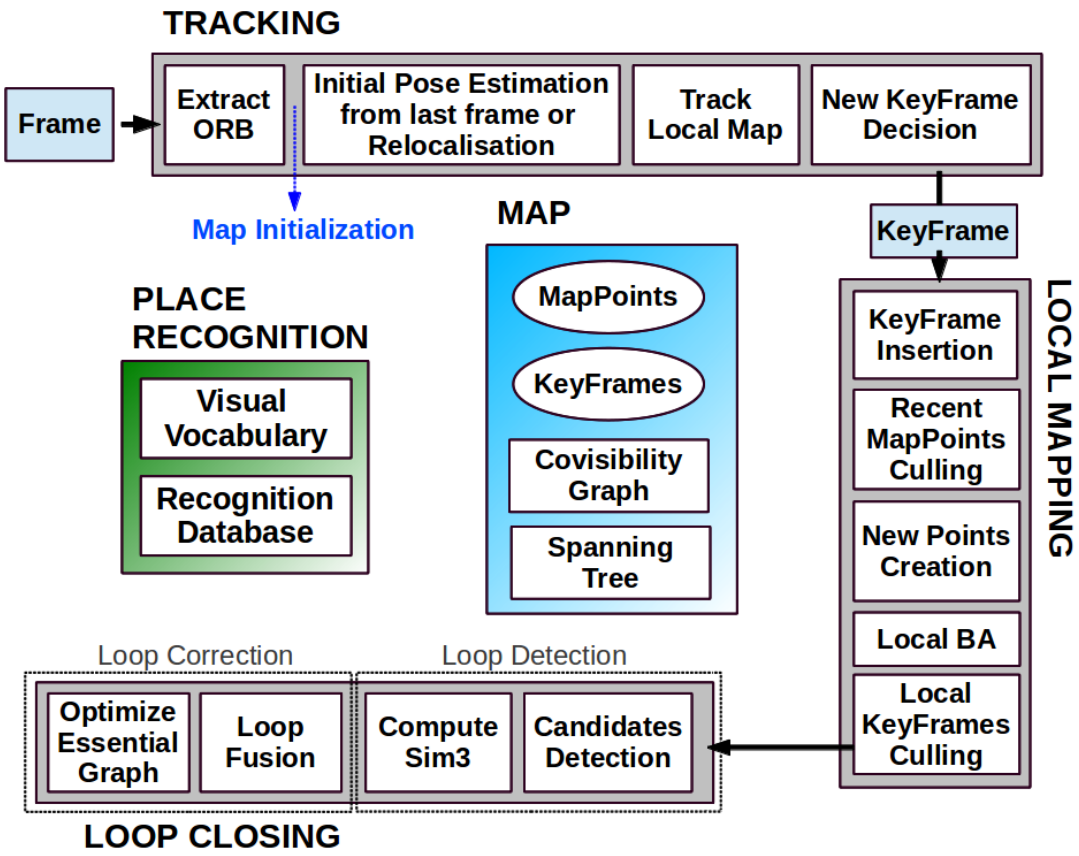

FIGURE 2.1: ORB-SLAM system overview [7]

The following sections briefly describe the steps performed at each stage of the algorithm. ORB-SLAM is able to operate in two modes: SLAM and Localization only mode. The first mode performs the localization and mapping and the second uses a static map and only performs the localization within it. In this work, we make use of the localization only mode, thus, more detail of this part is provided. For more details about the other parts or the whole system, it is recommended to review the related publications cited, in particular [9][12] and [7].

2.1.1 Automatic Map Initialization

The map initialization is performed in the fist stages of the algorithm with the objective of estimating the relative pose of the camera between two frames to triangulate an initial set of map points. The two frames are processed to extract the ORB features trying to find initial correspondences. If enough correspondences are found, two geometrical models are computed in parallel.

One model is suitable for planar scenes and the other for non-planar scenes. The model that better explains the scene based on a score is selected. With these model, multiple motion hypotheses are tested to see if one is significantly better than the others. If this is the case, a full bundle adjustment is done, otherwise the initialization process starts over.

Bundle adjustment is a technique that optimizes the position of the points in the world coordinate system and the poses of the camera from which each frame was

captured, minimizing the re-projection error [15].

2.1.2 Tracking

The tracking part localizes the camera within the map and decides when to insert a new keyframe. As part of this stage, the ORB extraction is performed. Since an accelerator has been proposed for this step, we have decided to leave the explanation of this part as a separate more detailed section (section 2.2).

After pre-processing the input frame, a constant velocity motion model is used to predict the new pose of the camera from the last known pose (last frame). Then, the algorithm try to find enough map points in the current frame that were previously seen in the last frame. If this initial pose estimation fails, a global re-localization is performed, using the place recognition system.

If the tracking was not lost, the co-visibility graph of keyframes is used to get a local visible map. This local map consists of keyframes that share map points with the current frame, the neighbors nodes of these keyframes and a reference keyframe which share the most map points with the current frame. Later, the camera pose is optimized with all the map point found in the frame.

Finally it is decided if the current frame becomes part of the set of keyframes stored by the system on the map.

2.1.3 Local Mapping

Whit every new keyframe, the local mapping steps are performed. First, the new keyframe is inserted into the co-visibility graph adding a new node and updating the edges according the shared map point with other keyframes. Then, new map points, those created in the las 3 new inserted keyframes, are tested to ensure that are trackable and not wrongly triangulated. To this extend, the new map points must be found in more than 25 % of the frames in which it is predicted to be visible during this 3 keyframe insertion period.

In the next step, new map point are created by triangulating ORB features from connected kewframes in the co-visibility graph looking for a match with an unmatched keypoint in other different frame. Those matches must fulfill the epipolar constraint.

Finally a local bundle adjustment is performed before the local keyframe culling: Keyframes whose 90% of associated map points can be seen by three other keyframes in the same scale-level, are discarded. This eliminates keyframes that do not provide much information useful for mapping.

2.1.4 Loop Closing

The loop closing thread takes the last kewframe added to the co-visibility graph and ties to detect and close loops: detect that the system have returned to a previously observed point. The detection of a loop adds restrictions that can help to improve the estimation of the poses of the camera and positions of the map points.

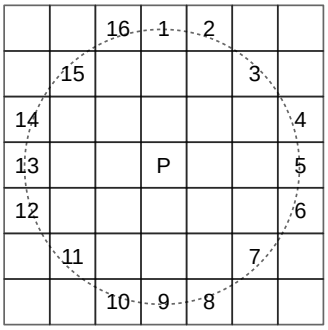

FIGURE 2.2: Bresenham circle or radius 3 showing the pixel access pattern for FAST segment test around the candidate pixel P.

2.2 ORB

ORB-SLAM uses ORB [13] features because of the speed in which these features can be extracted from images and their desirable properties such as viewpoint, illumination and rotational invariance [12].

ORB builds on the well-known FAST [16][17] feature detector and the BRIEF [18] descriptor. Both of these techniques are attractive because of their good performance and low cost. However, the convenient properties referred to above, do not emerge from the combination of this tandem by itself. ORB is the result of a combination of FAST and BRIEF, with a method to compute a orientation of the FAST features detected.

To understand how ORB works, it is necessary to know how each of these elements is computed. This is described in the following sections.

2.2.1 FAST

Features From Accelerated Segment Test (FAST), first introduced in [16], is a corner detector that can be used to extract feature points. FAST performs a test to classify a candidate pixel, p, as corner/no corner. This test consists on compare the intensity of p with the intensities of the 16 pixels that form a Bresenham circle around the candidate, as shown in figure 2.2. A feature is detected at the candidate pixel p, if the intensities of at least $n = 12$ contiguous pixels out of the 16 are all above or all below the intensity of p by some threshold, t.

The test for this condition can be optimized by examining pixels 1, 9, 5 and 13, to reject candidate pixels more quickly, since a feature can only exist if three of these test points are all above or below the intensity of p by the threshold.

However, there are a few limitations to this method [17]:

- For $n < 12$ the high speed reject test does not work.
- The order in which the 16 pixels are queried implies implicit assumptions about the distribution of feature appearance.
- Multiple features are detected adjacent to each other.

The first two issues listed above are addressed with a machine learning approach that can be consulted in [17].

