UNIVERSITÀ DEGLI STUDI DI SALERNO

Dottorato in Informatica e Ingegneria dell'Informazione
Curriculum Informatica

Coordinatore:
Prof. Alfredo De Santis
XV Nuovo Ciclo

About the Development of Visual Search Algorithms and their Hardware Implementations

Relatori:
Ch.mo Prof. Giancarlo Raiconi
Ph.D. Mario Vigliar

Candidato:
Luca Puglia
Matr. 8888100007

Anno Accademico 2016/2017

"...Inside my heart is breaking
My make-up may be flaking
But my smile still stays on..."
— The Show Must Go On, *Queen*

To anyone who has ever believed in me.

Contents

Introduction

The main goal of my work is to exploit the benefits of a hardware implementation of a 3D visual search pipeline. The term visual search refers to the task of searching objects in the environment starting from the real world representation. Object recognition today is mainly based on scene descriptors, an unique description for special spots in the data structure. This task has been implemented traditionally for years using just plain images: an image descriptor is a feature vector used to describe a position in the images. Matching descriptors present in different viewing of the same scene should allows the same spot to be found from different angles, therefore a good descriptor should be robust with respect to changes in: scene luminosity, camera affine transformations (rotation, scale and translation), camera noise and object affine transformations. Clearly, by using 2D images it is not possible to be robust with respect to the change in the projective space, e.g. if the object is rotated with respect to the up camera axes its 2D projection will dramatically change. For this reason, alongside 2D descriptors, many techniques have been proposed to solve the projective transformation problem using 3D descriptors that allow to map the shape of the objects and consequently the surface real appearance. This category of descriptors relies on 3D Point Cloud and Disparity Map to build a reliable feature vector which is invariant to the projective transformation. More

sophisticated techniques are needed to obtain the 3D representation of the scene and, if necessary, the texture of the 3D model and obviously these techniques are also more computationally intensive than the simple image capture. The field of 3D model acquisition is very broad, it is possible to distinguish between two main categories: active and passive methods. In the active methods category we can find special devices able to obtain 3D information projecting special light and. Generally an infrared projector is coupled with a camera: while the infrared light projects a well known and fixed pattern, the camera will receive the information of the patterns reflection on a certain surface and the distortion in the pattern will give the precise depth of every point in the scene. These kind of sensors are of course expensive and not very efficient from the power consumption point of view, since a lot of power is wasted projecting light and the use of lasers also imposes eye safety rules on frame rate and transmissed power. Another way to obtain 3D models is to use passive stereo vision techniques, where two (or more) cameras are required which only acquire the scene appearance. Using the two (or more) images as input for a stereo matching algorithm it is possible to reconstruct the 3D world. Since more computational resources will be needed for this task, hardware acceleration can give an impressive performance boost over pure software approach.

In this work I will explore the principal steps of a visual search pipeline composed by a 3D vision and a 3D description system. Both systems will take advantage of a parallelized architecture prototyped in RTL and implemented on an FPGA platform. This is a huge research field and in this work I will try to explain the reason for all the choices I made for my implementation, e.g. chosen algorithms, applied heuristics to accelerate the performance and selected device. In the first chapter we explain the Visual Search issues, showing the main components required by a Visual Search pipeline. Then I show the implemented architecture for a stereo vision system based on a Bio-informatics inspired approach, where the final system can process up to 30fps at 1024×768 pixels. After that a clever method for boosting the performance of 3D descriptor is presented and as last chapter the final architecture for the SHOT descriptor on FPGA will

be presented. A more complete description of the work can be found in [1, 2, 3, 4, 5, 6]

Chapter 1

Visual Search

Visual Search is a complex task and over the years many approaches have been proposed to solve it. Since the real world can be modelled with different kinds of data, visual search methods may differ a lot with respect to the chosen technique. Generally speaking, a representation of the real world can be acquired in multiple ways. Two kinds of sensors are used in this field: camera sensors and range sensors. Both sensors have advantages over the other. the images produced with a camera are relatively easy to manipulate and visualize but a slanted surface in the scene will result in major drawback in the visual search algorithm performance (in Fig. 1.1 is shown an instance of the problem). By using a range sensor it is possible to obtain a 3D representation of the world, but unfortunately this representation is less compact than an image but more information about the scene structure is carried by the data (Fig. 1.2); furthermore, changes in the position of the sensors potentially bring projective changes in the object pose, and a change in the object projection onto the image plane will dramatically change the appearance of some object, using a 3D representation no problems arise when the object is rotated since the whole structure is defined.

In order to have better results in the Visual Search pipeline, we decided to

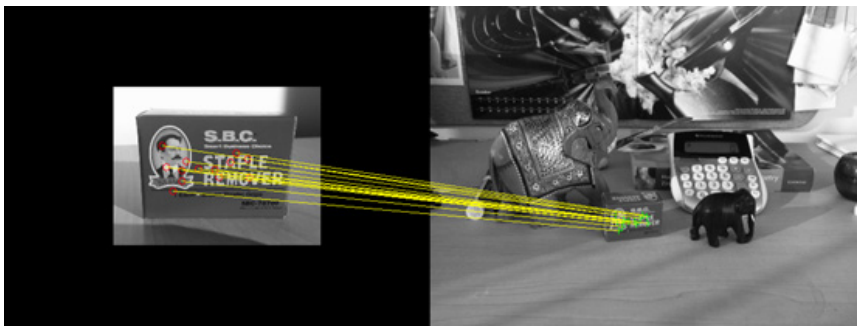

Figure 1.1: Example of visual search problem using images; starting from a template (left) some keypoints are extracted, using a visual search algorithm is possible to match these points in a cluttered scene (right). This kind of approach only works well when the template pose is close to the one present in the scene.

utilize the more robust representation brought by range sensors. This kind of data are more complex to acquire. Usually each device uses its own output format and a conversion phase is required to use those data. These systems can be distinguished in active and passive. The former category uses a light sensor paired with a light emitter, the emitter projects a pattern over the scene, while the sensor registers changes in the known pattern; By using this strategy it is possible to measure the distances of the scene positions. Some drawbacks occur when using these sensors in outdoor enviroment and direct sunlight: in fact, sun rays can interfere with the light emitter pattern. Some of these systems are available off-the-shelf (e.g. Kinect^{TM}, Asus Xtion Pro Live, ...). The latter category, instead, is based on just camera sensors. By using images, in fact, it is possible to obtain information about the object position, the system generally results more power efficient than an active approach since no light emitters are used, but of course more computational power is required because a complex algorithm must run.

For this work I created range data starting from a camera pair (so the passive approach was chosen). Starting from a stereoscopic system it is

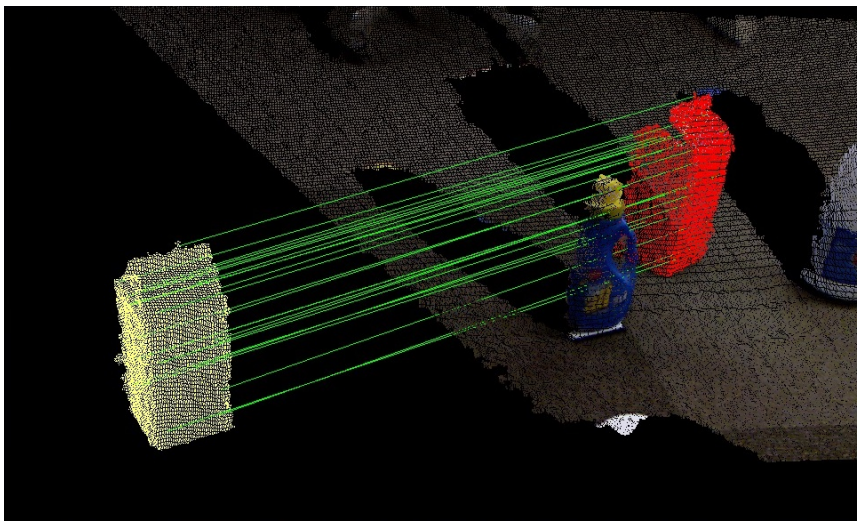

Figure 1.2: Example of 3D visual search, the template in yellow (on the left) is matched against the 3D representation of the scene (on the right).

possible to obtain a 3D representation of the scene, but, as mentioned previously, more computational power is required.

1.1 Active range sensors

In the past few years there has been a growing interest in processing 3D data. Computer vision tasks such as 3D keypoint detection and description, surface matching and segmentation, 3D object recognition and categorization have become of major relevance. The relevance of such applications has been fostered by the availability, in the consumer market, of new low-cost Active range sensors (also called RGB-D cameras), which can simultaneously capture RGB and range images at a high frame rate. Such devices are either based on structured light (e.g. Microsoft Kinect, Asus Xtion) or Time-of-Flight (TOF) technology (e.g., Kinect II) (Fig. 1.3), and belong to

Figure 1.3: Example of off-the-shelf RGB-D sensors.

the class of active acquisition methods.

Independently from the specific technology being used, each sensor acquires 3D data in the form of range images, a type of 3D representation that stores depth measurements obtained from a specific point in 3D space (i.e., sensor viewpoint) — for this reason, it is sometimes referred to as 2.5D data. Such representation is *organized*, in the sense that each depth value is logically stored in a 2D array, so that spatially correlated points can be accessed by looking at nearby positions on such grid. Conversely, data are *unorganized* when 3D representation is simply stored in 3D coordinates in an unordered list. These kinds of representation (unorganized and organized) are called point clouds.

1.2 Passive range sensors

Stereo Matching is the problem to solve to obtain a 3D representation of the world. In order to triangulate every point in the scene a special pair of sensors is required. This problem usually requires two cameras as input sources. It can be defined as the process of finding couples of pixels in the two images representing the same points in the real world. It is possible to distinguish two categories of problems, namely: sparse stereo matching and dense stereo matching. The first category is generally easier and faster to solve since only some specific (key)points are matched between the images

Figure 1.4: Sparse stereo matching input (first two images) and output (last image). The horizontal lines represent the few matched point on both images.

(Fig. 1.4). The second category can be thought as a generalization of the first one: instead of finding just some matches in the two images, a match is find for every point (pixel) (Fig. 1.5); the final result of this process is called a disparity map.

Figure 1.5: Dense stereo matching problem input (first two images) and output (last two images).

Since [7] all the efforts in the Stereo Vision community have been focused on finding the best algorithm to match correct pixel pairs. Many of these algorithms are discussed in [8], where a taxonomy is defined that distinguishes between global and local methods. According to [8] Stereo Vision algorithms are composed of five different phases:

- Pre-processing;
- Matching cost computation;

- Cost aggregation;
- Disparity computation;
- Disparity refinement.

The search for the best method has caused an explosion of creativity regarding the matching technique. Disparity map quality, system performance, frame rate, image resolution, power consumption and platform cost are just few of the objectives that can be achieved by a stereo vision system. Obviously, many of these targets are in contrast, e.g. it is not possible to have good disparity map quality with high performance on a cheap platform, therefore a trade-off has to be reached. The complexity of the algorithm impacts on the disparity map quality, in fact, it is tightly coupled to the number of comparisons to be performed: for every pixel in the first (reference) image a matching pixel must be found in the second (target) image.

1.2.1 Pre-processing

This is a common phase for every image processing algorithm, in the stereo-vision context is used to remove image distortion and noise. The following methods are encompassed by the pre-processing:

- Laplacian of Gaussian (LoG) filtering [9];
- Histogram equalization;
- Neighborhood average value subtraction [10];
- Bilateral filter [11].

1.2.2 Matching Cost

Since a comparison between pixels is required a matching cost function must be defined, the easiest technique is to subtract pixel channel values: let I_L be the reference image (sensor on the left), I_R be the target image (sensor on the right), we define:

- Absolute intensity difference (AD): $|I_L(x_1, y_1) - I_R(x_2, y_2)|$
- Squared intensity difference (SD): $(I_L(x_1, y_1) - I_R(x_2, y_2))^2$

where $I(x, y)$ represents the pixel of position x, y.

1.2.3 Disparity Computation

Once a cost function is defined it is possible to compute the disparity between pixels. The plot in Fig. 1.6 is obtained by comparing pixels on the left image with every single pixels in the homologous right line.

If the cost function is good enough, the function minima will represent the correct match and consequently the disparity. This kind of strategy has been named "winner takes all" (WTA) (see Fig. 1.7). By using this kind of strategy it is hard to obtain good results since the cost function has to be robust enough to accounnt for every possible situation present in the images.

1.2.4 Cost Aggregation

In order to obtain a more robust algorithm it is possible to aggregate the cost functions of more than one pixel:

$$\sum_{(x,y)\in W} \Delta(I_L(x, y), I_R(x + \delta, y))$$

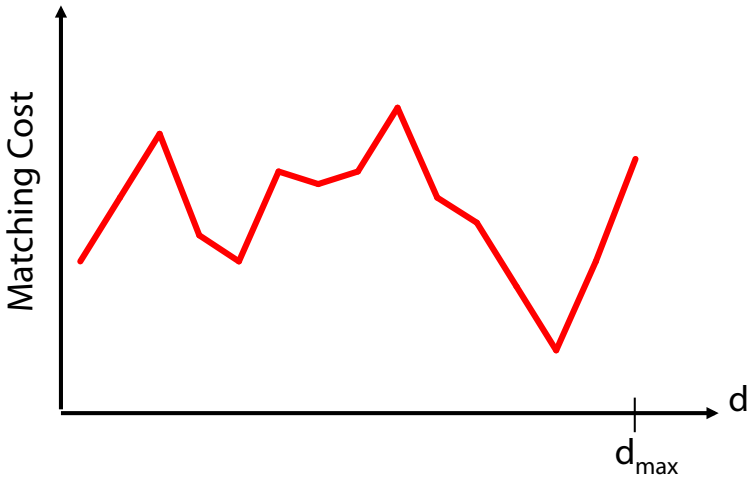

Figure 1.6: Disparity Function.

This function is called SAD (Sum of Absolute intensity Differences) or SSD (Sum of Squared intensity Differences) if the square value is used. In this case a window of pixels W is used to compute the cost of the matching. Fig. 1.8 shows how much the final result is improved.

1.2.5 Disparity Refinement

In order to enhance the final results of Cost Aggregation, a post-processing phase of disparity refinement is applied. Since the relative pixel distances are always integer numbers, the disparity function acts as if the world is discrete, but this is obviously not true for the real world. To improve the final result the disparity function could be interpolated to give a more smooth appearance to the disparity map (Fig. 1.9).

Furthermore, a cost aggregation strategy will decrease the quality of the disparity map around the corners; this happens because the different point of view between the cameras change dramatically the object projections

Figure 1.7: Winner takes all applied to Absolute intensity Difference.

Figure 1.8: Winner takes all applied to Sum of Absolute intensity Differences.

close to the edges, for this reason running an outlier detector/corrector increases the overall quality (Fig. 1.10).

1.2.6 Main Problems

Using window approaches (SAD, SSD) has some drawbacks. The main problems to address are the following:

- object edges;
- repetitive texture;
- uniform areas;
- small structure.

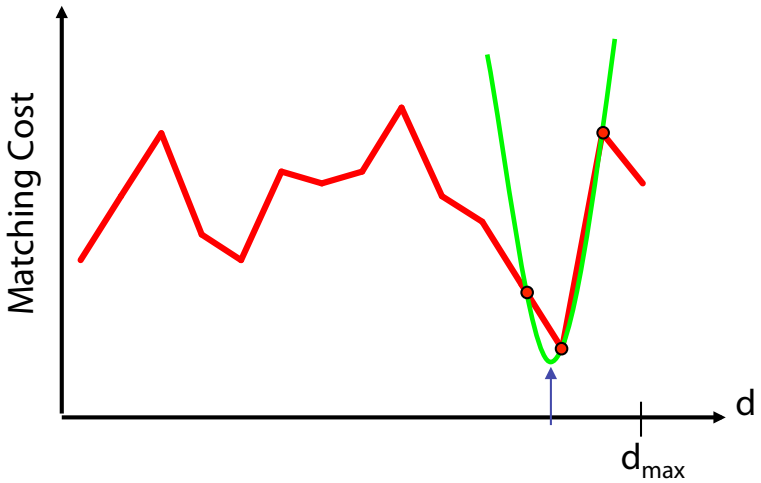

Figure 1.9: Interpolated Disparity Function.

In Fig. 1.11 some instances of these problems are shown.

In square 1 repetitive textures are present caused by multiple books with the same cover; in squares number 2 and 3 flat zones are highlighted; in square number 4 a slanted surface does not allow an easy matching; in 6 and 7 the edges of the objects affect the accuracy, finally in 5 the window is larger than the contained object.

1.2.7 Global Techniques

The methods shown in the previous sections are called local because they utilize only local information to match pixels, however other kinds of local methods use more complicated matching criteria based on correlation [12] and variable size windows [13, 14]. In order to obtain better results, a global method could be applied: these methods use whole image chunks during the optimization of the global function.