The last one is addressed using non-maximal suppression. Non-maximal suppression is a post-processing method that allows filtering corners based on a measure or score. Only the corners with the score local maxima within a neighborhood prevail. There are many definitions of the score, V, of a corner, for example:

- Sum of absolute difference between p and 16 surrounding pixels values.
- The maximun value of t for which p is still a corner.
- Harris corner measure [19].

Typically, the candidate's neighborhood is defined as the area within a fixed-sized square patch centered on the considered pixel.

2.2.2 oFAST: FAST Keypoint Orientation

ORB uses the intensity centroid [13] to add the orientation component to FAST. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector can be used to impute an orientation.

The moment of a patch can be defined as [20]:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y), \tag{2.1}$$

where $I(x,y)$ is the intensity of the pixel at the relative position x,y within the patch. With this definition it is possible to compute the orientation centroid:

$$C = (\frac{m_{01}}{m_{00}}, \frac{m_{10}}{m_{00}}) \tag{2.2}$$

An then compute the angle of the vector formed between the center point of the corner, O, and the centroid C, $\vec{OC}$.

$$\theta = atan2(m_{01}, m_{10}) \tag{2.3}$$

$Atan2$ is the quadrant-aware version of the arctangent.

In addition, it is possible to compute the $sin(\theta)$ and $cos(\theta)$ using the moments in the following way:

$$sin(\theta) = \frac{m_{10}}{\sqrt{m_{01}^2 + m_{10}^2}}, \qquad cos(\theta) = \frac{m_{01}}{\sqrt{m_{01}^2 + m_{10}^2}} \tag{2.4}$$

oFAST is the combination of the application of the segment test to determine if a pixel is a corner and the computation of the orientation. This information can be used to generate features.

2.2.3 BRIEF

The BRIEF descriptor [18] is a bit string description of an image patch constructed from a set of binary intensity tests. A binary test, τ, is defined by:

$$\tau(p;x,y) = \begin{cases} 0 & , p(x) > p(y) \\ 1 & , p(x) \geq p(y) \end{cases}, \tag{2.5}$$

where $p(x)$ is the intensity of p at a point x. The feature is defined as a vector of n binary tests:

$$f_n(p) = \sum_{1 \leq i \leq n} 2^{i-1} \tau(p;x_i,y_i) \tag{2.6}$$

It is recommended to smooth the image before performing this tests, for example with a Gaussian blur filter. The vector length usually is $n = 256$

2.2.4 rBRIEF: Steered BRIEF

One of the most attractive features of ORR is the in-plane rotation invariance. To achieve this, the binary test coordinates of BRIEF are rotated according to the orientation obtained previously for FAST features.

For any feature set of n binary tests at location (x_i, y_i), define the $2 \times n$ matrix:

$$S = \begin{pmatrix} x_1, ..., x_n \\ y_1, ..., y_n \end{pmatrix} \tag{2.7}$$

Using the patch orientation θ and the corresponding S_θ it is possible to compute a steered version of the binary tests original positions:

$$S_\theta = R_\theta S, \tag{2.8}$$

2.2.5 Image pyramid

FAST does not produce multi-scale features but scale invariance is desirable. To achieve this, a scale pyramid of the images is used. At each level, the image from the previous level is subject to repeated smoothing and subsampling.

2.2.6 Dynamic threshold

ORB-SLAM employ an additional technique. Each frame is divided into a grid of approximately 30×30. In each tile of the grid, the algorithm tries to extract FAST corners with using a default threshold. If no corners were found, the algorithm tries again to extract corners but now using a lower threshold.

Chapter 3

Methodology

In this chapter, we review the experimental methodology employed throughout this master thesis in order to develop and evaluate the architecture of the proposed accelerator.

3.1 Hardware Development Methodology

Figure 3.1 illustrates the different stages we take in order to develop the hardware accelerator. We extensively use the open-source python-based hardware generation, simulation and verification framework, PyMTL [21]. The use of this framework enable us to concurrently model the hardware at different abstraction levels:

- Functional level (FL): At this level, models implement the functionality but not the timing constraints of a target. Because the functionality was already established beforehand, the implementation at this level served to strengthen the understanding of the algorithms and to construct a model whose results served as a reference when testing.
- Cycle level (CL): Models capture the cycle-approximate behavior of a hardware target. Simulation at this level allows us to quickly discard or take design paths in our architecture based on the simulation predicted results.
- Register Transfer Level (RTL): RTL models give us a cycle-accurate and resource-accurate representation of hardware. PyMTL is able to translate the RTL model to Verilog which allows us to use other Electronic Design Automation (EDA) tools to synthesize and estimate performance, area, power and energy accurately.

This methodology involves the vertical integration of all levels of abstraction. The design is done with a top-down approach. First, the application's target algorithm is studied, then an FL implementation is developed. After verifying the correct operation, the design entities can be migrated to CL or RTL. One of the benefits of using PyMTL, is that this whole process is unified under the framework. This allows the designer to combine FL, CL and RTL models into a multi-level simulation and verify them iteratively. The process ends when all the parts of the design have been verified and tested at RTL.

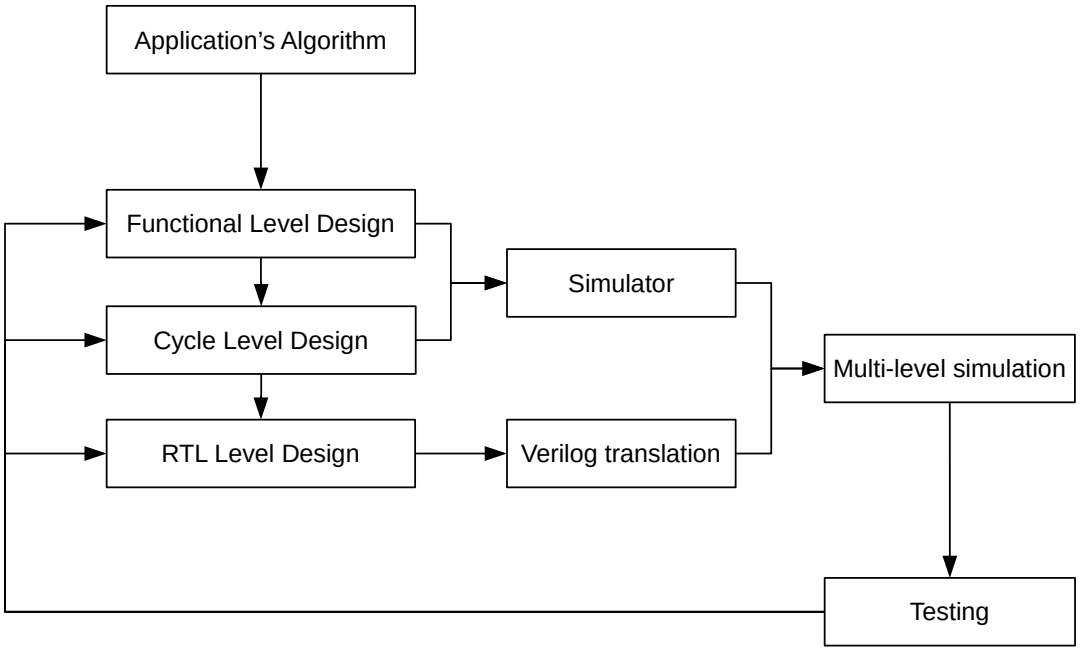

FIGURE 3.1: Methodology followed for designing the hardware accelerator.

3.2 Analysis Methodology

This section describes the methodology used to perform the characterization of the original algorithm and the proposed accelerator solution.

3.2.1 Datasets used

Both the characterization of the original ORB feature extraction algorithm used in ORB-SLAM and the proposed accelerator evaluation was performed using the KITTI dataset [14]. We use the odometry benchmark that comprises various recording of cameras from drivings around the city of Karlsruhe. This dataset was originally used to benchmark ORB-SLAM [9], consequently its choice for the various analysis performed was convenient. The odometry benchmark consists of 22 grayscale stereo sequences, saved in lossless PNG format.

FIGURE 3.2: One of the frames from the KITTI odometry benchmark [14].

In particular, we use the sequence 00 of the odometry benchmark. Figure 3.2 shows one of the frames of aforementioned sequence. In addition, table 3.1 show a brief summary of the relevant sequence properties.

Number of frames	4541
Image Resolution	1241 x 376
Type	Grayscale

TABLE 3.1: Properties of the Sequence 00 (odometry benchmark).

3.2.2 CPU baseline

A non-functional evaluation of ORB-SLAM is necessary in order to identify the critical parts and bottlenecks on which to focus our efforts. Several tests were carried out using an Intel(R) Core(TM) i7-7700K CPU. The specifications of the CPU are shown in table 3.2.

CPU	Intel(R) Core(TM) i7-7700K
Number of cores	4
Number of threads	8
Technology	14 nm
Frequency	4.2 GHz
L1, L2, L3	0.25 MiB, 1 MiB, 8 MiB
TDP	91 W

TABLE 3.2: CPU specifications.

Parameters

ORB-SLAM can be configured through the tuning of a series of parameters. We use the default parameters recommended by the author. The default parameters related to ORB are shown in the table 3.3.

Target number of features	2000
Scale factor	1.2
Number of levels	8
Initial FAST threshold	20
Minimum FAST threshold	7

TABLE 3.3: ORB parameters used for characterization.

Execution Time

The ORB-SLAM source code was modified inserting profiling instructions in order to be able to measure the execution time spent in each part of the algorithm through the use of calls to C++11 `std::chrono::steady_clock`.

Energy and Power

We employ the Intel RAPL library [22] to measure energy consumption of relevant algorithm sections.

Equation 3.1 was used to obtain the power dissipated according to the energy and execution time measurements done.

$$Power = \frac{Energy}{ExecutionTime} \tag{3.1}$$

3.2.3 Hardware accelerator

Once an implementation of the architecture designed according to the methodology described in section 3.1 is developed, it is necessary to evaluate the execution time, frequency, power, energy and area of it.

We use Yosys [23] and the open-source 45 nm technology Process Design Kit (PDK) provided by North Carolina State University (NCSU), FreePDK45 1.4 [24], to synthesize the accelerator.

Execution Time

In order to compute the execution time, it is necessary to perform two steps. First, the number of cycles required by the accelerator to process a set of relevant input data is calculated. Then it is necessary to know the critical path with which to determine the maximum frequency of operation. This can be performed using Yosys again. Finally the execution time can be computed using equation 3.2.

$$ExecutionTime = \frac{Cycles}{Frequency} \tag{3.2}$$

Area

Yosys is able to enumerate the number of gates and elements used in a synthesized design. In addition, the library cell contains area values for each element. Our approach to estimate the area is to perform the sum of the area of all of the elements needed to synthesize the accelerator.

Energy and Power

The power dissipated by the accelerator was estimated using Synopsys Design Compiler [25]. Using equation 3.1 we are able compute the energy consumption.

Chapter 4

Analysis of a CPU baseline system

In this chapter, we present the results obtained from the characterization performed to ORB-SLAM on the Intel platform described in table 3.2. This characterization allows us to justify ORB feature extraction, as one of the most interesting parts to accelerate.

4.1 ORB-SLAM

As stated in the previous chapter, we use the sequence 00 of the odometry benchmark as input data. We use the tracking only mode of ORB-SLAM but there is no available way to load a static map without which the results would not be realistic. To solve this issue, first, we run ORB-SLAM in slam mode. Then we activate the localization only mode and start measuring the latency of each part of the algorithm. Note that all the results shown in this section are an average of the values obtained in each frame of the sequence.

Furthermore, we perform this test trying to extract a different number of features to be able to observe the sensitivity of the measurements to the variation of this parameter. Figure 4.1 shows the absolute results measured. Viewing the results obtained we can draw the following conclusions:

- *Pre-process input* step is the one with the highest latency followed by *Track local map*.
- The algorithm loses the track with less than 2000 features. This can be inferred by observing that *Relocalization* step consumes a significant part of the time only when extracting less than 2000 features.
- Latency is increased with 3000 features but does not improve the functional performance of the algorithm.

Figure 4.2 shows the normalized values. This allows us to clearly see the conclusions stated above. The *pre-process input* step consumes approximately a 60% of the total execution time so it may be convenient to reduce its latency using a specifically designed hardware accelerator. At most, we can achieve a 2.5x speedup in the whole algorithm accelerating this part.

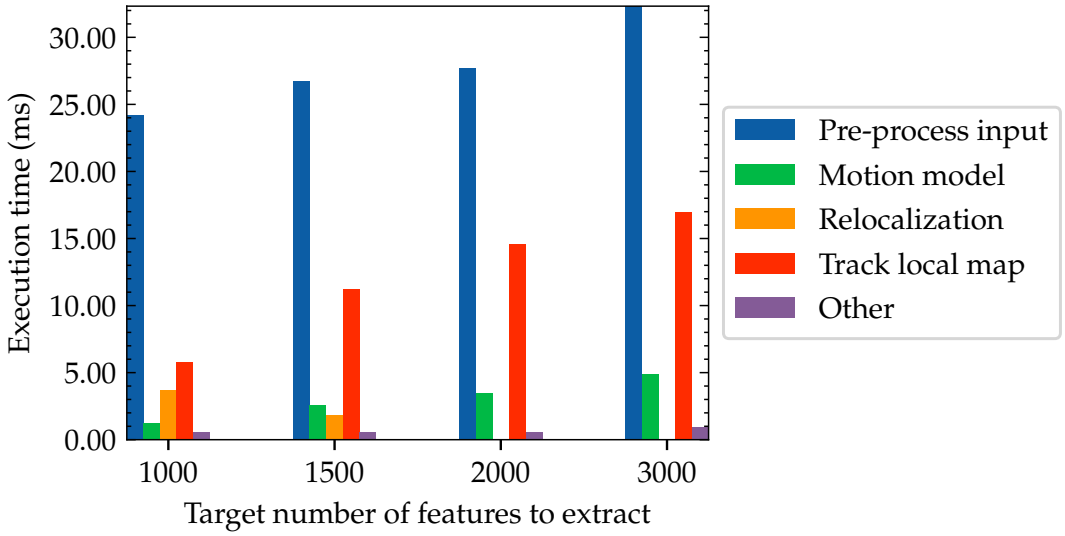

FIGURE 4.1: Average execution time of each relevant part of the tracking task per frame for a varying number of target features.

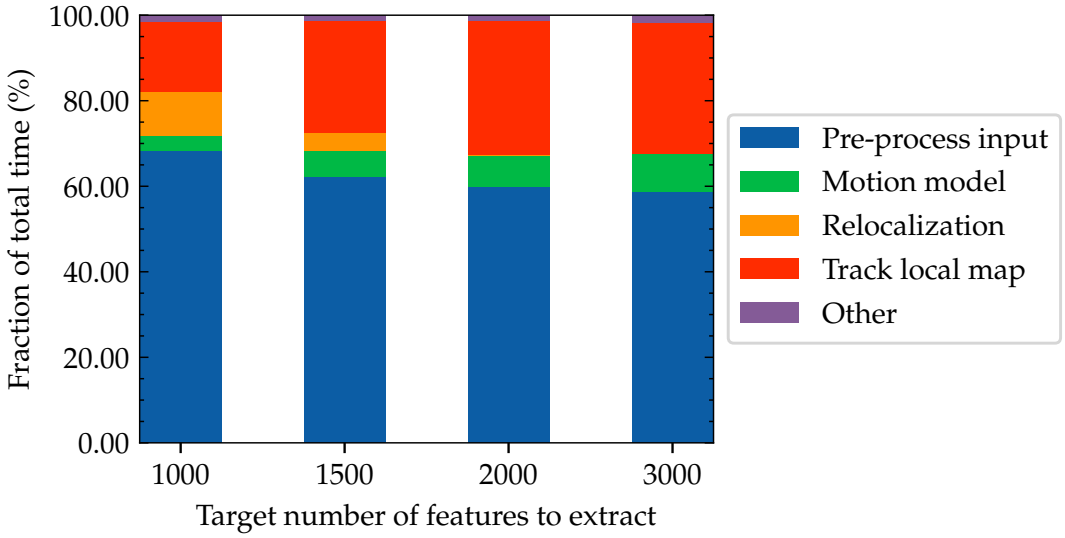

FIGURE 4.2: Average relative (normalized) execution time of each part of the tracking task per frame.