More formally, we define an objective function for each pixel in the reference

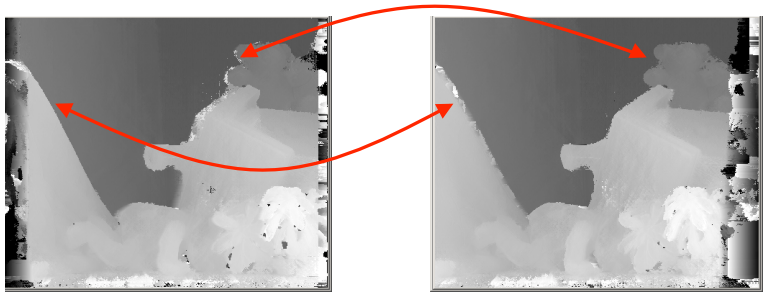

Figure 1.10: Outlier detection and correction.

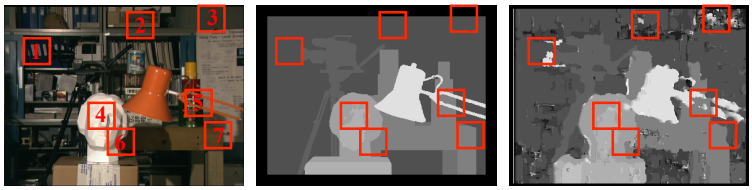

Figure 1.11: Windows which are responsible for major drawbacks in the result quality.

image; the functions depend on the parameter δ which represents the relative offset in the image space:

$$d(x,y) = \underset{\delta}{\operatorname{argmin}} \left(\Delta(I_L(x,y), I_R(x+\delta, y))\right)$$

The matching problem is now converted into an optimization problem. By minimizing the function for each pixel (x, y) we obtain a monochrome disparity map in which color channel are substituted by the solution $d(x, y)$. The Δ function can be defined as explained above using SAD and SSD functions:

$$d(x,y) = \underset{\delta}{\operatorname{argmin}} \left(\sum_{(x,y)\in W} \Delta(I_L(x,y), I_R(x+\delta, y)) \right)$$

This method enhances the quality of the algorithm in plain zones but deteriorate the quality near the object edges.

1.2.8 Accelerating Stereo Vision

Depending on the technique chosen, stereo matching can be a complex and time-consuming task, which makes CPU implementation very expensive (or unfeasible for global methods) in terms of cost of the chip and power consumption, and consequently a very large number of implementations for both GPU and FPGA devices have been developed in recent years. A remarkable GPU implementation can be found in [15] in which an Absolute Difference (AD)-Census algorithm is implemented; this algorithm has been ranked first in the Middlebury benchmark [16]. GPUs are chosen for the native support for parallelism, in fact many cores on a GPU can process multiple pixel pairs at the same time reducing dramatically the processing time. But the shortcoming of this approach is power consumption: generally a desktop GPU consumes many tens of watts making this solution not valid for embedded applications. In recent years many artificial vision applications on mobile robot have been developed, one of the principal tasks to handle in building a fleet of robots is the power management and the cost of the single element, without an affordable and power efficient solution further progress in this field will be constrained. The only way to obtain a stereo vision system with a low power consumption and acceptable quality on a cheap device is to use a dedicated architecture, that can be produced using devices like FPGA and ASIC. Since ASIC requires thousands (if not millions) devices to be produced, in this work I have prototyped an FPGA architecture. For a more accurate comparison between GPU and FPGA limits see [17].

1.3 Differences between Active and Passive sensors

The principal difference between active and passive range sensors is the resultant output: generally, an active sensor will output a point cloud, while a passive system will output a disparity map. Luckily both formats are interchangeable, therefore, starting from one is possible to obtain the other (with a modest numeric precision loss in the distances).

Since my visual search pipeline will be composed of a stereo vision system and a 3D description module, a conversion from disparity map to point cloud is requieed. The conversion is straightforward, each pixel becomes a point. The conversion formula is based on the camera parameters and the baseline size between the two cameras.

In 1.13 a classic structure for stereo vision system is shown: the vertical bold lines represent the image planes of the cameras, each camera has a focus f and a focus length f_z. In stereo systems, it is common to have homogeneous sources, thus a set of identical cameras with same f_z is used. A baseline B represents the distance between the two focuses, while the distance from a point in the real word P_w to the baseline is Z. The projection of the point P_w onto the camera sensors is marked respectively by $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$. By orienting the reference system of both cameras inward it is possible to simplify the equation to represent the disparity d of the point P_w as the sum of x coordinate of p points:

$$d = x_1 + x_2$$

This disparity is inversely proportional to the distance and can be used to triangulate the Z of the real world point. By using triangular similarity it is possible to obtain the following equation system:

$$\begin{cases} \frac{X_1}{Z} = \frac{x_1}{f_z} \\ \frac{X_2}{Z} = \frac{x_2}{f_z} \\ X_1 + X_2 = B \end{cases}$$

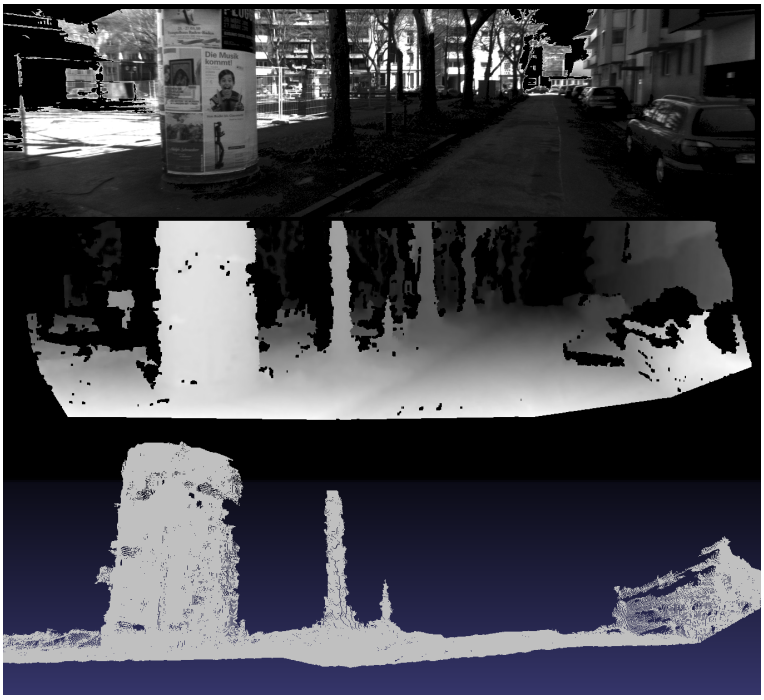

Figure 1.12: Disparity to cloud result.

This system can be solved for Z to obtain:

$$Z = \frac{Bf_z}{d}$$

by applying this rule to every disparity in the disparity map a point cloud is built.

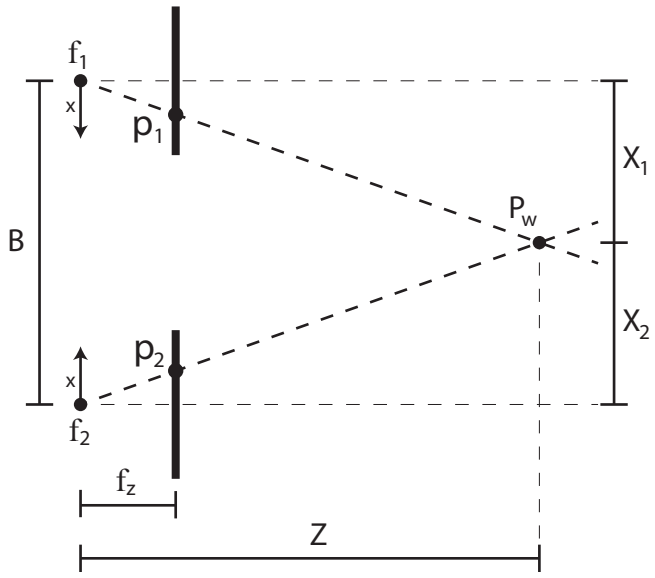

Figure 1.13: Scheme of a stereo system as it appears from the top.

1.4 3D Descriptors

The technological progress in the development of range sensors has led us to the ability in delivering high definition range map with an high frame rate. Each process that needs to be applied to this range map stream (e.g. key points extraction and key points description) has to satisfy a precise performance constraint in order to avoid bottlenecks in computation. To execute these algorithms on portable devices the extra constraint of low power usage is imposed.

The principal problem to address during 3D scene reconstruction is to find the correct rotation and translation matrix among many views obtained through passive or active range sensors (also called depth sensors).

These views are generally stored as 3D point clouds, lists of tuples containing

a 3D point (x, y, z) and an optional colour. Aligning point clouds is not easy and it is strictly related to the Simultaneous Localization and Mapping (SLAM) problem, but, once two disparity maps are correctly aligned, the relative positions of the sensors could be easily computed.

To solve the point cloud alignment problem is possible to use techniques based on the point matches among clouds. To make a correct match, it is necessary to compute a unique and descriptive representation for each point that is invariant to the sensor pose. For this reason, if this representation remains unambiguous among the views, it is said to be the "description" of the point. Specifically speaking, the description (or descriptor) of the point is stored as a multidimensional feature vector. The similarity between two descriptions is easily computed as the distance in the features space of the descriptors.

As keypoint extractions and descriptions are computationally intensive, these tasks are not suitable for embedded applications which require real time performance. Even modern Desktop CPUs struggle trying to compute a large number of descriptors of a 3D point cloud video. A possible solution is to use a GPU to parallelize the processing, but a battery dependent application could never use such a power consuming device, e.g. a drone trying to reconstruct the environment and at the same time to localize itself in the real word will never have the ability to align 3D cloud points obtained using a depth sensor. The only way to obtain a high performance system with a low power consumption and acceptable quality is to use a dedicated architecture that can be produced using devices like FPGA and ASIC. Since ASIC requires thousands (if not millions) devices to be produced, I chose to develop my application using an FPGA.

To avoid problems caused by rigid rotation of the objects, it is essential to define a reference system. The older techniques use the so called Local Reference Axis (LRA) which use the normal of the keypoint as a reference axis, while more recent approaches use a Local Reference Frame (LRF). A LRF is a new coordinate system that is used as a reference to encode the various geometries of the neighborhood. The LRF can be thought of

as a way to make the descriptors invariant to the sensor pose; this result is obtained by using the LRF as a way to encode the neighborhood of a keypoint.

Several 3D local feature descriptors have been designed to encode the information of a local surface. Among these approaches, many algorithms use histograms to represent different characteristics of the local surface. Specifically, they describe the local surface by accumulating geometric or topological measurements (e.g., point numbers) into histograms according to a specific domain (e.g., point coordinates, geometric attributes). These algorithms are categorized into 'spatial distribution histogram' and 'geometric attribute histogram' based descriptors. (Fig. 1.14).

1.4.1 Histogram based Descriptors

In this category we can find descriptors that use the point positions to build histograms.

Spin Image (SI) [19, 20] (Fig. 1.14.a) in SI the normal n to keypoint k is used as LRA, for each point q_i in the neighborhood an α and a β parameter are computed (Fig. 1.14.a). The discretization of these two distances is used to build a 2D Histogram, each point is accumulated in the histogram giving the final SI descriptor.

3D Shape Context (3DSC) [21] in 3DSC the normal n to keypoint k is used as the LRA. A spherical grid is superimposed on p, with the north pole of the grid being aligned with the normal n. The grid is then divided into several bins (Fig. 1.14.b). The divisions are logarithmically spaced along the radial dimension and linearly along the other two dimensions. The 3DSC descriptor is computed by counting the weighted number of points laying into each bin.

Unique Shape Context (USC) [22] USC is an extension of 3DSC which is more robust to the rotation and translation changes. The LRF is used instead of the LRA and it is computed starting from the neighborhood instead of the normal n to keypoint k (Fig. 1.14.c).

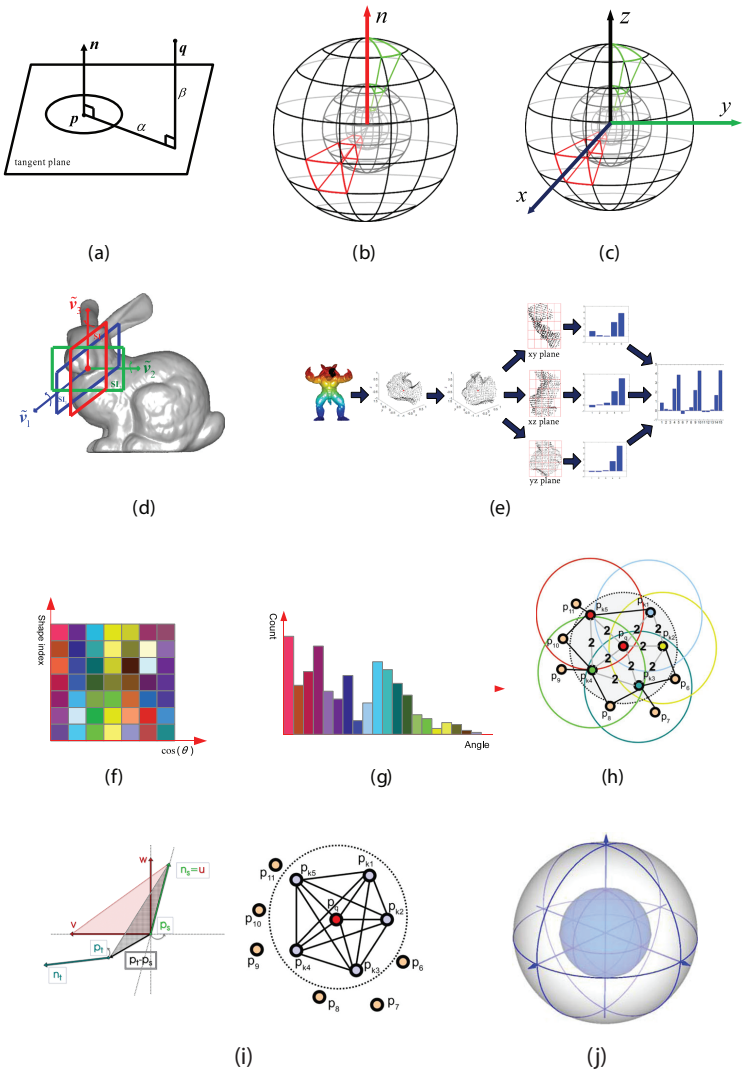

Figure 1.14: 3D Descriptors (from [18])

Rotational Projection Statistics (RoPS) [23, 24] RoPS is based on a novel, unique and repeatable LRF [24]. According to the LRF the points are rotated around the three coordinate axis. For each rotation the projection onto the xy, yz and xz space is performed and for each projection a distribution matrix is generated, which is used to encode five statistics. RoPS descriptor will be built starting from the concatenation of all the statistics (Fig. 1.14.e).

Tri-Spin-Image (TriSI) [25] TriSi builds a LRF in a similar way as it is done in RoPS. The surface is rotated according to the LRF and then a spin image is generated using the x-axes as its LRA. In addition, another two spin images are computed using the y and z-axes as the LRAs of these spin images (Fig. 1.14.d). The TriSI descriptor is formed by concatenating the spin images.

1.4.2 Geometric Attribute Histogram based Descriptors

These descriptors represent the local surface by generating histograms according to the geometric attributes (e.g., normals, curvatures) of the points on the surface.

Local Surface Patch (LSP) [26, 27] In LSP, the shape index [28] and the cosine of the angle between the normal of the neighboring points and the keypoint normal are calculated. The LSP descriptor is a 2D histogram, where the shape index value and the cosine of the angle between the normals are discretized in bins (Fig. 1.14.f).

THRIFT [29, 30] In THRIFT, a 1D histogram of the deviation angles between the normal of the keypoint k and normals of the neighboring points is built (Fig. 1.14.g). The density of point samples and the distance from the neighboring point to the keypoint are used to update the bins in a histogram.

Point Feature Histogram (PFH) [31] in PFH, for each pair of points in the neighborhood, a Darboux frame is defined using the normals and point

coordinates (Fig. 1.14.i). Next, four features are calculated for each point pair using the Darboux frame, the surface normals, and their positions. PFH is generated by accumulating points in particular bins along the four dimensions.

Fast Point Feature Histogram (FPFH) [32] First of all, a simplified point feature histogram (SPFH) is generated for each point by calculating the relationships between a point and its neighbors (Fig. 1.14.h). While in PFH there is the comparison between all the pairs in the neighborhood. FPFH is then built as a weighted sum of all the SPFH built upon every point in the neighborhood.

Signature of Histogram of Orientations (SHOT) [22] First, an LRF is computed for the keypoint k, then the neighboring points are aligned with this new LRF. Next, the support region is divided into several volumes along the radial, azimuth and elevation axes (Fig. 1.14.j). A local histogram is associated to each volume. The histograms are filled according to the angles between the normals to the neighboring points and the normal to the keypoint. The SHOT descriptor is the concatenation of all the local histograms. For further reading on the topic refer to [18].