4.2 Pre-process input

In the previous section we have shown that the *pre-process input* step has the highest latency in average. In the *pre-process input* step, the ORB features are extracted from the incoming flow of images from the camera. Note that the extraction of ORB features is divided into: the generation of an scale image pyramid, the detection of features *oFAST*, the non-maximal suppression and the generation of the *rBRIEF* descriptors. Figure 4.3 shows the impact of each of these parts on the ORB extraction.

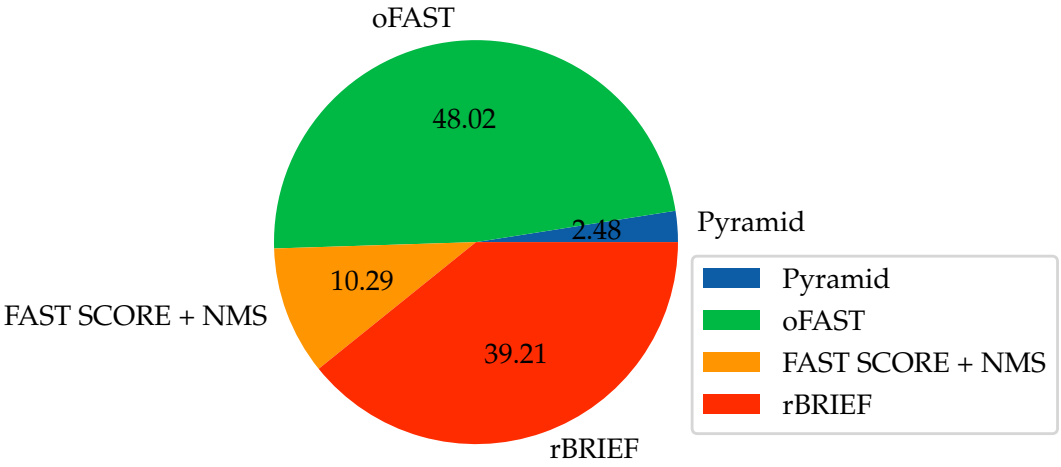

FIGURE 4.3: ORB extraction average execution time breakdown.

Chapter 5

ORB accelerator

In this chapter we describe the architecture of the proposed ORB accelerator. We first provide an overview of the architecture. Then, we dig deeper into the details of the design explaining the more relevant details in order to understand its operation. Finally, we show some experimental results and a comparison with the CPU baseline system. Additionally, the source code is available at [26].

5.1 Hardware Architecture

Figure 5.1 illustrates the architecture of the ORB accelerator. The accelerator process a stream of input pixels from the image, exploiting the space locality and implementing several hardware structures that provide a sliding window access pattern. The accelerator accesses to each pixel of every image just one time and is able to keep all the patches necessary for the FAST, Non-maximal suppression, Gauss filtering, rBRIEF descriptor generation and moment computation synchronized.

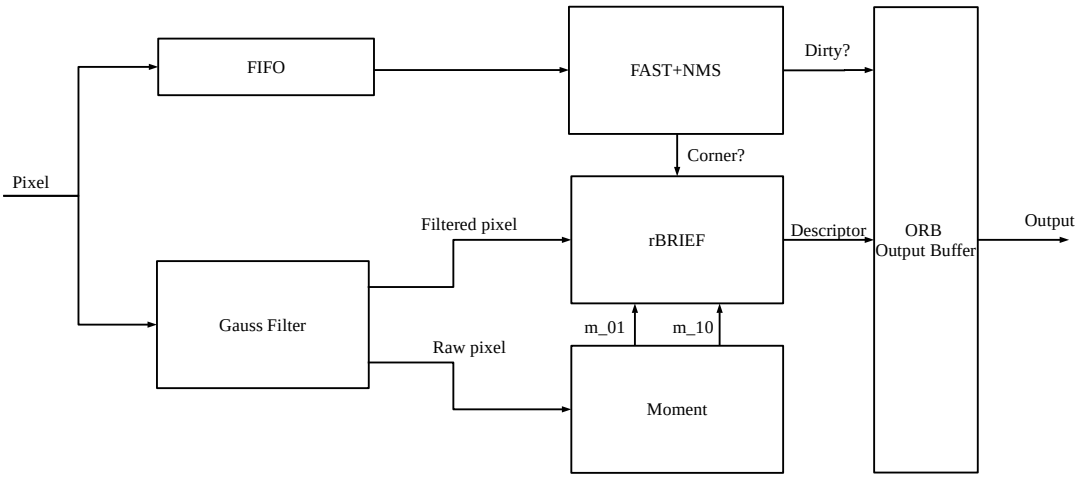

FIGURE 5.1: The architecture of the ORB accelerator.

The accelerator has two main design parameters:

- Maximun width: Size of sliding windows and other buffers that store parts of the input image temporarily on-chip. It must be multiple of 30 facilitate the management of the dynamic threshold grid.

- rBRIEF replication parameter: Number of replications of the rBRIEF sliding windows. More replications translates to lower rBRIEF latency. See section 5.1.5 for more details.

The following sections describe the implementation and behaviour of the accelerator.

5.1.1 Basic structures

This section describes some of the basic structures used in the design that is useful to clarify before anything else. The most important fundamental structures used in the design are the FIFO, the Delay FIFO and the sliding window.

The FIFO structure is well-know but we put here in order to distinguish it from the Delay FIFO. While the FIFO allows to organize a buffer so that the first element that arrives is the first to be processed, the Delay FIFO has the same behavior with the difference that this happens when the structure is full. In this way, the elements leave the structure a number of fixed cycles after their entry, that is, delayed. We implement the Delay FIFO as a circular buffer with two ports, one for read and the other for writing to the structure. The structure holds a pointer indicating the position to be read and written in the next enabled cycle. The Delay FIFO has the same behaviour as a chain of shift registers but with the benefit of only reading/writing one position, thus, it is more energy efficient.

The other fundamental structure to understand the design is the sliding window, a common structure used to support 2D convolutions particularly common in image processing hardware. This structure is used as a temporary storage and synchronization mechanism and its use has a great impact on the design. Figure 5.2 illustrates a sliding window of size 7×7, similar to the ones used by the FAST and Gauss units (described later).

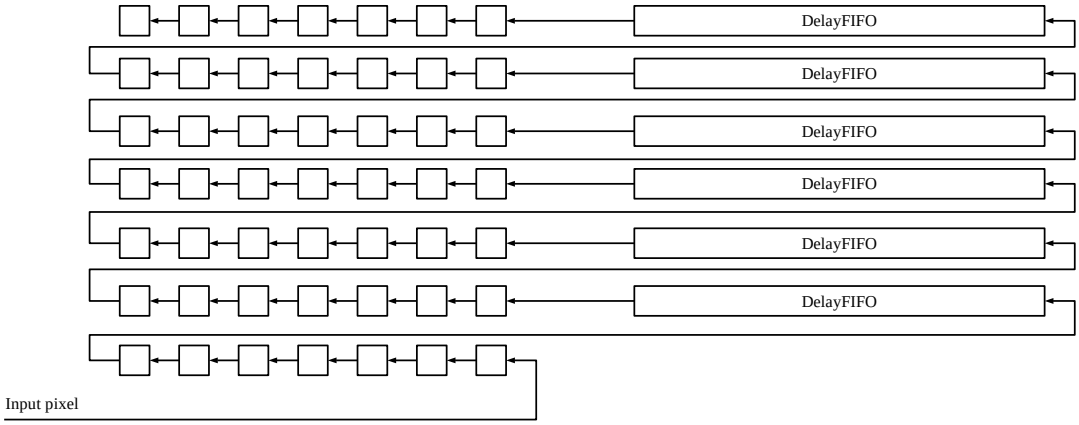

FIGURE 5.2: Gauss filter sliding window structure.

The pixels from an image are feed into the structure one per cycle in raster scan. The structure maps this pattern, therefore, each cycle the pixels will be shifted one position (eventually the pixels will be dropped at the end of the chain), providing access to a moving widow of pixels that iteratively covers every position of the image. The sliding window will take a fixed number of cycles until it can be used

because it is necessary to fill the entire structure with the whole local neighborhood. We usually refer to this as: warm-up time.

Another important design elements is the valid/ready (val/rdy) latency-insensitive micropotocol that allow us to easily perform flow control between the different units. Each cycle the producer determines if it has a new message to send to the consumer. If it is the case, the producer puts the message into the appropriate port and then sets the valid signal high. Concurrently, the consumer determines if it is able to process a new message from the produce. If is is the case, it sets the ready signal high. At the end of the cycle, the producer and the consumer can independently AND the valid and ready signals together. If the results is true the transaction is considered to be performed and both entities can update their internal state. If this condition fails, both entities must try again in the next cycle.

In our design, all units are capable of processing 1 pixel per cycle with the exception of the rBRIEF unit. In addition, it is necessary to perform a synchronization between the several sliding windows that the accelerator maintains. In these cases, the use of the described microprotol to operate the sliding windows, facilitates flow control.

5.1.2 Accelerator Interface

Figure 5.3 illustrates the black box interface that the accelerator exposes with the following input ports:

- Val/rdy interface support.
- Input: Input pixel of the image processed in raster scan.
- Width: Width of the image or tile to be processed.
- IniThr, MinThr: Thresholds values needed for FAST.

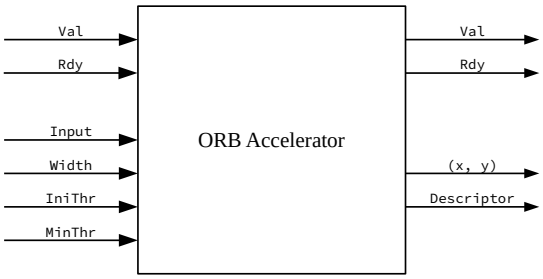

FIGURE 5.3: ORB accelerator black box interface.

The output of the accelerator is a stream of ORB descriptors and the corresponding local coordinates of the feature detected.

Another important aspect regarding the input interface is related with the preconditions that the input data must satisfy. It was previously said that the image data should be feed in raster scan which is a commonly used image storage pattern. In addition, the accelerator is implemented assuming that a padding at the edges will be added outside the accelerator. In our tests, we extend the image boundaries

with the OpenCV [27] *BORDER_REFLECT_101* type because is the option used by ORB-SLAM. The total size of the border is equal to half the width of the largest sliding window.

Finally, although the accelerator has limited resources and the FIFOs have fixed size (Maximum width parameter), it is possible to process images with a width larger than the maximum width of the sliding windows through the use of tiling technique. Tiling an image segments it into a number of smaller rectangular areas called tiles. Our accelerator is able to process an image of any size as long as it is segmented into tiles (see Figure 5.4). This is done transparently to the accelerator, since it is not relevant if the input data is part of a single image or a part of a larger one.

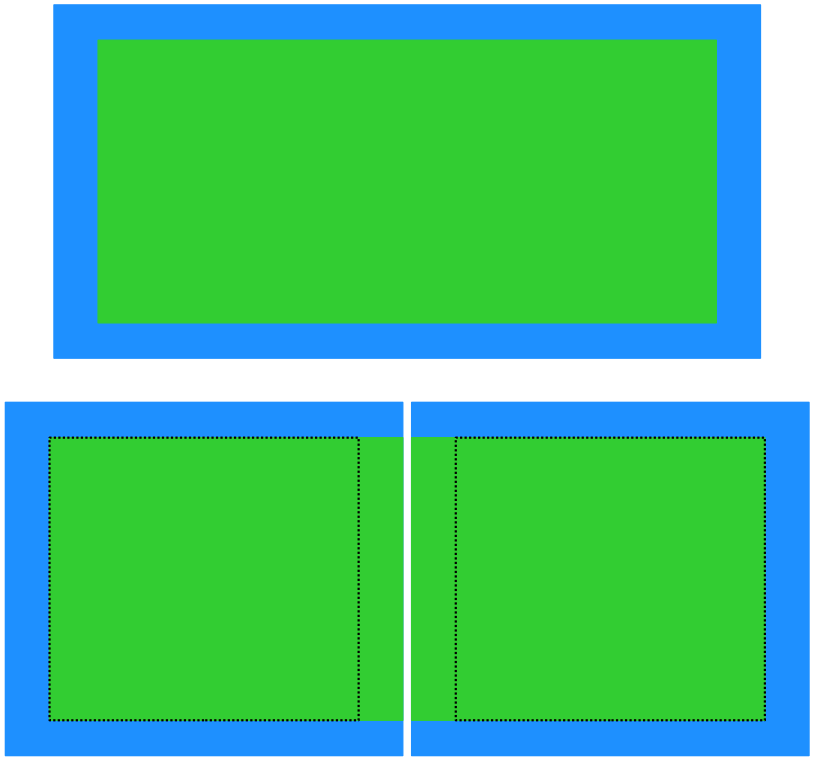

FIGURE 5.4: Tiling of a image (above) into two tiles (below). Green area corresponds to the original image. Blue area corresponds to the border added to the original image. The dashed line delimits the effective original image in the tilled version. Tilling adds new borders increasing the overhead.

5.1.3 FAST+NMS Unit

Figure 5.5 illustrates the architecture of the FAST+NMS Unit. The unit comprises two sliding windows. The module try to detect in parallel corners with the two thresholds passed as a parameter at runtime (IniThr and MinThr). The module detects corners with the MinThr until at least one corner of IniThr is found (if any) inside each 30×30 tile in which the image is divided. While no corners or MinThr corners are detected, the accelerator works under normal operation generating ORB descriptors for such corners. Those corners are stored in the ORB Store Buffer until the dynamic threshold tile is processed. If at least one corner with the high threshold

is found, the descriptors generated in that tile with the MinThr are discarded. The Dynamic Threshold module keeps track of the status of each dynamic threshold tile and is responsible to set the dirty bit high when a IniThr corner is detected.

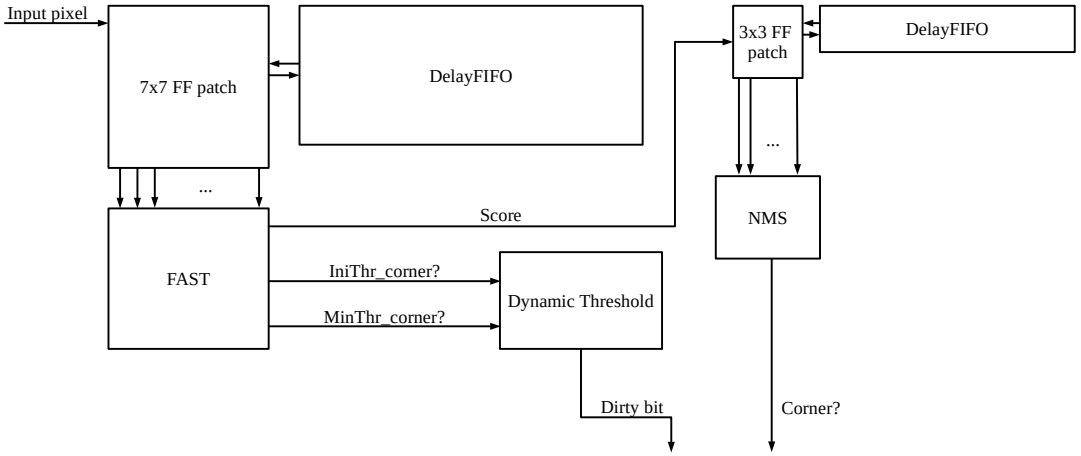

FIGURE 5.5: Architecture of the FAST unit.

Regarding non-maximal suppresion, a 3×3 sliding window through the scores of the pixels of the image is used to filter corners. The NMS module compares each cycle, the 8 corner scores around the center pixel to determine if it has the highest score, indicating the outcome setting the Corner? signal accordingly.

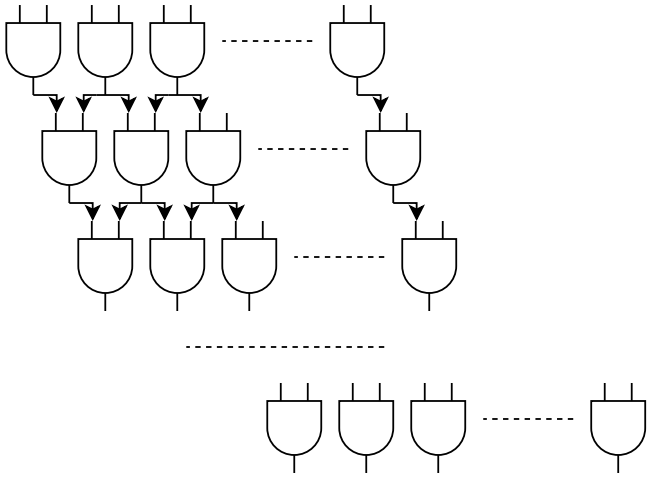

FIGURE 5.6: AND tree scheme that computes the segment test.