Chapter 2

Tools

The present work of research has been completed using mainly Altera and Xilinx Tools and its Intellectual Properties (IP). Prototyping a new architecture starting from scratch is not a simple task, for this reason Altera and Xilinx tools come in help to the hardware designer. The developed IPs were produced using Verilog and High Level Synthesis (HLS) by Xilinx languages. The board used to prototype the proposed system is a Zedboard (Fig. 2.1) mounting a Zynq 7000 chip (part number: xc7z020clg484-1). The Zynq 7000 family is very handy when used in prototyping, in fact, the programmable logic is coupled with an ARM dual-core A9. ARM core is crucial when an actual debug on the circuit takes place, in fact, it is easier to feed simulated data to the architecture using a microprocessor. The ARM core is able to run both bare metal applications and Linux kernel, making even easier the architecture debugging. Furthermore, for every IP present in the Xilinx IP catalog a Linux driver is available, speeding up the prototyping time.

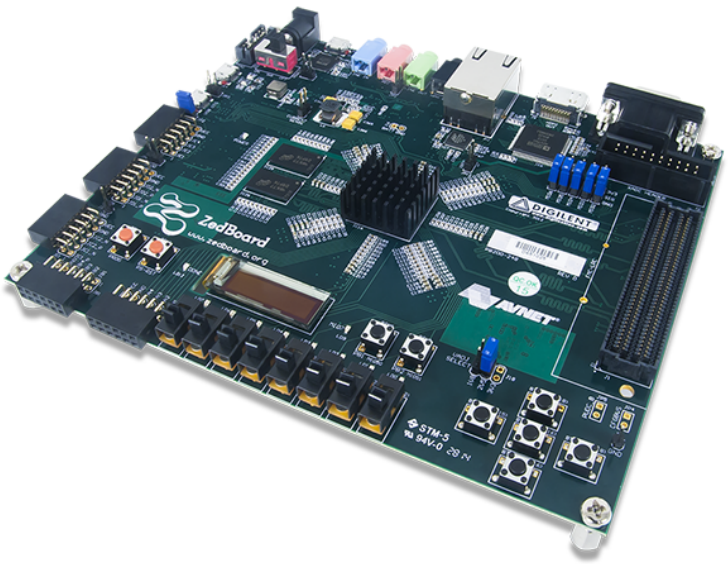

Figure 2.1: Zedboard.

2.1 ModelSim Altera Edition

Producing an IP could be a frustrating process, hours and hours spent simulating the circuits just to notice an error in the signal stimulations. The simulation is a critical task before the actual synthesis. For this purpose a lot of tools are available on the market and one of the best is without any doubt ModelSim Altera Edition (Fig. 2.2).

ModelSim combines simulation performance and capacity with the code coverage and debugging capabilities required to simulate multiple blocks and systems and attain ASIC gate-level sign-off. Comprehensive support of Verilog, SystemVerilog for Design, VHDL, and SystemC provide a solid foundation for single and multi-language design verification environments. ModelSim's easy to use and unified debug and simulation environment provide today's FPGA designers both the advanced capabilities that they

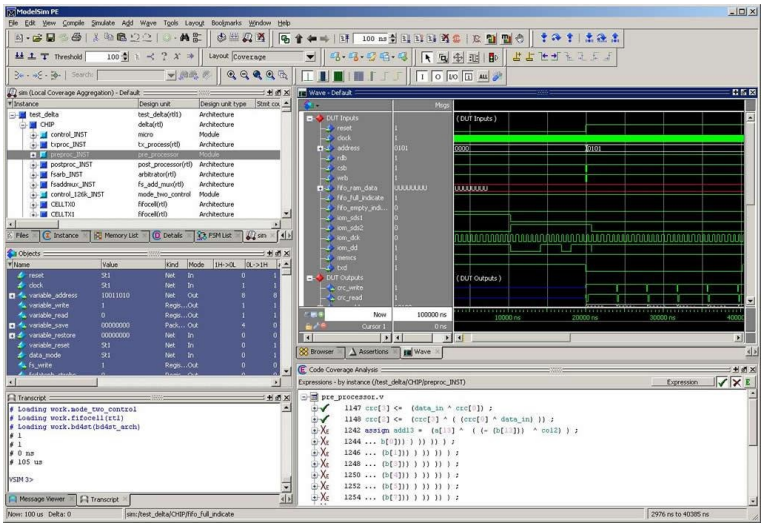

Figure 2.2: ModelSim Altera Edition.

are growing to need and the environment that makes their work productive.

The ModelSim debug environment's broad set of intuitive capabilities for Verilog, VHDL, and SystemC make it the choice for ASIC and FPGA design.

ModelSim eases the process of finding design defects with an intelligently engineered debug environment. The ModelSim debug environment efficiently displays design data for analysis and debug of all languages.

ModelSim allows many debug and analysis capabilities to be employed post-simulation on saved results, as well as during live simulation runs. For example, the coverage viewer analyzes and annotates source code with code coverage results, including FSM state and transition, statement, expression, branch, and toggle coverage.

Signal values can be annotated in the source window and viewed in the waveform viewer, easing debug navigation with hyperlinked navigation between objects and its declaration and between visited files.

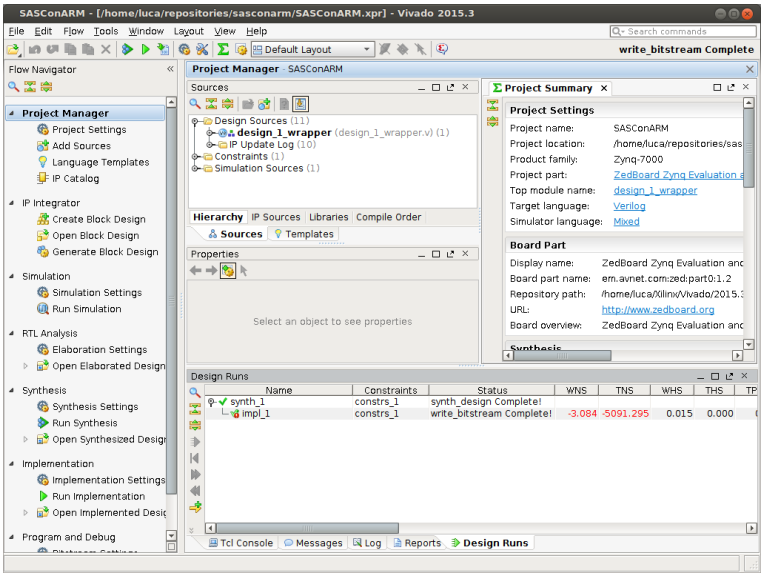

Figure 2.3: Xilinx Vivado.

Race conditions, delta, and event activity can be analyzed in the list and wave windows. User-defined enumeration values can be easily defined for quicker understanding of simulation results. For improved debug productivity, ModelSim also has graphical and textual dataflow capabilities.

2.2 Xilinx Vivado Design Suite

Vivado enables developers to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Vivado is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors.

Vivado was introduced in April 2012, and is an integrated design environment (IDE) with a system-to-IC level tools built on a shared scalable data model and a common debug environment. Vivado includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards based packaging of both algorithmic and RTL IP for reuse; standards based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems. A free version WebPACK Edition of Vivado provides designers with a limited version of the design environment.

2.3 Vivado High-Level Synthesis

Advanced algorithms used today in wireless, medical, defense, and consumer applications are more sophisticated than ever before. Vivado High-Level Synthesis included as a no cost upgrade in all Vivado HLx Editions, accelerates IP creation by enabling C, C++ and System C specifications to be directly targeted into Xilinx All Programmable devices without the need to manually create RTL. Supporting both the ISE and Vivado design environments Vivado HLS provides system and design architects alike with a faster path to IP creation by :

- Abstraction of algorithmic description, data type specification (integer, fixed-point or floating-point) and interfaces (FIFO, AXI4, AXI4-Lite, AXI4-Stream)
- Extensive libraries for arbitrary precision data types, video, DSP and more... see the below section under Libraries
- Directives driven architecture-aware synthesis that delivers the best possible QoR
- Fast time to QoR that rivals hand-coded RTL

- Accelerated verification using C/C++ test bench simulation, automatic VHDL or Verilog simulation and test bench generation
- Multi-language support and the broadest language coverage in the industry
- Automatic use of Xilinx on-chip memories, DSP elements and floating-point library

Chapter 3

Stereo Vision on FPGA

Many FPGA architectures for stereo vision have been proposed in recent years, but not without limits; for example, many of these use very expensive FPGAs (e.g. [33, 34, 35]). With nowadays technology it is not possible to obtain optimal quality with high frame rate on a cheap device. Another limit is the maximum number of disparities although this is not a huge problem when taking into account the field of use before synthesis; most of FPGA implementations have a maximum disparity level that leads the structure to be more regular and compact, but, of course, constraining the minimum depth of the final disparity map. The aim of this work is to obtain a good trade-off between performance and platform cost/power consumption. I therefore decided to implement a global method inspired by a bio-Informatics algorithm [36] on a Zynq-7000 FPGA. According to [8] every stereo matching algorithm can be divided in four (or less) phases: 1) matching cost computation 2) cost aggregation 3) disparity computation and 4) disparity refinement. In my implementation we have no cost aggregation nor disparity refinement to further reduce the resource requirement. This global method is very time consuming on both CPU and GPU, but it is possible to enhance the performance on FPGA by exploiting the well-known

structure of systolic array. In this work I present:

1. A hardware-friendly algorithm of a global method inspired by a previously presented bio-Informatics approach, that uses, heuristics to reduce the computation time with a minimum quality degradation of the disparity map;

2. A hardware architecture to implement the algorithm using reusable non optimized code, so that is possible to remap the aforementioned architecture to another FPGA board, without further modification;

3. A synthesis report for Zynq-7000 (part number XC7Z020CLG484-1) reaching real-time processing up to XGA resolution;

4. A comparison with the state-of-the-art FPGA architecture and the implemented low power counterpart.

3.1 The algorithm

Many approaches have been proposed to address the stereo vision task complexity, the vast majority of classical methods present a matching mechanism that tries to couple pixels in two stereo images. The simplest solutions use sliding pixel windows to do local matching, a more robust solution is represented by global methods that try to align whole pixels lines boosting the disparity map quality, to exploit benefit from both algorithm categories some semi-global method were proposed. In [36] a bio-informatic inspired approach is proposed that is based on the Needleman & Wunsh algorithm, a Dynamic Programming algorithm to find the optimal alignment in a nucleotides or amino-acids string [37].

Every DP algorithm consists of two phases: in the first one a score matrix is filled whose dimension depends on the input sequence length; the number of columns and rows corresponds to the length of the first and second

input sequences characters respectively. In the second phase the solution is extracted from the matrix via backtracking.

Let A and B two DNA sequences and F the score matrix, the following rules are used to fill the matrix:

Basis:

$$F_{0,j} = j \times GAP$$
$$F_{i,0} = i \times GAP$$

Recursion:

$$F_{i,j} = max \begin{cases} F_{i-1,j-1} + S(A_i, B_j) \\ F_{i,j-1} + GAP \\ F_{i-1,j} + GAP \end{cases}$$

Given the score dependency of previous rules, each cell can be filled only when the three upper right cells are computed (NORTH-WEST, NORTH, WEST) as shown in figure 3.1, alongside the score a direction is stored to indicate the score source.

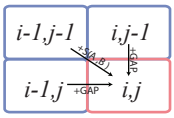

Figure 3.1: How the scores are produced during the scoring phase.

In 3.2 is shown a common example of nucleotides alignment. The function $S(A_i, B - j)$ is mapped using two values, when $A_i = B_j$ we have a match $(S(A_i, B_j) = 1)$ on the other hand when $A_i \neq B_j$ we have a mismatch $(S(A_i, B_j) = 0)$. The penalty GAP is used to model the dis-alignment in the two sequences, in particular when the score produced by the north or west cell is higher than the score produced by diagonal cell we are in a GAP situation in which a dash is added to the final alignment, this is due to

problems like insertion or deletion in the two DNA sequences. When a new cell is computed a direction is produced to mark the cell from which max value is found.

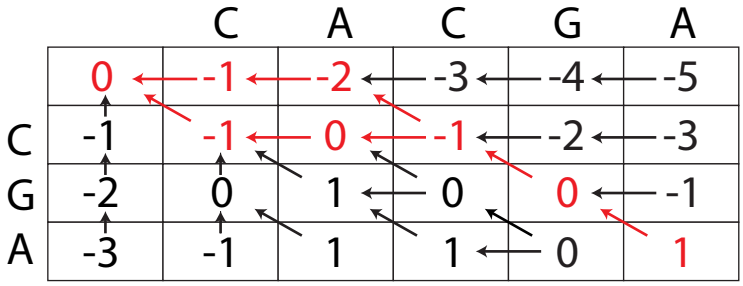

Figure 3.2: Needlemean and Wunsh example with match = 1, mismatch = 0, GAP = -1

This phase is also called scoring procedure, the final result is a score matrix in which the element in the last cell represents the value of the match between the two sequences. Knowing only the score of the match is useless, it's important to track the precise order in which the nucleotides are matched. To get the solution of the alignment, after the filling process a backtracking phase begins. In this process a path is built starting from the element in the last column and last row, following the directions stored alongside with the scores is possible to return to the first element in the matrix (red arrow in 3.2).

If two or more directions are stored in a single cell multiple optimal alignments arise during backtrack phase, this happens when multiple maximum values are produced in the recursive step of the scoring procedure. In this case some heuristics must be used to prefer an alignment over the others.

The path in red shown in 3.2 is an alignment that must be translated into a sequence of nucleotides, each red arrow in the path is translated into a character, starting from the first cell three things can happen: if a direction is WEST a GAP is inserted (-), if the directions is a NORTH no characters are added to the final result, in case of NORTH-WEST direction

the respective character is copied, so from 3.2 we obtain the following two alignments:

$$\begin{array}{c} --CGA \\ C--GA \end{array}$$

In [36] N&W is used to align pixels scanlines, the algorithm can be summarized with the following recursion rule:

$$F_{i,j} = max \begin{cases} F_{i-1,j-1} + S(A_i, B_j) \\ F_{i,j-1} + \{GAP \vee EGAP\} \\ F_{i-1,j} + \{GAP \vee EGAP\} \end{cases}$$

To use the algorithm for stereo matching purposes image pixels are substituted in place of nucleotides. The principal difference between DNA sequences and images alignment is the frequency of GAP penalty: while it is usual to find small recurring differences in two DNA sequences, in the context of stereo vision many large GAP zones are present, that represent occlusion in the images, i.e. parts of the scene visible from one camera but hidden in the other. To correctly model occlusion during alignment it is necessary to introduce the parameter Extended GAP ($\mathcal{EGAP}$) that promotes occasional large misalignment zones over small recurring $\mathcal{GAP}$∫.

Let IL_x and IR_x be the $x-th$ homologous lines in the left and right images, w their length, $\mathcal{M}$, $\mathcal{GAP}$ and $\mathcal{EGAP}$ respectively the Match score, the GAP and Extended GAP score (both constant) and $\mathcal{MS}$ be the MiSmatch score set proportional to the pixel relative distance in RGB space. The algorithm is shown in Fig. 3.3.

I have chosen to develop an FPGA architecture based on this algorithm to exploit:

1. the benefit of a gloabal approach on the quality of the disparity map;
2. the low complexity given by the Dynamic Programming
3. the intrinsic parallelizability of the operation during the score phase;

for $i \leftarrow 1 \dots w$ **do**
  for $j \leftarrow 1 \dots w$ **do**
    $\mathcal{MS} = -|IL_x(i) - IR_x(j)|$
    if $is\_GAP(M(i-1,j))$ **then**
      $nord \leftarrow M(i-1,j) + \mathcal{EGAP}$
    else
      $nord \leftarrow M(i-1,j) + \mathcal{GAP}$
    end if
    $diag \leftarrow M(i-1,j-1) + \mathcal{M} + \mathcal{MS}$
    if $is\_GAP(M(i,j-1))$ **then**
      $west \leftarrow M(i,j-1) + \mathcal{EGAP}$
    else
      $west \leftarrow M(i,j-1) + \mathcal{GAP}$
    end if
    $M(i,j) \leftarrow max(north, diag, west)$
  end for
end for

Figure 3.3: Needleman & Wunsh scoring procedure for stereo vision

3.1.1 Heuristics and Optimization

The Needleman & Wunsh algorithm as every global method is quite expensive in the number of operations. In this hardware implementation I tried to reduce this number using heuristics, while the remaining operations have been parallelised. Making some assumptions like the maximum disparity (minimum depth) is possible to further enhance the quadratic DP algorithm to a linear complexity, the parallelisation is achieved by a scalable parametrised systolic array.

The first chosen heuristic is the removal of multiple optimal backtrack path, in fact, when two or three maximal value from NORTH, DIAG and WEST score are computed multiple optimal path arise, to avoid this situation only one direction is stored.

But the principal heuristic used in the implementation is the reduction of the score/direction matrix size. While in the original algorithm a whole square matrix is filled, in this work I have first removed all the cells in the upper triangular part (representing alignment behind the background) and then removed all the cells below a certain distance from the principal diagonal (high disparity levels) (Fig. 3.4).