In order to perform the segment test, each pixel intensity of the Bresenham circumference is compared with the central pixel (taking into account the corresponding threshold) obtaining a 16 bit string. Figure 5.6 depicts the implementation developed to compute the segment test based on this string. Basically, the AND tree try to detect patterns with (9 consecutive ones) like 111111111XXXXXX, X111111111XXXXX, etc.

5.1.4 Gauss Unit

Figure 5.7 shows the Gauss Filter unit architecture.

We use the same Gaussian distribution parameters used in ORB-SLAM: Kernel size of 7×7 and $\sigma = 2$:

$$K = \begin{pmatrix} 0.005084 & 0.009377 & 0.013539 & 0.015302 & 0.013539 & 0.009377 & 0.005084 \\ 0.009377 & 0.017296 & 0.024972 & 0.028224 & 0.024972 & 0.017296 & 0.009377 \\ 0.013539 & 0.024972 & 0.036054 & 0.040749 & 0.036054 & 0.024972 & 0.013539 \\ 0.015302 & 0.028224 & 0.040749 & 0.046056 & 0.040749 & 0.028224 & 0.015302 \\ 0.013539 & 0.024972 & 0.036054 & 0.040749 & 0.036054 & 0.024972 & 0.013539 \\ 0.009377 & 0.017296 & 0.024972 & 0.028224 & 0.024972 & 0.017296 & 0.009377 \\ 0.005084 & 0.009377 & 0.013539 & 0.015302 & 0.013539 & 0.009377 & 0.005084 \end{pmatrix}$$

We multiply each element of the matrix by 16 and then we convert it to fixed point Q8.4. The module accesses each element of the sliding window and multiplies it by the corresponding kernel value. Subsequently a tree of adders is used to obtain the result of the convolution. This result is finally rounded and converted to integer back again.

It is important to understand that the output of this module is both the raw pixel intensity and the filtered pixel intensity once Gaussian filtering has been performed. The pixels that leave the window are redirected to the raw pixel exit port ensuring synchronization.

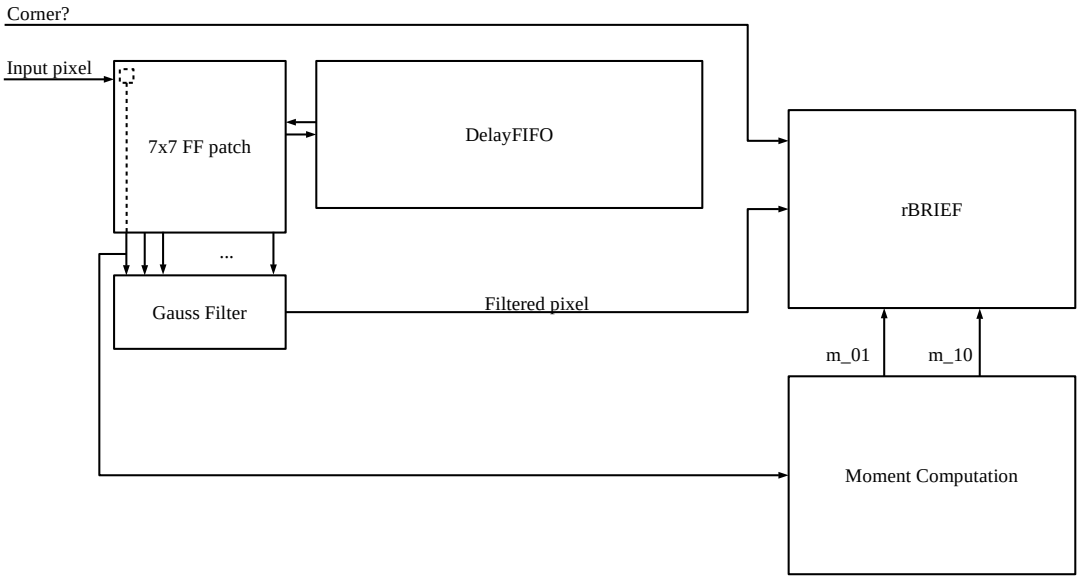

FIGURE 5.7: Architecture of the rBRIEF unit.

5.1.5 rBRIEF Unit

Figure 5.8 illustrates the rBRIEF hardware architecture.

The filtered pixels coming from the Gauss Filter unit are stored in a sliding window with a patch size of 37×37. This sliding window has some notable differences:

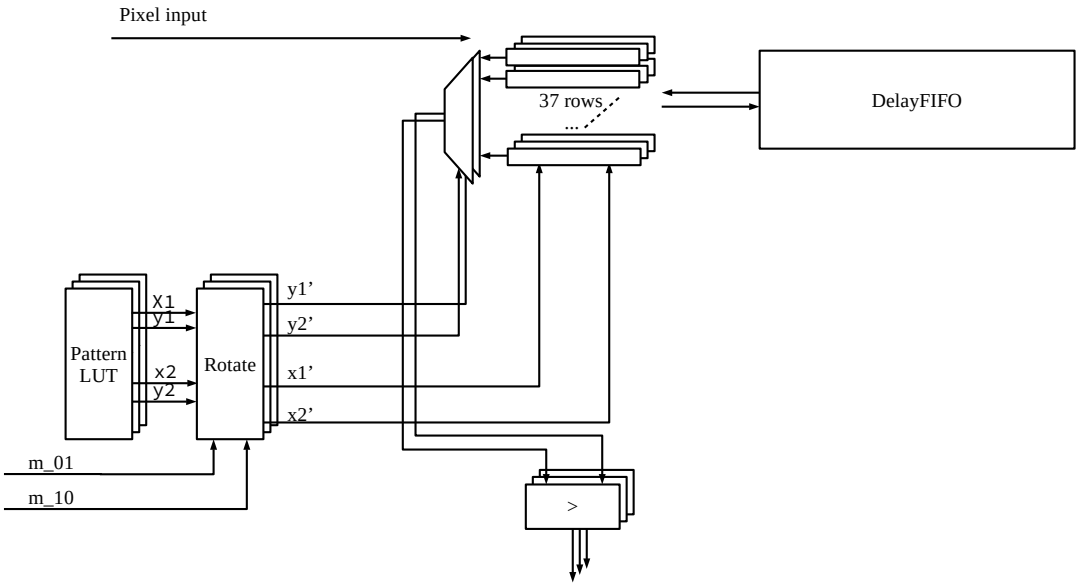

FIGURE 5.8: Architecture of the rBRIEF unit.

- The window is not implemented using flip flop chains in contrast to the FAST or Gauss Filter case. Instead, we employ a DelayFIFO structure to store each row of the window.
- The window is replicated a configurable number of times.

When a corner is detected by the FAST+NMS unit, the window contains the filtered pixels centered on the detected feature. However, we need to rotate the BRIEF pattern according to the angle θ of the centroid. When the operation starts, the unit enters a state in which the $sin(\theta)$ and $cos(\theta)$ are computed using the formula 2.4. After calculating sine and the cosine, it is possible to rotate the BRIEF coordinate pairs. Note that the BRIEF pattern is fixed and can be stored in a lookup-table (LUT). In our implementation this LUT is partitioned according to the level of replication in a way that each window and comparator can access the pairs and generate the BRIEF bit in-order. Each cycle the index with which this table is accessed to obtain a new BRIEF pair is increased.

Since 256 comparisons are necessary to generate an ORB descriptor, the unit blocks the rest of the pipeline forced to wait until completion. However, the latency of this unit can be reduced varying the replication factor. This factor controls the number of duplicated windows that are keep together. In this way it is possible to access the window more times per cycle reducing the time needed to generate the ORB descriptor.

5.1.6 ORB Output Buffer

The output buffer is a storage for the generated ORB descriptors that remain there until their validity is assured. There is a buffer for each 30×30 dynamic threshold tile which mean that we need $\frac{MaxWidth}{30}$ individual buffers (to keep track which descriptors belong to which tiles).

5.1.7 Integration

The key design decision behind the proposed accelerator is that it must be ensured that all the windows are synchronized and in harmony so that when a feature is detected, all the data is available.

To achieve this we must take care of the synchronization between FAST+NMS unit and the Gauss, Moment and rBRIEF units. Figure 5.9 illustrates the different positions of the four sliding windows maintained by the accelerator. The green region represents the image itself while the rest of regions represents the different border needed by the sliding windows. Each window are fed with pixel data with different offsets. Red, blue, orange areas contains the pixels needed for the Gaussian Filter, Moment and rBRIEF, and FAST unit respectively. The control logic must ensure that these regions are detected to feed each unit with the appropriate pixels.

In addition, Moment and rBRIEF windows are fed by the Gauss Filter Unit while NMS is fed by the FAST Unit. This means that at the end of the day, just two units are fed directly from the image pixels: FAST and Gauss Filter units. Since the Gaussian filter window is always in a more further position, we decided to put a DelayFIFO between the input and the FAST unit with a length equal to the offset between the FAST and Gauss input pixel streams. This way all windows can be synchronized.

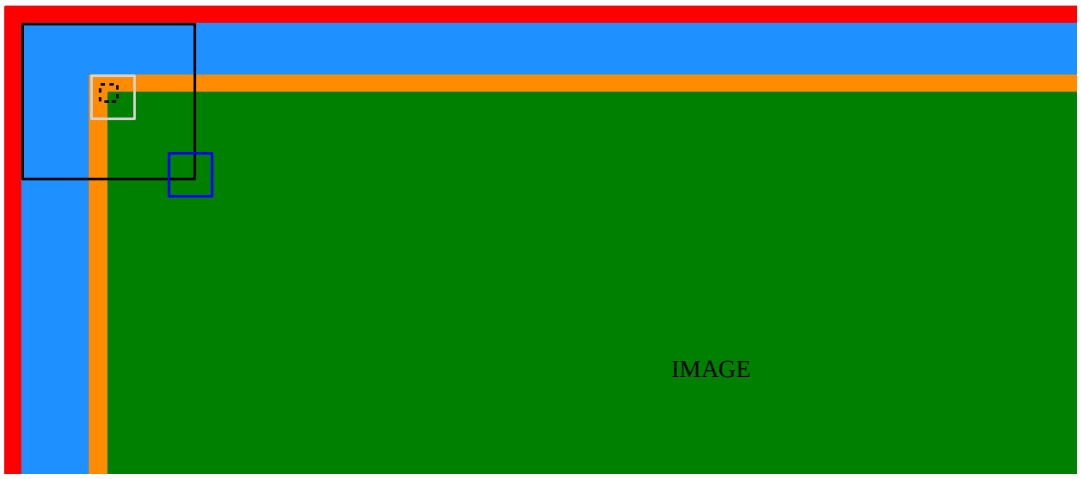

FIGURE 5.9: Example of the relative positions of each sliding window of the accelerator.

5.2 Design space exploration

Figure 5.10 shows the evolution of the average number of cycles required to process the 00 sequence of the odometry benchmark for different values of the maximum width parameter or alternatively the sliding windows length.

Based on these results, we have defined the three different profiles to test the accelerator. Table 5.1 shows the accelerator profile details.

Additionally, we perform various test in order to select an appropieta value for the rBRIEF replication factor. We choose a replication factor of 8, considering the

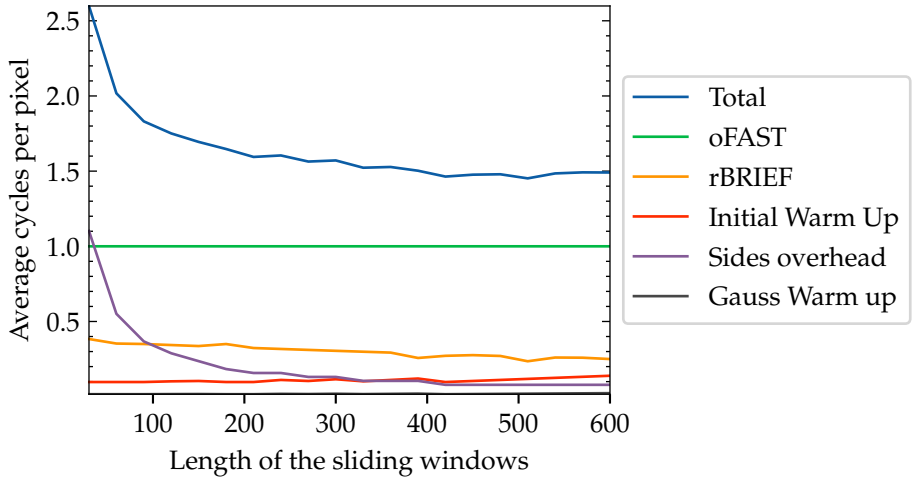

FIGURE 5.10: Average Execution time in different platforms.

Accelerator name	FIFO length (px)
Acc-Small	90
Acc-Medium	210
Acc-Larg	420

TABLE 5.1: Accelerator configurations tested.

compromise between performance improvement and increase in energy and area of the possible choices.

5.3 Results

In this section we discuss the results obtained after testing the accelerator implementation. We compare the results with the CPU implementation following the methodology described in chapter 3.

5.3.1 Performance

Figure 5.11 reveals a reduction in the execution time required to process the benchmark.

The three versions of the ORB accelerator significantly reduce the execution time. Graph 5.12 shows the speedup with respect to the CPU, so the three versions of the accelerators can be compared against each other. As expected, ACC-large has a better speedup due to the reduction of overhead incurred when tiling. The reduction of cycles wasted when the window changes from one row to another also has to do with this performance improvement. The speedups are 5.77x, 7.31x and 7.96x approximately for ACC-Small, ACC-Medium and ACC-Large respectively.

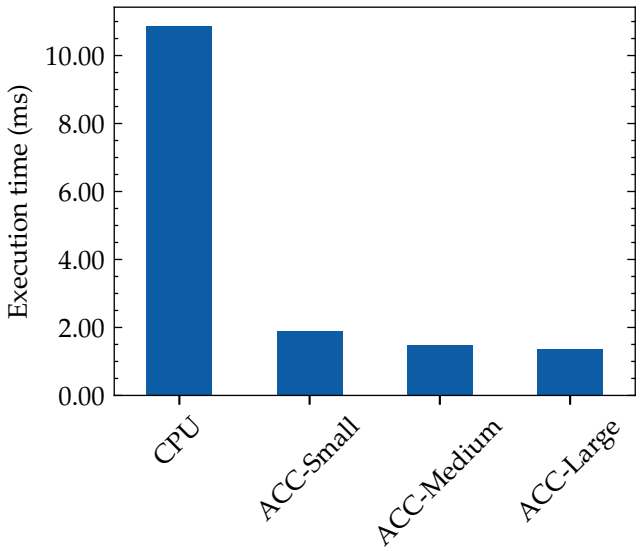

FIGURE 5.11: Average Execution time in different platforms.

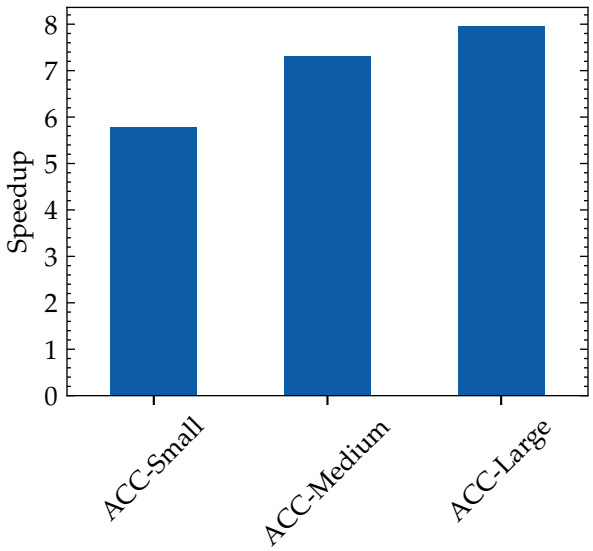

FIGURE 5.12: Speedup obtained vs CPU platform.