A further memory reduction can be done limiting the minimum depth recognizable, this heuristic is very common in stereo vision context on

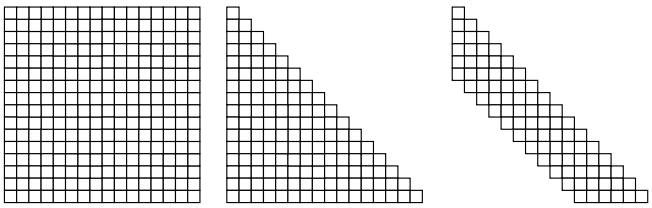

Figure 3.4: Score Matrix with different optimization: a) no optimization; b) upper triangular remove; c) cut on the maximum disparity;

FPGA. In general, two corresponding scanlines are very similar, this imply a small band of the matrix utilized in the backtrack. Cutting the matrix (3.4.c) reduce the minimum distance in the depth map and so the maximum disparity:

$$(d_{max} > d(x,y) \forall (x,y) \in I_L)$$

This is not a relevant problem since the minimum distance recognizable can be adjusted modifying the distance between the camera. By using:

$$Z = \frac{Bf_z}{d}$$

equation it is possible to calculate the minimum distance the system can recognize, with an higher maximum disparity object closer to the cameras are recognizable.

Since two scanlines are very similar and the resources on a FPGA are very limited it is important to reach the right trade-off between precision and power consumption. This choice can be made at synthesis time since the system is parametrised and it is very simple to modify. To further reduce the use of BRAMs only the direction matrix is stored in them.

3.2 Architecture

Given the score dependency of the algorithm, each cell can be filled only when the three upper right cells are computed (NORTH-WEST, NORTH, WEST). According to other authors [38] there is only one way to correctly parallelise this algorithm, this task is accomplished by filling the score matrix in antidiagonal order. A systolic array was implemented to parallelise the procedure of filling multiple cells in the same clock tick.

The systolic array is composed by Processing Elements (Fig. 3.5), PE from now on. Each PE has four inout vector ports plus four sync signals. Two ports are used for the pixel stream while the other two are used to exchange scores from the neighbour cells. Four other signals (enable and ready) are

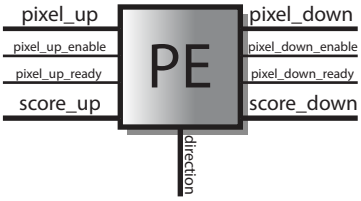

Figure 3.5: Processing Element interface with wire direction.

necessary to synchronize the modules. If no pixels are available, deasserting enable signal to the first PE stops the whole computation. Each PE fills a cell in antidiagonal order in a two phases (RIGHT-DOWN) stair-likes pattern.

In 3.6 are shown three PE used to compute the value of a score matrix, the computation proceed in antidiagonal order, the number progression in the figure indicates the order in which the cell are filled. In the first and second clock tick only two cells are computed by PE_1, in the third and fourth tick PE_1 and PE_2 are used to fill 4 cells, from that moment at each clock tick three cells are filled altogether. By the 28th and 30th clock tick respectively PE_3 and PE_2 became useless. Finally, in the last two ticks PE_1 finish to fill the matrix.

In 3.7 the systolic array is shown. As explained before, the number of PE determines the maximum recognizable disparity (d_{max}). The chosen approach is very similar to the one shown in [39] to solve dynamic programming on FPGA context. To avoid the use of a huge multiplexer to feed the PE in random access order in [39], the PE are feed in sequential mode, so the array is also used as pass-through channel for the data.

While in [39] the first PE able to compute a direction is the one in the middle of the array, in this work I have optimized this architecture to adhere more strictly to the problem field. Each element of the systolic array is filled starting from the right PE with the right image pixels (in a FIFO queue manner). When the first right pixel ($RP0$) met the first left pixel ($LP0$), the computation proceed fully pipelined till the end of the two lines

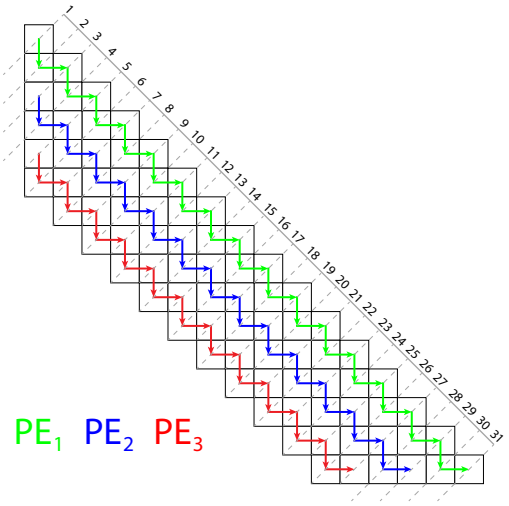

Figure 3.6: Processing order for the PEs.

(Fig. 3.8 and Fig. 3.9). Furthermore my architecture does not require $2 \times n$ PEs as in [39] since the data streams are introduced in the array one per clock tick, this gives a further reduction to the architecture area.

The whole processing can be summarized as a two phases computation. In the odd cycle PE_i uses the previous cached scores as WEST and NORTH-WEST, while the NORTH is read from PE_{i-1}; at the same time PE_{i+1} does the same as PE_i but uses its WEST score as a NORTH since it is placed one cell down and one cell left with respect to PE_i. In the even cycle the PE_i uses the two previous cached score as NORTH-WEST and NORTH, reads the WEST from PE_{i+1} and in the same way PE_{i-1} reads his WEST score from the NORTH score of PE_i. The values produced by PEs outside the matrix at the start and at the end of the computation are stored, but are ignored during the backtrack phase.

The BRAM requirement in the design is very little since it is directly proportional with the length of the scanlines. As mentioned previously only

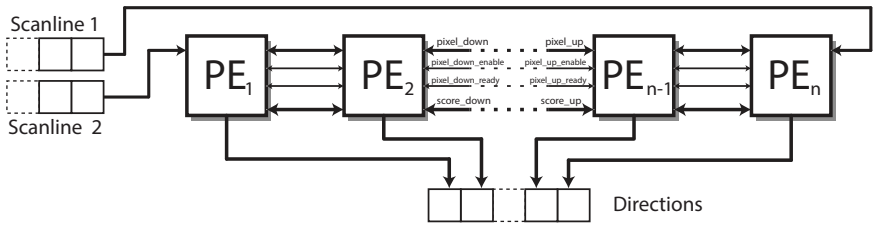

Figure 3.7: Systolic Array with $n = d_{max}$ Processing Elements.

the directions are stored in the BRAM, while the scores are exchanged by the PEs during the computation and hence it is not necessary to store them. Since each direction is represented using 2 bits (NORTH-WEST-DIAG) and at each clock tick d_{max} directions are produced, the BRAMs are configured to have words of length $2w$ bits and size $2d_{max}$ with a grand total of $2 \times w \times d_{max}$ bits requested by each direction matrix. The backtrack phase needs to read these directions and since a BRAM has a latency clock cycle to be read, two clock cycles are used to traverse each cell of the solution.

3.2.1 Complexity

During the scoring phase each PE processes $2w$ cells, bringing to the complexity of $O(2w)$. After the score procedure a backtrack phase starts so that the optimal alignment is computed. This last phase traverses in the worst case $w + d_{max}$ cells and since a BRAM read has one cycle clock latency, the tracker pointer needs 2 clocks cycle to move from one cell to the next, and this leads to the complexity of $O(2(w + d_{max}))$ clock to resolve a single lines pair which is linear with the size of the input. A further improvement is done pipelining the two phases permitting to work simultaneously on two scanlines couples, so while the systolic array is processing a line pair at the same time the backtrack module is computing the optimal alignment of the previous lines pair; this is possible only storing both direction matrix for both scanline pairs. This double buffer approach bring to the final

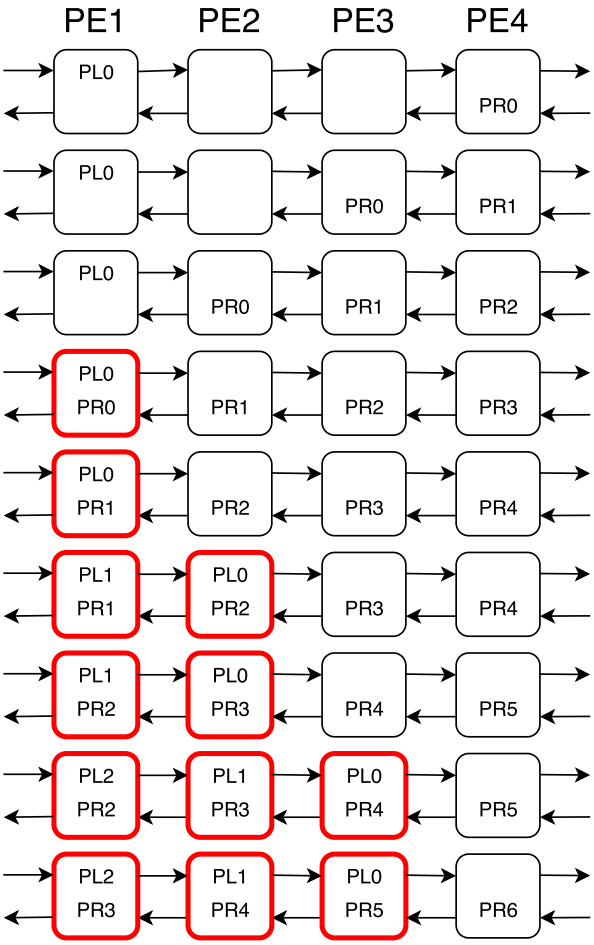

Figure 3.8: Systolic Array time sequence with four Processing Elements, Starting phase: in red are highlighted the PEs in which the actual computation take places.

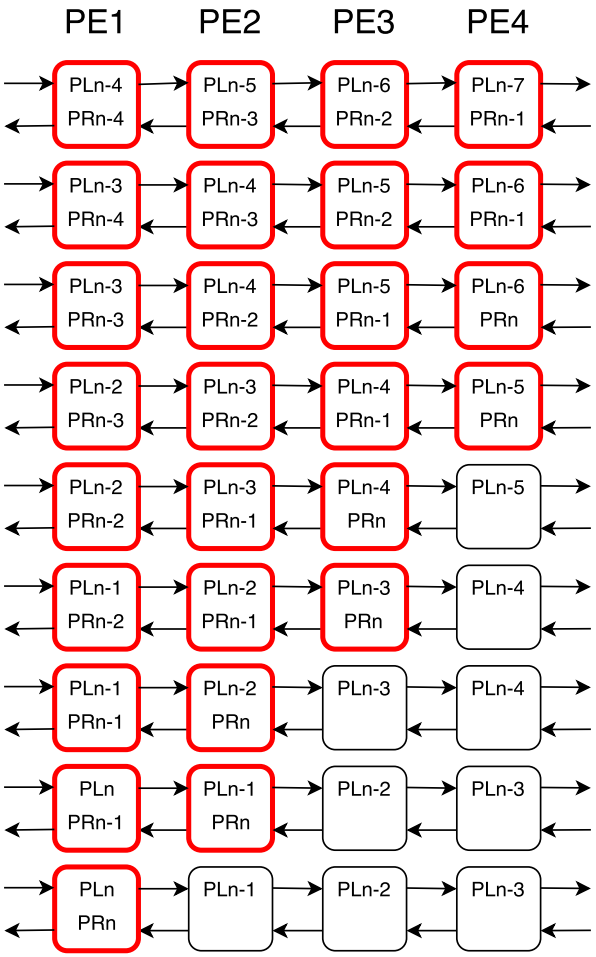

Figure 3.9: Systolic Array time sequence with four Processing Elements, Ending phase: in red are highlighted the PEs in which the actual computation take places.

complexity of $max(O(2w), O(2(w + d_{max}))) = O(2(w + d_{max}))$

The two scanlines flow in the array from the first and the last PE, when the first pixel of both scanlines meets each other in PE_1 the computation starts and at every clock tick d_{max} directions are produced; since each direction can have three different values, NORTH, WEST and NORTH-WEST, 2 bits are required; the directions produced are stored in a BRAM with word length $2d_{max}$ and size $2w$. The total BRAM space utilized in the architecture is of $4(w \times d_{max})$ bits, a very little amount since the common FPGA BRAMs size is in the order of few Mega bits.

3.3 Performance & Comparison

Synthesis has been made on a Zynq-7000 All Programmable SoC Xilinx chip (part number XC7Z020CLG484). Syntesizing a single PE I have obtained a little occupation, only 249 LUT and 116 FF. The combinatorial logic occupation is double respect to the sequential logic due to the operations executed in the PEs. Given the computational complexity of $O(2(w+d_{max}))$ to complete a single scanlines couple, it is possible to calculate the number of clock ticks needed for various image resolution as shown in Tab. 3.1.

Table 3.1: Clock Ticks needed to process a real time (30 fps) video stream at different resolution (Upper bound)

Resolution	#clk per frame	#clk for real-time
1024×768	1,572,864	47,185,920
800×600	960,000	28,800,000
640×480	614,400	18,432,000
320×240	153,600	4,608,000

This means that with a 50 MHz clock it is possible to obtain real-time performance (>30fps) on image size up to XGA (1024×768), a really good performance for a global optimization algorithm on a cheap device.

Figure 3.10: Algorithm results on Middlebury standard datasets.

The algorithm quality is shown in 3.10. As with every scanline optimization algorithm it suffers from horizontal strike problems.

Since some heuristics have been adopted in the hardware implementation (multiple optimal path removal and maximum disparity), the resultant disparity quality is a little degraded. Respectively the error rate for the image in the Middlebury dataset [16] Tsukuba, Venus, Teddym Cones is: 7.19%, 9.54%, 21.0%, 17.8%. As we can observe the difference between FPGA implementation and the original in [36] is very small.

In Table 3.2 various FPGA implementations are compared in term of various parameters: resolution, disparity levels, frame per second and other two measure to allow the comparison of heterogeneous architectures. Each implementation uses a global or a semiglobal solution, other architectures using local methods are not compared (i.e. [40]) since the intrinsic difference of the approaches. Let the number of disparity pixels present in a frame be dp, the frame rate fr and the number of dispairty level d_{max}, the Million Disparity Evaluated per second (MDE/s) is calculated as $dp \times fr \times d_{max}$, and is expressed in millions per units. Clearly this measure might be misleading because cheaper FPGA will never compete with larger and expensive devices, the same it could be said taking into account that FPGA built on older

Table 3.2: Power Estimation

	Resolution	DL	FPS	MDE/s	ER	FPGA	PC	EE mW
Ours	**1024 × 768**	**64**	**30**	**1510**	**14%**	**Vx-7 XC7Z020**	**0.172W**	**0.11**
[41]	680 × 400	128	25	870	8.13%	Vx-4 XC4VFX140	1.243W	1.42
[42]	1024 × 768	128	129	13076	7.65%	Sx IV EP4SGX230	1.518W	0.12
[33]	1024 × 768	60	58.7	1082	8.71%	Vx-4 XC4VLX160	1.404W	1.3
[35]	1024 × 768	64	60	3019	8.2%	Sx III EP3SL150	1.558W	0.52
[43]	640 × 480	128	103	4050	6.7%	Vx-5 XC5VLX	2.313W	0.57
[34]	1600 × 1200	128	42.61	10472	5.61%	Sx-V 5SGSMD5K2	2.792W	0.26
[44]	1024 × 768	60	199.3	9362	6.05%	Vx-6 XC6VLX240T	3.350W	0.36

technology nodes are less power efficient, but to the best of my knowledge I have tried to do the most accurate and precise report. For this purpose I introduced the Power Consumption (PC) of the FPGA logics only (no I/O nor peripherals) for each architecture. Since it is not common to find power usage information about the compared architectures I retrieved this data using Xilinx and Altera tools for the power estimation. These tools start from the FPGA type, area occupation and clocks speed to compute a power analysis of the architecture. Since many papers do not provide information like BRAM and DSP usage, flip-flop occupancy or clock frequency, the table numbers represent always a lower bound to the actual value (no I/O devices are taken into account), meanwhile my implementation has been estimated in the most precise and accurate way since all the architecture module are well known. As a final measure to make the architecture performances directly comparable we define the Energy Efficiency (EE) as $PC/MDE/s$, this value express how much power is used to compute one milion disparities in one second. As we can see my architecture is the most efficient in term of power consumption in absolute terms and in terms of Watts per MDE/s and is implemented on the cheapest chip.

As a final result I show the synthesis obtained using 90nm standard cell technology:

Worst Slack	8893.80 ps
Total Power	46.14 mW
Total Area	777153 um2

my architecture improves greatly the performance (2× faster) and the

power consumption (3.7× less) if we opt for a design on chip instead of a re-configurable device.

3.4 Prototype

In the prototype 3.11 two MT9D111 sensors are used as a stereo camera system and a Digilent Zedboard^{TM} is used to connect the Zynq-7000 to the two cameras and to a LCD monitor. As an example application I have synthesized a design for the real-time processing of a video stream at 1024×768 30 fps with 64 disparity level. The final design occupies 29057 LUT (57%), 15208 FF (14%) and 2.6 Mb of BRAM (53%). This architecture greatly outperforms the software version of the program, in fact the CPU time for processing a single frame is of 5.4s on a Intel i7-4558U CPU @ 2.80GHz with 8Gb RAM, leading to a speed-up of 162x;

Barely half of the FPGA is occupied and synthesis correctly closed at 50 Mhz. The total power consumption (with I/O and peripherals) is just 2 Watts. The proposed architecture is easily customizable at synthesis time since no optimization for the target device has been performed, and by using verilog standard code it is possible to migrate easily to every FPGA platform.

In 2.3 the final architecture is shown, as can be seen from the Vivado block diagram viewer. The camera controllers are highlighted in orange, each of them is connected directly to the output pins of the PMODs and the pixel bits are sent in parallel. The two chosen cameras can work in RGB mode so that no debayering is necessary and the two streams are then sent to the main memory (a DDR2 memory) through two Video Direct Memory Access (VDMA). The VDMA is a Xilinx IP used to manage video data: briefly, by using a start address and a frame size it is possible to acquire a frame and save it to a memory location. The stereo vision co-processor is highlighted in green. The input data are streamed from the frame buffers to the main memory, then, when the output is ready it will be flushed again to the DDR memory. The final step of the process is performed by the VGA-controller

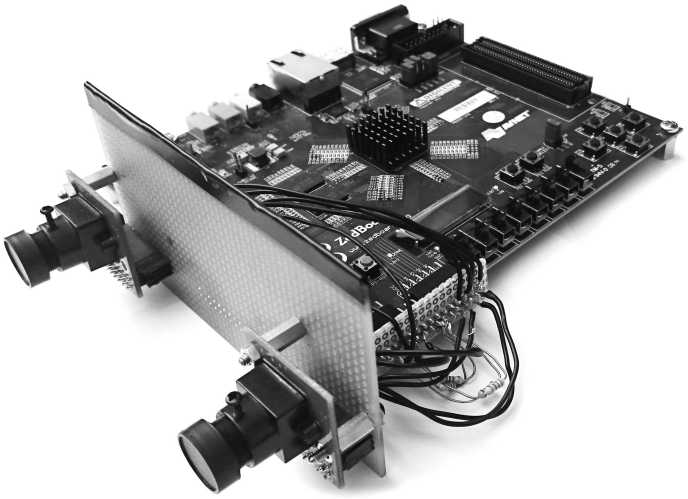

Figure 3.11: Final prototype of the device, the cameras are connected through the PMOD pin, also four resistors are needed to pull up the i2c interfaces.

which is highlighted in purple, the controller is directly connected to a DAC converter that will output the disparity map on a VGA monitor.

3.4.1 Camera Controllers

Two MT9D111 sensors are used as cameras for the system. These are attached to the Zynq usign PMOD connectors on the Zedboard. Two kind of PMOD connectors are present on the board: the PMODs connected to the Programmable Logic and the PMODs connected to the ARM core. In order to configure the cameras, an I2C interface is available on the external pins. The interfaces are directly connected to the ARM core through the second kind of PMODs, while the 8 pins used to transmit the pixels are connected to the Programmable Logic. The camera can be configured in RGB 565

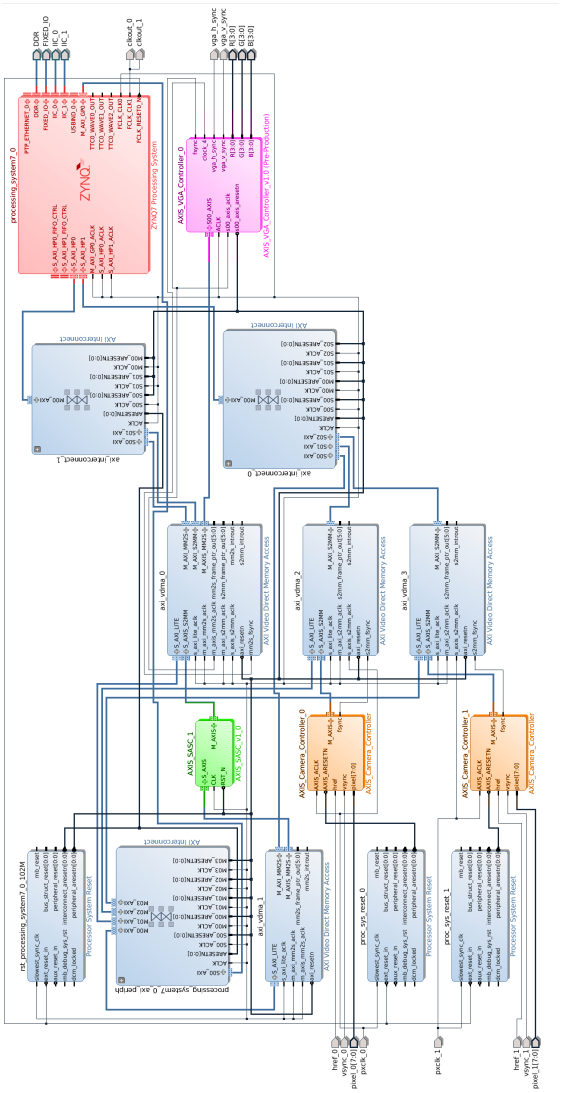

Figure 3.12: The architecture developed for the Stereo Vision system.

format, in this configuration the 8 external pins are used to transmit 1 pixel every two clock cycle in the following pattern:

- In the odd cycle $|R_7|R_6|R_5|R_4|R_3|G_7|G_6|G_5|$ are transmitted
- In the even cycle we have: $|G_4|G_3|G_2|B_7|B_6|B_5|B_4|B_3|$

Camera controllers are used to deserialize this stream in such a way that for every clock tick a complete pixel is available, the clock frequency in the output will be halved with respect to the camera one. Furthermore, since the architecture is real time, no buffering is needed neither in the camera controllers nor in the hardware co-processor.

3.4.2 VGA Controller

The zedboard has a VGA connector that can be driven by the Programmable Logic. Between the actual connectors and the PL, a DAC converter is available, which has just 4 bits per channel. Since we are using 64 disparity levels, it is necessary to map each level to a color using 4+4+4 bits, an heat map color scheme in which the hotter the color the closer the pixel has been used:

```
1  assign R =  (!tready)   ?  4'b0000:
       (disparity < 128)  ?  4'b0000:
       (disparity < 192)  ?  disparity[5:2]:
                             4'b1111;

6  assign G =  (!tready)   ?  4'b0000:
       (disparity < 64)  ?   disparity[5:2]:
       (disparity < 192)  ?  4'b1111:
                    ~disparity[5:2];

11 assign B =  (!tready)   ?  4'b0000:
       (disparity < 1)   ?   4'b0000:
       (disparity < 64)  ?   4'b1111:
       (disparity < 128)  ?  ~disparity[5:2]:
```

```
4'b0000;
```

Listing 3.1: Heat Map conversion from disparity level.

An LCD monitor is then plugged to the socket to visualize the result.

3.5 Conclusion

I have presented a new architecture for scanline optimization in a stereo matching context. We utilised a global method to achieve real-time performance for higher resolution than VGA standard. My systolic array has reached good results thanks to the chosen heuristic, e.g. limiting disparity levels and storing only one direction per cell in the score matrix. The power usage is very low; no CPU nor GPU can achieve this power consumption with similar performance.

I have compared my work with the performance of other proposed architecture and showed that my implementation is the most energy efficient and based on the cheapest device, furthermore the results show a boost of 165 times over the software implementation.

The highly parametrized design is intended to be used on every FPGA brand, no optimization has been made regarding the target device, the design is written in plain verilog code and further optimization can be achieved taking into account the target device, but my goal was to maintain the code as general as possible.

Chapter 4

3D Descriptors and Detectors

Arguably the most ubiquitous task performed on 3D data for the aforementioned computer vision applications is represented by the Nearest Neighbors Search (NNS), i.e. given a query 3D point, find its k nearest neighbors (*kNN Search*), or, alternatively, all its neighbors falling within a sphere of radius r (*Radius Search*). This is for example necessary for computing standard surface differential operators such as normals and curvatures. In addition, NNS is a required step also for keypoint detection and description on 3D data, which are deployed, in turn, for 3D object recognition and segmentation. Another relevant example (among many others) of the use of NNS is the Iterative Closest Point (ICP)[45] algorithm, a key step for most 3D registration, 3D reconstruction and SLAM applications.

When NNS has to be solved on a point cloud, being it an unorganized type of 3D data representation, efficient indexing scheme are typically employed to speed up the otherwise mandatory linear search. Nevertheless, despite such schemes are particularly efficient, the NNS on point clouds can still be extremely time consuming, since the complexity grows with the size of the point cloud. In particular, over the years several methods have been proposed to optimally solve the NNS problem in the fastest way possible

based on heuristic strategy [46], clustering techniques (e.g. hierarchical k-means, [47]) or hashing techniques [48]. Currently, the most popular approach is the kd-tree approach [49], or its 3D-specific counterpart known as octree.

In addition to exact algorithms, also approximated methods have been proposed, which trade-off a non optimal search accuracy with a higher speed-up with respect to the linear search. In [50], a modified kd-tree approach known as Best Bin First (BBF) is presented, where a priority queue with a maximum size is deployed to limit the maximum number of subtrees visited while traversing the tree bottom up, i.e. from the leaf node to the root. In [51] a similar approach is proposed, where the stop criteria is imposed as a bound on the precision of the result. More recently, [52] proposed the used of an ensemble of trees where the split on each dimension is computed randomly and that rely on an unified priority queue: such approach is known as multiple randomized kd-trees, or randomized kd-forest. In [53], a library including several approximated NNS algorithms is proposed, including multiple randomized kd-trees[52], BBF kd-tree[50] and hierarchical k-means[47]. In addition, [53] also proposes a method to automatically determine the best algorithm and its parameters given the current dataset. Such, library, known as Fast Library for Approximate Nearest Neighbors (FLANN), is one of the most used libraries for NNS on point clouds: for example, it is the default choice for NNS within the Point Cloud Library (PCL)[54], the reference library for 3D computer vision and robotic perception.

Although all of the aforementioned methods for approximated NNS on points clouds can be used also on range images simply by turning this 3D data representation into a point cloud, it is possible to leverage on the organized trait of such data representation to speed up the search. Nevertheless, exploiting the 2D grid available when dealing with range images is not trivial, since nearest neighbor on the 2D grid are not guaranteed to be nearest neighbors also in 3D space (think about two points lying nearby on the image plane but on two different sides of a depth border). Furthermore, and especially for the Radius Search case, it is not trivial to turn a metric

radius into a pixel-wise radius in the general case, when calibration data are not available.

In literature, the specialization of NNS to the case of range images is almost unexplored. One of the most relevant techniques is the one implemented in the PCL library for both the Radius Search and the kNN Search, where the main idea is to adaptively define the extent in pixels of the search area on the image based on the 3D data as well as the camera parameters. In particular, in the kNN Search case, the query point is first projected onto the range image by explicitly taking into account the intrinsic camera parameters and the camera pose. In case the projected point lies outside the range image, the nearest element in the image is used as start position. Then, the first k nearest neighbors are sought for by looking in the nearby positions on the image plane. The search area in pixel is then defined based on the distance, projected on the image plane, of the query to the farthest point among the found k neighbors, which is finally searched exhaustively to refine the list of retrieved neighbors. Instead, in the Radius Search case, the intrinsic parameters and the camera pose are used also to translate the input metric radius into the pixel-wise radius of the search area on the image plane, by projecting the estimated 3D spherical neighborhood onto the image plane of the sensor. Additionally, each neighbor on the range image 2D grid is also checked with respect to its 3D distance from the query before being added to the list of neighbors. Notably, this radius search algorithm is similar to the one presented in [55], where NNS is applied to the specific task of normal estimation on range images.

In this work I present a method, dubbed *Radial Search Method* (RSM), which can be used as an alternative to the methods available in PCL[54] for fast approximate NNS on range images. The idea of my approach is, starting from the query point, to incrementally look for neighbors on the 2D grid along radial regions of increasing radius. Specific stop conditions are employed to terminate the search when the candidates obtained are estimated to approximate well enough to the real set of neighbors. In particular, I propose two variants of such approach, derived for both the kNN Search and the Radius Search problems. Notably, and advantageously

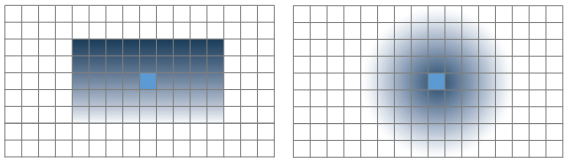

Figure 4.1: Depiction of *Organized* (left) and RSM (right) search strategies: the light-colored square in the middle is the query point, while the darker a cell is the earlier it is explored.

with respect to [54, 55], my method does not require knowledge of either the intrinsic parameters or of the sensor nor the sensor pose. By means of experimental results on 3D data obtained with a consumer RGB-D camera, I evaluate the performance of my approach against the NNS methods in PCL and against the FLANN randomized kd-forest approach in terms of both accuracy as well as efficiency. In addition, I also show how RSM can help in improving the performance of a typical 3D computer vision application of NNS such as 3D keypoint detection and description.

4.1 RSM algorithm

The presented approach is based on incremental exploration of the neighborhood starting from the query point, along concentric frames. While in the NNS search for range images available in PCL[54], hereinafter referred to as *organized*, the search is carried out over an image sub-region row after row (*raster scan*), in my method the search is made in radial order as depicted in Figure 4.1. This allows my algorithm to evaluate fewer points in the neighborhood of the query point to obtain a similar level of approximation in the search result. Another important characteristic of my exploration strategy is that it is adaptive, i.e. the size of the 2D neighborhood that is considered changes at each query point by evaluating a stop condition that depends on the improvements of the search at each step.

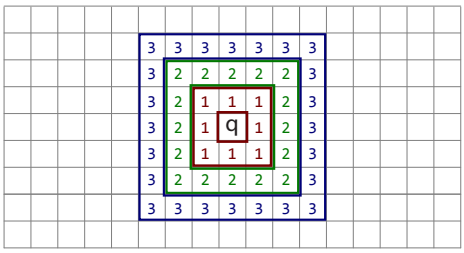

Figure 4.2: Elements in the set of frames $f_1(q)$, $f_2(q)$ and $f_3(q)$.

Let q be the query index and e an element of the range image. If the query point is only available as a point in 3D space, it is projected onto the image and the nearest element to the projection is used as starting point q. We define a non-euclidean distance $\hat{D}(q, e)$ on a range image I as the minimum number of horizontal, vertical or diagonal moves to reach e from q. I also use the classic euclidean distance $D(q, e)$ to measure the distance in 3D space between the (x, y, z) points corresponding to q and e. Furthermore, we define $\mathcal{Q}$ as a min-priority queue of (e, p) pairs in which e is the index of a range image element and $p = D(q, e)$ is the priority key. Finally, we define the frame at distance h from a query point q as:

$$f_h(q) = \{e \,|\, \hat{D}(q, e) = h\}$$

Figure 4.2 shows the elements belonging to the sets of the first 3 frames, i.e. $f_1(q)$, $f_2(q)$ and $f_3(q)$, of a query point q.

The key idea is to explore, at every iteration, the space of 3D point candidates defined by one full frame of pixels around the query point, and to stop the search whenever the number of *inliers* (e.g., found nearest neighbors) in the currently explored frame is too low. It is also important to make the exploration robust with respect to the presence of invalid points in the range image. In fact in many practical cases the acquired range images contain multiple invalid points, due to limitations in the sensing range or

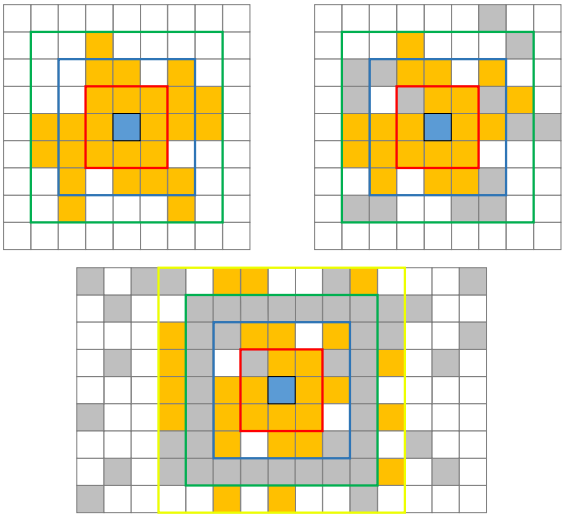

Figure 4.3: The solution is represented by yellow points, while invalid elements are denoted in gray. On the left, an example of the solution in the absence of invalid points; on the right, an example of the same neighborhood where some of the points are invalid, e.g. due to a change in viewpoint. Bottom: a case where an entire frame is composed of invalid points. The valid points in the neighborhood should still be evaluated and inserted in the solution.

in dealing with specific surfaces, such as dark and reflective surfaces for active sensors, non-Lambertian surfaces and low-textured regions for passive sensors. Figure 4.3 shows how the algorithm has to work in the presence of invalid points. Even if an invalid point is encountered along the exploration of a frame, the search must continue beyond it. In the extreme case of one or more frames of invalid points, the stop condition should offer a setup that allows to continue the search in the next valid frames to be able to find other valid neighbors.

```
function RSM_KNN(K, q, δ̃)
    i ← 1
    Q ← (q, 0)
    while f_i(q) ∈ image do
        ι ← 0
        ν ← 0
        for all e ∈ f_i(q) do
            if is_valid(e) then
                ν ← ν + 1
                if Q.size() < K then
                    Q.push(e, D(e, q))
                    ι ← ι + 1
                else if Q.top > D(e, q) then
                    Q.pop()
                    Q.push(e, D(e, q))
                    ι ← ι + 1
                end if
            end if
        end for
        if ν > 0 then
            δ ← δ + (ν − ι)/ν
        else
            δ ← δ + 1
        end if
        if δ > δ̃ then break
        if ι > 0 then δ ← 0
        i ← i + 1
    end while
    return Q
end function
```

Figure 4.4: Pseudo-code for the RSM algorithm, kNN Search mode.