5.3.2 Energy

In figure 5.13, we can observe a very significant reduction in energy consumption per frame.

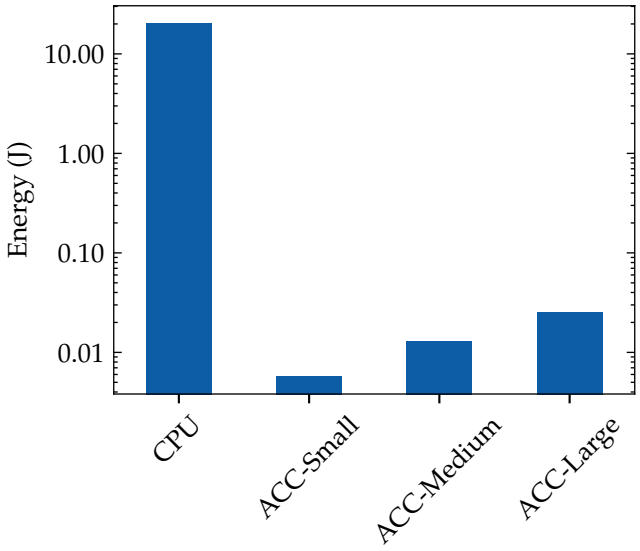

FIGURE 5.13: Energy dissipated per frame.

The results are expected, energy consumption is positively correlated with the increase in the size of the storage used mainly due to the leakage power dissipated by the storage elements. The normalized energy dissipation per frame is shown in the graph 5.14.

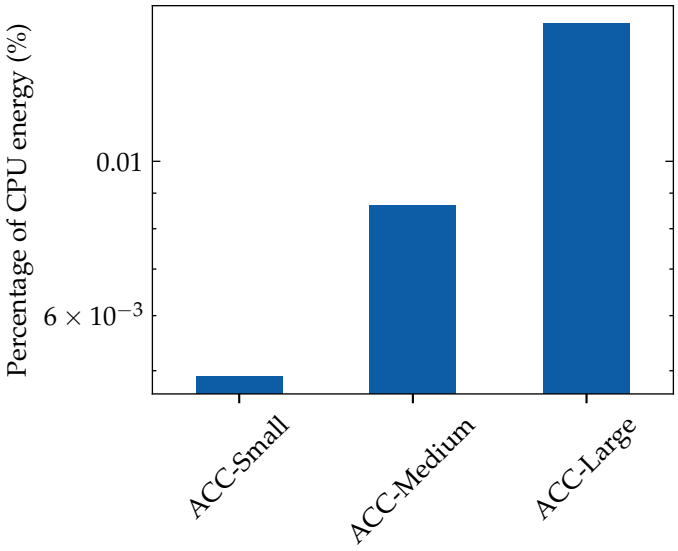

FIGURE 5.14: Normalized energy consumption per frame.

ACC-Medium achieves more than 2000x reduction in energy consumption.

5.3.3 Power dissipation

Power consumption of each platform is show in the figure 5.15. This is an alternative measure to the energy consumption shown in the previous section.

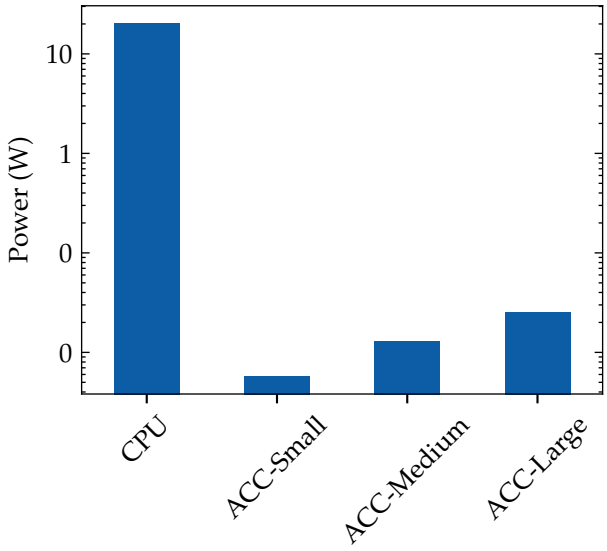

FIGURE 5.15: Power dissipation of each platform.

5.3.4 Area

Following the area estimation approach described in chapter 3, we obtain a value of $2.6mm^2$ of total area needed to implement the ACC-medium ORB accelerator.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Self-driving Car Systems can revolutionize our lives. The technological improvements driven by the development of AI, seems to suggest that in a few years there is going to be a great improvement in this area.

Although progress has been made, SDC systems presents many challenges of different types. Powerful platforms that are capable of satisfying the computational need and at the same time being energy efficient will be necessary. This can be a great research opportunity in the context of computer architecture as it has been verified in this project.

We have been able to develop an accelerator capable of improving performance and reducing the energy consumption of one of the state-of-the-art SLAM systems achieving our main goal. Furthermore, the proposed accelerator has room for improvement, for example through some of the ideas described in the next section.

6.2 Future Work

Some improvements could be made to optimize performance and to extend the functionality of the accelerator. The most promising ones are listed below:

- rBRIEF unit is one of the bottlenecks of the accelerator. The way in which the data is organized does not map well whith the rotation accessing pattern needed. It would be interesting to investigate some data organization that allow better access patterns.
- Investigate a dynamic schedule for ORB generation. The pixel intensity comparison between BRIEF pairs are independent from each other. Solutions such as the one presented, waste a great amount of bandwidth and do not exploit the possible data parallelism.
- Develop an architecture with replicated ORB accelerators. This could provide huge speedups taking into account that the data parallelism is high within a frame and within an image.
- Design an extension to the accelerator to provide the possibility to detect multi-scale features. It would be necessary to implement some hardware architecture

capable of generating the scale image pyramid. This will increase the exploited parallelism reducing the latency of the feature extraction.

Bibliography

[1] *Road traffic injuries*, 2019. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[2] *Nthsa fatalities report*, 2008. [Online]. Available: https://crashstats.nhtsa.dot.gov/api/public/viewpublication/811059.

[3] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving", *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 751–766, Mar. 2018. DOI: 10.1145/3296957.3173191. [Online]. Available: https://doi.org/10.1145%2F3296957.3173191.

[4] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: A survey", *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[5] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles", *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[6] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3d-ndt", *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827, 2007.

[7] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras", *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017. DOI: 10.1109/tro.2017.2705103. [Online]. Available: https://doi.org/10.1109%2Ftro.2017.2705103.

[8] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam", *Journal of intelligent & robotic systems*, vol. 61, no. 1-4, pp. 287–299, 2011.

[9] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse", *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[10] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016", *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 16, 2017.

[11] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time", in *2011 international conference on computer vision*, IEEE, 2011, pp. 2320–2327.

[12] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system", *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf.", Citeseer.

[14] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012, pp. 3354–3361.

[15] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment — a modern synthesis", in *International workshop on vision algorithms*, Springer, 1999, pp. 298–372.

[16] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking.", in *ICCV*, Citeseer, vol. 2, 2005, pp. 1508–1515.

[17] ——, "Machine learning for high-speed corner detection", in *European conference on computer vision*, Springer, 2006, pp. 430–443.

[18] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features", in *European conference on computer vision*, Springer, 2010, pp. 778–792.

[19] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector.", in *Alvey vision conference*, Citeseer, vol. 15, 1988, pp. 10–5244.

[20] P. L. Rosin, "Measuring corner properties", *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.

[21] D. Lockhart, G. Zibrat, and C. Batten, "Pymtl: A unified framework for vertically integrated computer architecture research", in *47th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2014, pp. 280–292. DOI: 10.1109/MICRO.2014.50.

[22] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with papi", in *2012 41st international conference on parallel processing workshops*, IEEE, 2012, pp. 262–268.

[23] C. Wolf, *Yosys open synthesis suite*, http://www.clifford.at/yosys/.

[24] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, *et al.*, "Freepdk: An open-source variation-aware design kit", in *2007 IEEE international conference on Microelectronic Systems Education (MSE'07)*, IEEE, 2007, pp. 173–174.

[25] *Synopsys*, https://www.synopsys.com/, Accesed: 2019-10-16.

[26] *Orb accelerator source code*. [Online]. Available: https://github.com/taranco/ORB-Accelerator.

[27] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.