To meet all these goals, in my proposal two statistics are accumulated while exploring each frame: one is the number of valid candidates $\nu(f_i(q))$, i.e. all neighboring points with valid 3D coordinates, the other one is the number of inliers for the current search, $\iota(f_i(q))$. At the end of the exploration of each frame, the following stop condition is tested:

$$\delta(f_i(q)) = 1 - \frac{\nu(f_i(q)) - \iota(f_i(q))}{\nu(f_i(q))} > \tilde{\delta} \tag{4.1}$$

where $\tilde{\delta}$ is a user defined parameter. Intuitively, if the percentage of outliers (i.e., 1 minus the percentage of inliers) for the current frame is greater than a pre-defined threshold, the search is terminated.

To take into account the possibility that multiple frames are entirely composed of invalid points or outliers we allow the parameter $\tilde{\delta}$ to take integer

values greater than 1. In this case, the parameter counts the number of frames entirely composed of invalid points or outliers to be consecutively met before stopping the search. Every time such kind of frames are met, we set

$$\delta\left(f_i\left(q\right)\right) = \delta\left(f_{i-1}\left(q\right)\right) + 1 \tag{4.2}$$

and then check the stop condition.

The pseudo-code of the kNN Search can be found in Figure 4.4. For each query point, the algorithm keeps the discovered neighbors in a priority queue $\mathcal{Q}$, which holds the sorted result at the end of the search. $\mathcal{Q}$ is a min-priority queue in which the order is based on the distance between the query point and the element. Starting form the query point, I evaluated the frames in order of increasing distance, push each valid element of the frame in the priority queue if needed, and check if the termination criterion is met after processing every frame. After initialization of the data structures, the search starts from the first frame and continues until the stop criterion is met. In particular, the algorithm accumulates for each frame the number of valid examined candidates ν, as well as the number of those currently included in the nearest neighbor set, i.e. the inliers ι. With this value, it updates the percentage of outliers in the explored frames, δ, taking into account the previously exposed rules in case of fully invalid frames. When such percentage exceeds the user provided parameter $\tilde{\delta}$, the search ends.

The pseudo-code of the Radius Search can be found in Figure 4.5. For each query point, the algorithm keeps all the elements that have distance less than the radius parameter R. All the points are successively stored in a list $\mathcal{L}$, which is sorted at the end of the search to return the points in distance order. The overall structure of the algorithm is similar to the kNN Search, but due to the nature of the radius search, if a point is pushed into the list it is never removed because it definitively belongs to the final solution. For the same reason, the list is not kept sorted during the exploration, and is just sorted at the end, to save computation time.

The only step that may introduce approximations in the result in both algorithms is the stop criterion, the results being identical to the linear

```
function RSM_RADIUS(Radius, q, δ̃)
    i ← 1
    L ← (q, 0)
    while f_i(q) ∈ image do
        ι ← 0
        ν ← 0
        for all e ∈ f_i(q) do
            if is_valid(e) then
                ν ← ν + 1
                if D(q, e) < Radius then
                    L.push(e, D(e, q))
                    ι ← iota + 1
                end if
            end if
        end for
        if ν > 0 then
            δ ← δ + (ν − ι)/ν
        else
            δ ← δ + 1
        end if
        if δ > δ̃ then break
        if ι > 0 then δ ← 0
        i ← i + 1
    end while
    return L.sort()
end function
```

Figure 4.5: Pseudo-code for the RSM algorithm, Radius Search mode.

search one in the case of exploration of the whole range image. Therefore, the parameter $\tilde{\delta}$ trades off search accuracy for efficiency: since unnecessarily high accuracy negatively affects run-time performance, it is important to choose the right value of such parameters. In the Experimental results section I will analyze the sensitivity of the RSM algorithm to this parameter and provide guidelines on how to choose it appropriately.

Figure 4.6: Examples showing one object view of each of the four datasets used in my experiments.

4.2 Experimental results

In this section, I provide an experimental evaluation of the RSM method. The method has been implemented in C++, and it is here compared with the randomized kd-tree forest algorithm available in FLANN [53], as a representative of the state of the art for approximated NNS on point clouds, as well as with the *organized* NNS algorithm available in PCL [54], as a representative of approximated NNS algorithms for range images.

The comparison has been performed on a PC equipped with an Intel Xeon E312xx 2.00 GHz (4 cores) processor with 8Gb of RAM. I have compiled this framework under Visual Studio 2013 with optimization O2, and inline function expansion level set to Ob2. The evaluated algorithm don't includes any kind of parallelization, so the tests are always run on a single core.

The experiments were performed on four datasets composed of RGB-D images acquired with a Kinect dataset, recently proposed in literature and publicly available[1]. These datasets were originally proposed for the task of point cloud registration, and each of them includes different views of an object without the background: they are denoted here as *Frog*, *Mario*, *Squirrel* and *Duck*. A sample view for each dataset is shown in Figure 4.6. The measured average distance of each point from its nearest neighbor on

[1] http://www.vision.deis.unibo.it/lrf

this data is approximately 1 millimeter. Each dataset includes at least 13 range images. On each range image, 1000 query points have been randomly extracted from the available valid 3D point set, and the results averaged over this set.

I evaluate both the execution times and the accuracy achieved by the tested algorithms. To measure the accuracy, the NNS for each query point has been also carried out by a brute force algorithm performing an exhaustive investigation, and used as ground truth in order to count the number of correctly retrieved neighbors by each approximated NNS algorithm.

4.2.1 Parameter sensitivity analysis

The first experimental analysis I carried out is a sensitivity analysis with the goal of choosing a good value for parameter $\tilde{\delta}$, which is the main parameter the RSM algorithm relies on, in both kNN and Radius versions. In particular, as anticipated, such parameter trades-off accuracy for run time: the higher it is, the more precise the outcome of the search will be compared to that of an exhaustive search, but also the longer the whole process will take.

Figure 4.7 reports the charts relatively to the results, in terms of accuracy and efficiency, on the evaluated datasets where each curve is associated to different values of parameter $\tilde{\delta}$. The two top charts report results in the kNN Search case, while the two bottom charts are relative to the Radius Search case. In each case, the left chart measures the relative search accuracy with respect to the exhaustive search (number of correct neighbors found), while the right chart reports the average time to process 1000 query points in a range image. In the kNN Search case, the x axis reports increasing values of k, while in the Radius Search case, it reports increasing values of the radius (in meters), with values typically used in most applications of such 3D NNS algorithms. In particular, the tests were performed using, for the k parameter, a range of values between 2 and 150, while for the Radius parameter I have chosen a range from 0.005 to 0.030 meters.

From the charts related to the accuracy, RSM shows to be equivalent to

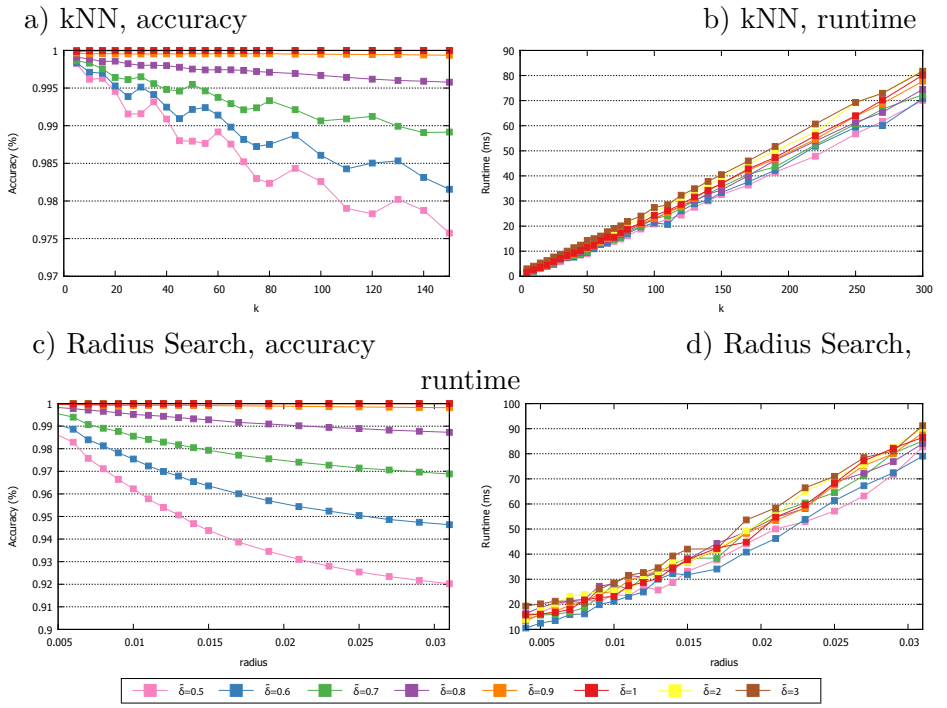

Figure 4.7: Sensitivity analysis for the parameter $\tilde{\delta}$. The accuracy is reported on the left charts, the runtime on the right ones. Top charts: kNN Search; bottom charts: Radius Search. The scale of the y axis in a) and c) has been expanded for better visualization of the results.

the exhaustive search if the value of $\tilde{\delta}$ is greater than or equal to 1. Yet, the drop in performance is limited even if $\tilde{\delta}$ is set to low values: the worst result I get is to retrieve 92% of the real neighbors when using $\tilde{\delta} = 0.5$ in the radius search. This result confirms the intuition that a radial search can be a good exploration pattern for NNS and shows that the statistics used to define the stop condition are able to limit the explored neighbors to the most interesting ones. At the same time, looking at the reported

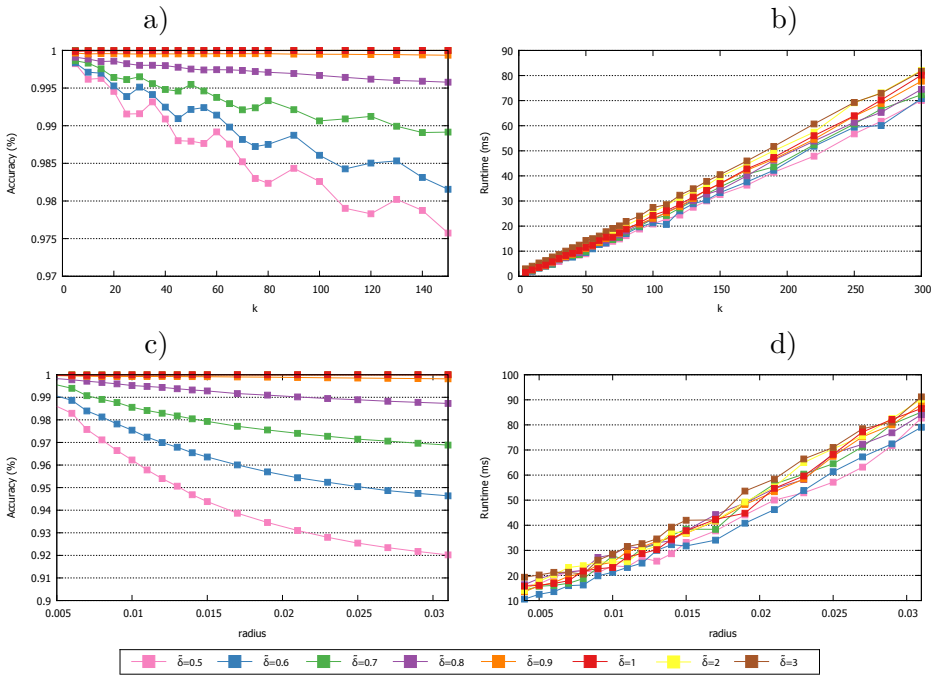

Figure 4.8: Accuracy (a,c) and runtime (b,d) reported by RSM, FLANN and *Organized* methods on the evaluated dataset. The scale of the y axis in a) and c) has been expanded for better visualization of the results.

runtime, as expected the lower the value of the parameter, the faster the overall efficiency. The gap between different approximation levels increases the larger k or radius get. As the gap in runtime is limited between the different choices of $\tilde{\delta}$, in the remainder of the experimental Section, I will employ the value of $\tilde{\delta} = 1$, given that it yields the highest efficiency among those reporting perfect accuracy.

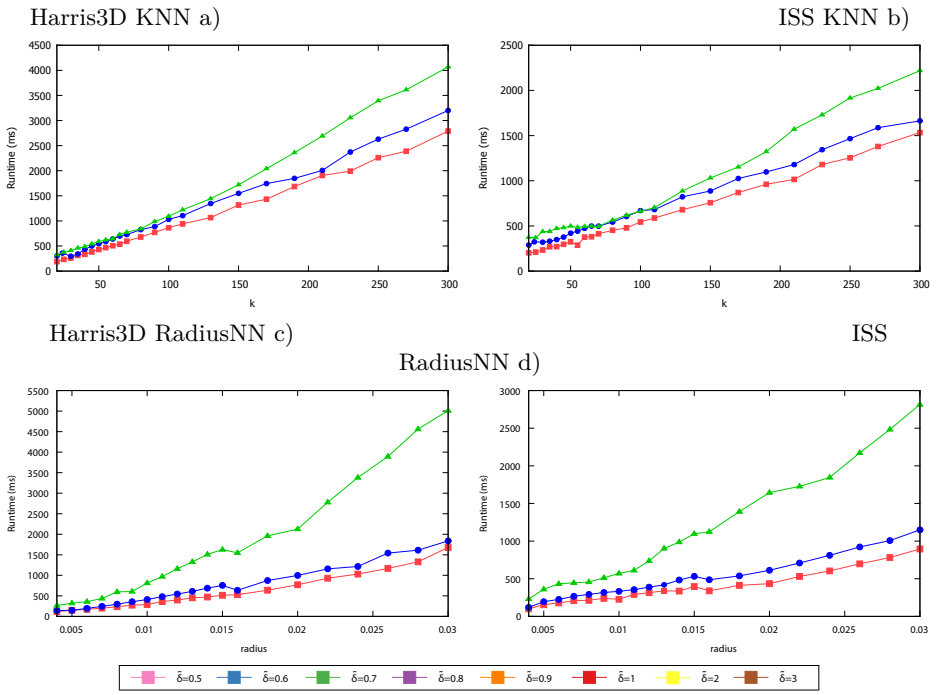

Figure 4.9: Runtime reported, respectively, by the Harris3D detector (a,c) and the ISS detector (b,d) when employing, respectively, RSM, FLANN and *Organized* methods for the NNS.

4.2.2 Comparison with the state of the art

Figure 4.8 shows the results in term of accuracy and runtime reported by the evaluated methods (RSM, FLANN and *Organized*) on the test dataset. As before, the top charts report the kNN Search case, while the bottom charts are relative to the Radius Search case, each chart showing the results at increasing values of the k and the radius parameter. In each case, the left chart measures the relative search accuracy with respect to the exhaustive

search (number of correct neighbors found), while the right chart reports the average runtime over all the images of the datasets when processing 1000 query points on each range image.

Interestingly, in terms of accuracy all methods report, in both Search cases, a negligible loss of accuracy with respect to the exhaustive investigation. As far as the runtime is concerned, FLANN shows to scale worse than the other methods when k or the radius increases. Nevertheless, when just a few neighbors are sought, it turns out faster than *Organized*, which is surprising given that *Organized* has been specifically conceived for range images. RSM is consistently the fastest for all k and radii. This confirms that taking advantage of the structure inherent to range images can improve NNS efficiency with respect to using a general purpose solution like FLANN, and that a more natural exploration pattern like the radial one deployed by RSM can explore more promising areas of the image first and terminate the search earlier than the raster-scan search deployed by *Organized*.

4.2.3 Relevance of NNS in keypoint detection

To complement previous results, I have compared FLANN, *Organized* and RSM when used within a real and widely deployed application of the NNS problem such as 3D keypoint detection. As for the datasets, I have used the same data used in the previous experiments. In this case, results have been measured in terms of overall runtime of the whole detection process, so to measure out how much the computational advantage brought in by RSM impact in terms of the whole application. In addition, I have also measured the accuracy of the NNS in terms of the final application, i.e. by computing the relative repeatability [56] between the extracted 3D keypoints on pairs of overlapping views. To perform the repeatability evaluation, I have exploited the registration ground truth available with the dataset, that provides the 3D translation and 3D rotation registering each view into all other overlapping ones.

As for the choice of the 3D detectors, based on the analysis in [56] I have

selected the Intrinsic Shape Signatures (ISS) detector [57], which provides a good trade-off between repeatability, distinctiveness and computational efficiency. In addition, I have also included in the comparison the Harris3D detector[54], which is an extension of the Harris corner detector[58] to the 3D case. For both detectors, I have used the available implementation in PCL[54]. As typically done by most 3D keypoint detectors[56], both methods rely on a NNS at each point of the range image to compute a local saliency: the extrema of such saliency are then used to localize distinctive keypoints. I have appropriately modified the code so to use, for the NNS, one method among RSM, *Organized* and FLANN, which have been tested both in the kNN Search as well as in the Radius Search version.

Figure 4.9 shows the results in terms of overall detection time for Harris3D (left charts) and ISS (right charts), both in the kNN Search (top charts) and in the Radius Search (bottom charts) case. These charts show that by deploying RSM, the overall time required to perform keypoints detection can be clearly reduced, especially when analyzing structures at larger scales, i.e. those defined by larger k or radii, and confirm the practical importance of designing efficient methods to solve the NNS in 3D data.

4.2.4 Relevance of NNS in descriptor computation

In Figure 4.10 are shown the results relative to the descriptors tests. On the vertical axis are shown the perfomances relative to the mean time for a descriptor computation. As in the previous tests on the detectors the runtime is evaluated on the Kinect datasets (Figure 4.6), the keypoints used are obtained using ISS detectors. Since PCL implements only radius searchability for each descriptor algorithm only the RadiusNN search is compared in the tests. In the figures is possible to see how much the performances are influenced by the neighborhood search. First of all using Fast Point Feature Histograms (FPFH) [32] both RSM and Organized are faster than FLANN (4.10.a), but since the main computation is spent around the non-search part (the runtime axis is expressed in thousands of milliseconds) they are very close in the diagram, more explicative results were

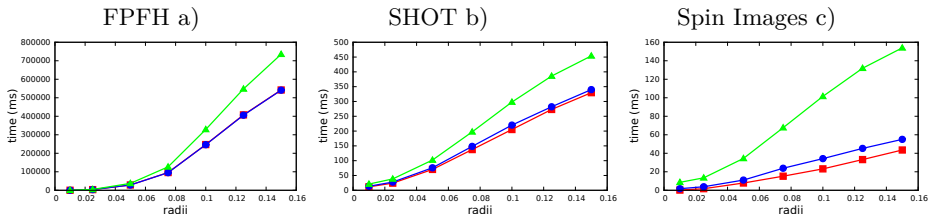

Figure 4.10: Mean runtime reported for each keypoint descriptor evaluation, respectively we have FPFH (a), SHOT (b) and Spin Images (c), every descriptor was evaluated with RadiusNN search on a ISS type keypoint.

obtained using Signature of Histograms of OrienTations (SHOT) [22] and Spin Images [19] (respectively 4.10.b 4.10.c), looking closer is possible to see the improvement given by RSM over both Organized and FLANN methods. Furthermore, comparing descriptors results is impossible to notice differences between a description obtained using FLANN algorithm compared to one obtained using RSM or Organized.

4.3 Concluding remarks

In this work, I have proposed a new method for NNS on range images, dubbed RSM, which proves to be faster than available algorithms for NNS on this kind of organized data, while preserving the same level of accuracy as the currently employed NNS methods for points clouds and range images. In particular, RSM is able to leverage on the organized structure of range images and on effective stop conditions applied while exploring the neighborhood radially from the query point to terminate the search process as soon as the currently probed locations do not seem to contain additional nearest neighbors. RSM proved to provide computational savings both in the Radius Search as well as in the kNN Search case. Furthermore, the method proved to be particularly effective for speeding up 3D keypoint detection

and description without reducing the quality of the results in terms of repeatability and distinctiveness.

Chapter 5

SHOT Descriptor on FPGA

In this work, I present a new FPGA circuit that computes SHOT 3D descriptor starting from the reference key point and from the point cloud of the model. The circuit was implemented using Xilinx High Level Synthesis language. Thanks to the interpolation parallelization we are able to describe a point in linear time with the size of the neighborhood, the structure is pipelined so that one neighbor is processed every clock cycle once the pipeline is full. Each interpolation is performed separately and the resultant weight is updated in a BRAM without any delay. Final synthesis reports a clock frequency higher than 100 MHz. Results are compared against the CPU/GPU implementations showing performances that outperform CPU and are similar to the GPU one with a consistent power consumption difference.

Progress in range sensor development now allows us to deliver high definition range maps at high frame rates. Each process that needs to be applied to this range map stream (e.g. key points extraction and key points description) has to satisfy a precise performance constraint in order to avoid bottleneck in the computation. To execute these algorithms on portable devices there is an additional constraint - low power.

The principal problem to address during 3D scene reconstruction is to find the correct rotation and translation matrix among many views obtained through passive or active range sensors (also called depth sensors).

These views are generally stored as 3D point clouds, lists of tuples containing a 3D point (x, y, z) and an optional colour. Aligning point clouds is not easy and it is related to the Simultaneous Localization and Mapping (SLAM) problem, but, once that two disparity maps are correctly aligned, the relative positions of the sensors could be easily computed.

To align point clouds, it is possible to use techniques based on matches between points in the cloud. To make a correct match, it is necessary to compute a unique and descriptive representation for each point that is invariant to the sensor pose. For this reason, if this representation remains unambiguous among the views, it is said to be the "description" of the point. Specifically speaking, the description (or descriptor) of the point is stored as a multidimensional feature vector. The similarity between two descriptions is easily computed as the distance in the features space of the descriptors.

Over the years many algorithms for the 3D points description have been proposed. These techniques try to describe a point p using its neighborhood N_p; for each p it is possible to define a neighborhood N_p within a maximum distance (or radius) R as:

$$N_p = \{q \in PointCloud \mid dist(p, q) < R\}$$

where $dist(p, q)$ is defined as the common Euclidean distance. The concept of neighborhood will be used later as a way to reduce the computational load. These algorithms are, of course, computationally expensive since a unique and robust description (invariant to the sensor pose) needs, generally, a large set of neighboring points. The number of operations required is proportional to the point cloud size ($|PointCloud|$) and to the size of the neighborhood ($|N_p|$), leading to a complexity of $\mathcal{O}(|PointCloud| \times |N_p|)$. So the larger the neighborhood the higher the complexity. Furthermore, the description of as many points as possible could be useful in certain

applications like the 3D object recognition and localization, but the action of describing every single point is often not feasible. For a 640×480 pixel depth sensor (already small in 2017), 3×10^5 point descriptors must be computed. Processing time can be reduced by selecting a small set of key points of maximum information (or entropy), usually found on scene edges and corners. One problem is that noise or different sensor poses may lead to key points in one view being misinterpreted or hidden from the other. So there is a trade-off between storage requirements, accuracy and processing speed.

Because of their high computational cost, the key point extractions and descriptions are not suitable for embedded applications requiring real time performance and low power usage. By now, modern desktop CPUs struggle to compute a large number of descriptors of a 3D point cloud video. A possible solution is the parallelization of the processing employing GPU, but a battery dependent application could hardly use such a power consuming device, e.g. a drone should consume an important part of the battery power just to run the GPU algorithm that allows to align 3D point clouds obtained using a depth sensor, when trying to reconstruct the environment and localizing itself in the real word. The only way to obtain an high performance system with a low power consumption and acceptable quality is to use a dedicated circuit that can be produced using devices like FPGA and ASIC, we show later that our implementation consumes nor more than a quarter of Watt, making it far more efficient than any graphic card available on the desktop/laptop market.

Many important feature descriptors have been implemented on FPGAs, e.g. Chang et al's SIFT [59] and Krajnik et al's SURF [60]. We verified our design in simulation using, as a target, a cheap and power efficient FPGA system: the Xilinx Zynq-7000 (part number XC7Z020CLG484) [61]. This particular chip contains, in addition to the FPGA fabric, an ARM Cortex-A9 dual core. The cores can be used to support the FPGA computation or to arbitrate the various modules present in the circuits.

Generally, as described by Tombari et al [22], 3D descriptors are divided in

two categories, named signatures and histograms techniques. The techniques in the first category try to describe p in N_p using a Local Reference Frame (LRF). A LRF is a new coordinate system that is used as a reference to encode the geometry of the neighborhood that is invariant to the sensor pose, unique and repeatable [22]. The feature vector is built starting from a series of conditions on the neighbors geometric properties, if one of the property has a value that lies on the boundaries of such conditions a small perturbation to the LRF could change dramatically the feature vector, as in the case of the 3D version of Knopp et al's SURF descriptor [62]. On the other hand, histogram techniques try to describe a point counting the number of time in which a specific property (curvatures, angles, distance, etc.) occurs in the neighborhood, as an example we have Rusu et al's PFH [31] and FPFH [32] and Zhong et al's ISS [57].

Salti et al's SHOT descriptor [63] combines the advantages of both signature and histogram techniques it uses an LRF coupled with a set of local histograms. The descriptor is a signature formed from a set of histograms. As described in [63] combining both approaches gives more robustness, for this reason we have developed a module to calculate this descriptor. There are two versions of SHOT, the first one uses just the shape of the scene, while the other uses both scene and color information. In this work we present a circuit for the first version.

For this work, we decided to use the High Level Synthesis programming (HLS) language made by Xilinx. Thanks to this language a lot of implementation details are hidden to the programmer operations like square-root and trigonometrical functions are already implemented and can be used as easily as a function call. Thus we can focus on the correctness of the algorithm, leaving all of the details to the compiler.

The sections describing the architecture are are structured in the following way:

- In the first part I describe in details the descriptor;
- In the second part I introduce the circuit;

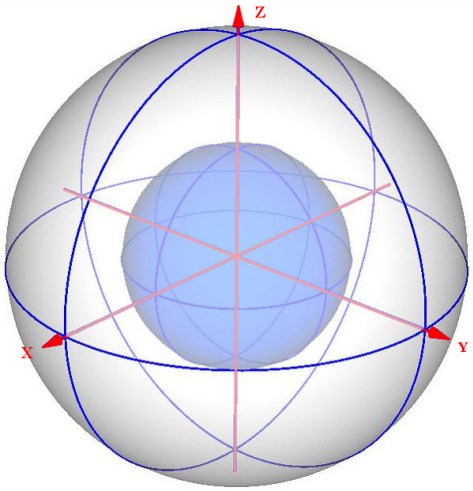

Figure 5.1: The spherical signature of SHOT descriptor with highlighted local reference frame (LRF) in read. For clarity reason in the figure are shown only four horizontal sections. Source:[63]

- In the last part I compare the CPU/GPU implementation with ours circuits.

5.1 SHOT descriptor

SHOT takes advantages of both signature and histograms techniques. The description procedure takes as input: the key point to describe, its neighborhood and the normals of the neighboring points with respect to the surface. The procedure is divided in phases. The first phase computes normals over the whole image and it is used in every subsequent key point calculation. In the second phase, the neighbors of the key point are selected. In the third phase, starting from the neighborhood, the LRF is computed and the description could start. In the fourth phase the feature vector is obtained,

while in the last phase the descriptor is normalized using L2 norm. The process is now explained more in details.

Estimating the normals of a plane tangent to the surface is an approximation of the true normals, this problem become a least-square plane fitting estimation problem [64]. Let p be the point of which we want to obtain the normal, a distance radius R is chosen to define the maximum distance of the neighborhood N_p. We define for each point $q \in N_p$ the distance between p and q as d_{pq}. The analysis of the eigenvectors of the following covariance matrix C gives the normal to the point q we want to obtain:

$$C = \frac{1}{|N_p|} \sum_{q \in N_p} (q-p)(q-p)^T$$

Where $(q-p)(q-p)^T$ is commonly known as the outer product between two vectors and give as result a matrix. A very similar procedure is used to get the LRF of a key point. Let k be the key point we want to describe, then the LRF of k is computed using the eigenvectors of the following covariance matrix:

$$M = \frac{1}{\sum_{q \in N_k}(R - d_{kq})} \sum_{q \in N_k} (R - d_{kq})(q-k)(q-k)^T$$

It should be noticed that each point in the neighborhood contributes to the matrix proportionally to the distance from the key point. The eigenvectors of this matrix are ordered according to the respective eigenvalue. The eigenvector with the highest eigenvalue will be used as the LRF X axis, while the second will be used as Y and the Z axes will be computed as the cross product of the first two.

This new coordinate system, the LRF, is used to superimpose a spherical grid which has as center the key point Fig: 5.1. The grid divides in different spherical sectors the neighborhood, and to each sector of the grid is associated a histogram that will encode information about the neighbors relative position.

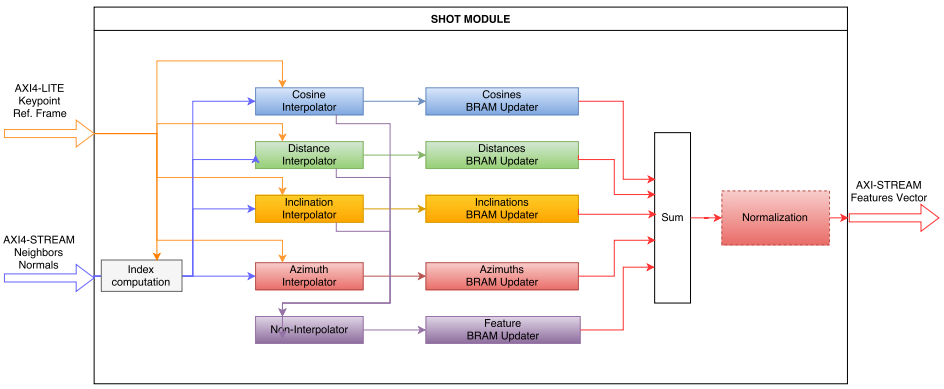

Figure 5.2: FPGA-SHOT Architecture.

The number of bins and sectors used in the descriptor was chosen based on Salti et al's finding that 32 segments with an 11 bin histogram in each was a reasonable balance between feature size and descriptiveness [63]. These parameters give a feature vector of size 352. As shown in Fig: 5.1 the spherical grid has two elevation levels (up and down the equator), eight horizontal sections (for clarity purpose in the figure are shown only four) and two distance ranges (the inner and outer sphere).

Now that a valid LRF is available, the computation of the feature vector can begin. The position of every point is checked against the grid, each sector is associated to an histogram, whenever a point lies in a specific sector the relative histogram will be updated. Each histogram has a precise number of bins, a bin represents the quantization of the interval of the cosine between the q normal and the Z axes of the LRF. Updating just one bin per point will not give a robust description of the point k and problems arise when points lie near the boundaries of a sector. In this latter case, a little perturbation in the LRF will cause a dramatically change in the feature vector. To avoid this, histogram update rule is modified, for each point in the neighborhood four further histogram bins are updated in a weighted manner according to the distance from the relative sector boundaries. Let b

be the index of a histogram bin in the s sector, the multiple update, named quadrilinear interpolation, is performed on:

- the bin $b \pm 1$ in the same sector (cosine interpolation)
- the bin b in the adjacent sector on the horizontal plane of s (azimuth interpolation)
- the bin b in the adjacent inner or outer sector of s (distance interpolation)
- the bin b in the adjacent sector of s on the elevation (elevation interpolation)

Each bin is updated in a weighted manner according to the property of the processed neighbor. In particular, each bin is incremented by a weight of $1 - d$ for each interpolator. As for the same sector histogram, d is the distance of the current cosine from the central quantized value of the bin. As for elevation and azimuth, d is the angular distance of the neighbor from the central value of the volume. Finally, along the radial dimension, d is the Euclidean distance of the neighbor from the central value of the volume. The value of d is always measured in units of the histogram. At the end of the interpolation, the original bin b is updated by the sum of the slacks given by each update.

This approach makes the descriptor robust to the LRF perturbation. In fact, when a neighbor switches sector it will cause no great differences in the feature vector because the histogram of both sectors is anyway updated. Each operation has a different level of computational load, e.g. for the distance interpolation a square root operation is needed, while for the azimuth interpolation the atan2 function must be computed.

The normalization of the feature vector is necessary to make the descriptor invariant to the scale of the object, for this reason the feature vector is normalized making it's magnitude equals to one. Let v be the feature vector

obtained after the quadrilinear interpolation and $\hat{v}$ the normalized vector we have:

$$\hat{v} = \frac{v}{||v||}$$

where $||v||$ is the L2 norm of vector v.

A commonly used library for applications that use point clouds to model 3D data is Point Cloud Library (PCL) [54], we used it to run the open source implementation of SHOT and use the results as gold reference for our implementation.

5.1.1 Descriptor discussion

In Fig. 5.3 is shown the time taken by each phase of the computation as measured by Palossi et al's in [65, 66]. The most time consuming step is the histograms and interpolation update. We can also observe how the LRF computation and the neighbors search is critical. The normal computation is performed once that point cloud is acquired, since there is no data dependencies between the frames, while the key point of one frame are processed, the normals of the next one can be computed in parallel. The main concern of this work is to correctly implement the histogram and interpolation phases. Others like Bravo et al's have described

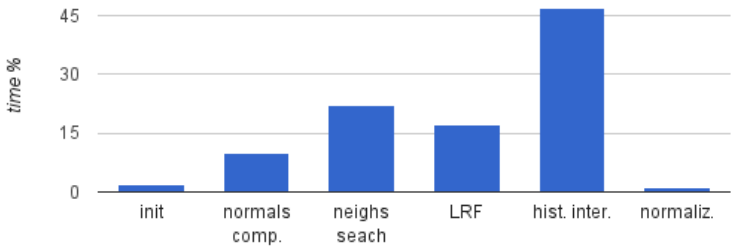

Figure 5.3: Relative times for each computation step.

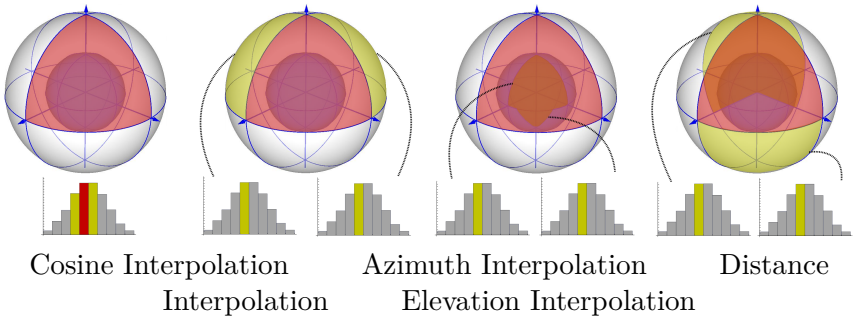

Figure 5.4: Quadrilinear interpolation: taking as reference bin and sector marked in red, the bins and sector marked in yellow are the bins that are considered during the interpolation, the rules applied in the interpolator are used to discriminate between them.

Jacobi matrix eigenvector computation needed for the normals and LRF computation [67, 68], it is beyond the purpose of this paper further discuss about these two phases. Furthermore, no neighbor search is required since it is easier and faster to pass a whole region of interest of the point cloud that encompasses the neighborhood size. The more the region of interest is accurate (smaller) the more rapid the computation. The extraction of the region of interest could be simply based on the resolution of the point cloud; this method is implemented in PCL [54] and it is implemented in the class OrganizedNeighbor. Thus the neighborhood search is treated like a mere check on the distance while the interpolation and normalization steps are fully implemented.

5.2 Implementation Details

In the original software implementation of the algorithm there is a massive use of floating point arithmetic. To avoid such a computational heavy structure we decided to utilize fixed point arithmetic. This choice makes the job easier for the HLS compiler, in fact, we utilized HLS libraries optimized

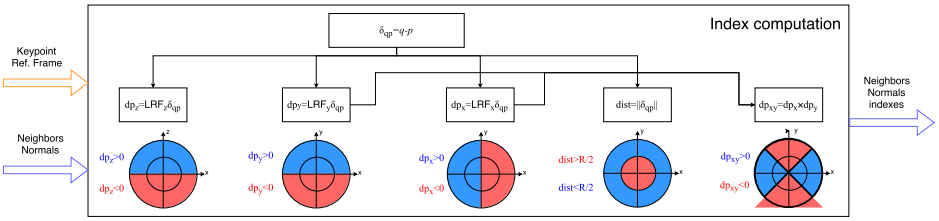

Figure 5.5: Index computation module: the 32 sectors of the sphere are indexed using a conditional approach, five bit register is required, each bit is compute as shown in the figure and packed in the following way $S = \{dp_z > 0, dp_y > 0, dp_x > 0, dist > R/2, dp_{xy} > 0\}$

for this kind of operation, although this will imply some loss in accuracy.

Using a zynq-7000 the heavy arithmetical operation like multiplications and divisions are automatically assigned to DSP48E1 [69]. This kind of DSP supports multiplication up to 25×18 bits, to avoid an overuse of the few DSPs present on the board, and to keep the quality of the results acceptable, we set the size of the fixed point register to 36 bits, in this way for each fixed point multiplication exactly 2 DSP48E1 DSP are utilized. Although this may seems restrictive, thanks to the development tools it is easy to re-target the whole project to another platform. In this way the programmer can write a code as general as possible keeping away the risk of an architecture dependent implementation.

The whole circuit is structured as an Intellectual Property (IP), two input source must be provided, the first one should provide the key point and its LRF once for each processing phase, while the other should provide one neighbor point and its normal per each clock cycle. Once that every neighbor has been processed the output of the IP is ready to be fetched as the actual descriptor of the key point. The Input and output port implements AMBA AXI4 Protocol [70], an on-chip interconnect standard for the connection of blocks in SoC designs. In particular the key point and LRF port is defined as an AXI4-LITE port with a data width of $36 \times 3 = 108$ bits, four values

(each with 3 components) are expected by this port, one key point and three vectors representing the LRF. This port can be driven using a simple controller implementing AXI4-LITE protocol, using a zynq-7000 is possible to interface the ARM processor directly to the input port. The second input port is of AXI4-STREAM type, this protocol allows the fast transmission of data between a RAM like memory and a circuit component connected to the BUS, using a DMA is possible to feed the module with the required timing. Both point and normal have the width of $36 \times 3 = 108$ bits, this determines a bus width of 216 bit. The output port is also of AXI4-STREAM type, once that the computation stops 352 values are ready to be read from the system. Every value is 36 bits wide, and using a DMA to write to the memory is possible to get the descriptor after 352 clock cycles. Thanks to the Zynq architecture it is possible to connect the module to the bus without using external pins of the FPGA. Both microprocessor and programmable logic reside on the same chip and a finite number of AXI interconnect is available between them.

The principal goal of the project is to obtain a circuit capable of processing points and normals in a fully pipelined manner. Waiting a little latency of 165 clock cycles, we obtained a constant complexity ($\mathcal{O}(1)$) for the single neighbor processing, so, to compute the whole descriptor a complexity of $\mathcal{O}(|N_p|)$ is required. In Fig: 5.2 is shown the proposed circuit. The interpolation is computed in separated sub-modules which are directly connected to Block RAM updaters. When the descriptor is computed, a final stage of normalization is applied to report the feature values in a range between zero and one.

In order to perform interpolation (shown in Fig: 5.4) it is necessary to found the index of the sector in which the neighbor point lies. The block diagram of the index computation module is shown in Fig: 5.5. Since there are 32 sectors, a five bit register is needed. We denote with δ_{qk} the difference vector between point q and key point p, with s the sector index, with b the index of the histogram bin that is updated with the non-interpolation technique and with LRF_x, LRF_y, LRF_z respectively the X, Y and Z axis of the LRF. The index computation is performed in the following way:

- $dp_z = LRF_z \cdot \delta_{qk}$
- $dp_y = LRF_y \cdot \delta_{qk}$
- $dp_x = LRF_x \cdot \delta_{qk}$
- $dist = ||\delta_{qk}||$
- $dp_{xy} = dp_x \times dp_y$
- $S = \{dp_z > 0, dp_y > 0, dp_x > 0, dist > R/2, dp_{xy} > 0\}$

Where S is obtained as the combination of the truth value of the conditions. In this way it is easier to specify a sector using the bit of the index register i.e. if we have sector x and we want to select the sector that lies above or below in the elevation sense, the selection is as easy as $x \pm 16$ since the elevation bit (dp_x) is the fourth in the register.

Each sectors stores an eleven bins histogram, to choose the right bin it is necessary to compute the consine of the angle between q normal and the LRF_z. The quantization of this interval gives a bin index b. Linearizing the whole descriptor into an array of 352 bins the non-interpolator module should update the value at position $B = b + s \times 11$.

The index computation module takes as input the LRF system, the keypoint k, a point q and its normal for every clock cycle. In output of the system there are: the feature vector F index B and the point q and its normal. In the following paragraphs each interpolator is described to explain the different iteration latencies among the modules, the values are reported considering the index module in cascade to the interpolators.

5.2.1 cosine interpolation

This is by far the least complex module. It takes only 28 clock cycles of latency to fill its pipeline. It checks which adjacent bin to B should be updated by checking the cosine between the normal of point q and the LRF_z.

If this value lies on the right side of the mean of bin B, then $B+1$ bin will be updated, $B-1$ otherwise. A precise value of the cosine is needed since the correct bin should updated.

```
cos_z = dot_product(normal_q*LRF_z)
if(cos_z>0) F[B+1] += cos_z
else F[B-1] += cos_z
```

basicstyle

5.2.2 distance interpolation

Since the actual distance is needed, a square root operation is synthesized using *math.h* library from HLS, in this case the operation is implemented using an iterative method which is unrolled by the compiler to get fully functional pipeline. This distance is compared with 1/4 and 3/4 of the radius distance. This comparison discriminate between the inner, the outer and the sector on the opposite side. 57 clock cycles are used to process a single neighbor in a fully pipeline way.

```
r = (distance - (R*3/4)) / (R/2)
if(distance>R/2)
    if(distance>R*3/4) F[B-(2*11)] += r
else
    if(distance<R*1/4) F[B+(2*11)] += r
```

basicstyle

5.2.3 elevation interpolation

For this interpolator the *arccosine* function is used to compute the elevation between the LRF_z and δ_{qk}. The result of this operation is compared against four values, the value in radians of the angles 45°, 90°, 135° and 180°. In this way it is possible to discriminate between the vertically adjacent sectors

and their histograms. The interpolation function is approximated using a Lagrange polynomial which has a worst case approximation error of 0.18 radians:

$$acos(x) \approx -0.6981[\dots]x^3 - 0.8726[\dots]x + 1.5707[\dots]$$

Again we take advantage of HLS which takes care to implement the polynomial computation using DSPs and keeping the structure pipelined. This interpolator has the highest latency of 165 clock cycles, once that every neighbor has been processed the other interpolators are required to wait for this module, the programmer do not takes care of these details thanks to the HSL compiler that produce a hardware description which is correctly translated and synchronized.

```
x = acos(elevation)
if(x>90)
    if(x<135) F[B+(16*11)] += (x-135)/90
else
    if(x>45) F[B-(16*11)] += (x-45)/90
```

basicstyle

5.2.4 azimuth interpolation

In this sub-module the most complex operation is computed, the atan2 function is used to find which one of the horizontal adjacent sector must be updated, the arguments of atan2 are the dot product between the δ_{qk} and LRF_x and the dot product between δ_{qk} and LRF_y. The function is then approximated using the atan2 definition:

$$\text{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y > 0, \\ -\frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y < 0, \\ \arctan\left(\frac{y}{x}\right) \pm \pi & \text{if } x < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

and using the following polynomial minimax approximation to compute the arctangent function:

$$arctan(x) \approx -0.0465x^7 + 0.15931x^5 - 0.32762x^3 + x$$

The latency of this module is 130 clock cycles.

```
x = arctan2(dp_x,dp_y)
if(x>0)
    F[B+(1*11)] += (x-135)/90
else
    F[B-(1*11)] += (x-45)/90
```

basicstyle

5.2.5 Why not CORDIC?

The most used trigonometric approximator in FPGA field is the CORDIC algorithm [71]. In order to use CORDIC in this project we output values that are 36 bits wide; all these bits are necessary to maintain a good degree of accuracy. Furthermore, in our case the pipelined version must be used (e.g. the one proposed in [72]) implying the use of a 36 levels CORDIC. Furthermore, since each trigonometric function is applied to a different angle, is not possible to use only one CORDIC, but one for each trigonometric function is required. To avoid the use of further area into the already congested FPGA fabric we decided to use DSP to approximate polynomial

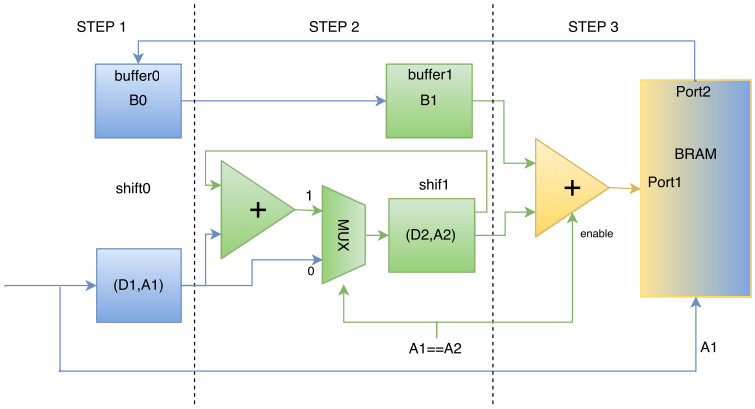

Figure 5.6: BRAM Updater. At each clock cycle a couple (value,address) is presented to the circuit, if the address of the previous couple A2 is the same as the present address A1 the data D1 and D2 are summed. This avoid to lost the previous data D2 lost.

interpolation functions. In particular, the 36 bits size has been chosen to perfectly fit two DSP48E1 for each multiplication.

5.3 BRAM updater

Each interpolator module gives as output a weight that must be added to the final feature vector. Each interpolator fill a memory space of $36 \times 11 \times 32 = 12672$ bits. This is due to the fact that there are 11 histograms for each one of the 32 sectors. Since there are five modules that operate on the feature vector, five separate memories should be instantiated, with a total amount of $12672 * 5 = 63360$ bits to store, to avoid the waste of too many flip flop we decide to implement each memory as a BRAM, in this case in fact at most five BRAMs are required.

Since our weights must be used to increment the value already present in memory, it is mandatory to wait a one clock iteration latency to read the

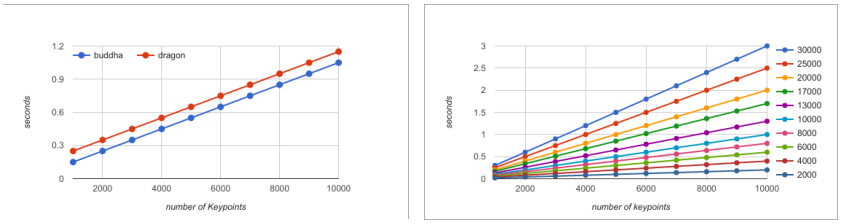

Figure 5.7: On the left execution times of the GPU implementation on the two color-less point clouds as shown in [65], on the right performances expressed for various set of key points using different size of neighborhood (the number of points is computed using the radius search parameter).

BRAM and then another clock cycle to write the updated value. Unfortunately this structure is not suitable for a pipelined circuit since a double update could occur leading to stalls in the whole pipeline: if imagining the case in which two values that update the same location arrive in adjacent clock time, every time a new value arrives the BRAM is read and then written. In this case, using a pipelined circuit, an error occurs because while the first value issues a write request to the BRAM, the second, in the same clock cycle, issues a read request. As final result, the second value updates the old location content erasing the previous one. To avoid this overwrite, a separated sub-module has been built for the memory update (Fig: 5.6), we used a true-dual-port BRAM in order to read and write in the same clock cycle in different location.

Each updater requests a value pair per clock cycle, the pair is composed by a 9 bits wide address (used to choose between 352 locations) and the 36 bits wide weight that should be added to the final descriptor. The circuit is divided in three pipelined steps, in the first one a couple (value,address) is stored in a register (`shift0`) and a BRAM read request is issued to the memory causing the write of `buffer0`, in the second step the value present in `buffer0` is forwarded to `buffer1`, simultaneously the content of both `shift` registers is checked; if the addresses are the same no memory write happens

a) b) c)

Figure 5.8: The three models used as dataset for the performance comparisons.

and the `shift1` is incremented by the value of `shift0`, otherwise the BRAM content is updated. This mechanism resolves the erasing increment retaining the weight values when a double (or more) update happens.

Adding extra buffering allows the latency of the interpolator modules to be matched. Thanks to this stratagem, various parts of the circuit are kept synchronized allowing a fully pipelined structure. The whole structure took 165 clock iterations and, after this interval, a weight is available for each BRAM updater at every clock cycle. In this way, a point in the neighborhood and its normal could be sent at every clock interval.

5.4 Sum and normalization

Once that each interpolator has finished its work, the weights generated from each sub-module are sent to the sum module. This module read each location of the BRAMs present in the updaters. The final result is a BRAM containing a non normalized feature vector, the sum operations are done in parallel thanks to a pipelined tree structure. In this way the whole process has a constant complexity of $\mathcal{O}(352) \sim \mathcal{O}(1)$.

Max Frequency Target	100 MHz
SLICE Occupation	6881
Look Up Table	20065/53200 (37.7%)
Flip Flop	23809/106400 (22.3%)
DSP	144/220 (65.4%)
number of BRAM	5/280 (1.8%)
SRL	1822
Total On-Chip Power	0.27 W
Worst Path	8.6 ns

Table 5.1: Final synthesis results for FPGA.

During this process an accumulator variable sums the square of each value of the final feature vector, at the end of the process a square root operation is performed to obtain the L2 norm of the feature vector. Finally, the L2 norm is used to divide each component of the feature vector normalizing the result.

5.5 Results and comparison

We synthesized our circuit using a cheap device like the zynq-7000 xc7z020 present on a ZedBoard [73]. In Tab: 5.1 is shown the synthesis report for our circuit. The device is half empty and more modules, like the eigenvalue decomposition for the normals and LRF computation, could be easily placed in the design. A single clock is used for the whole module, the synchronization of the various sub-modules is managed by HLS compiler, the final result after synthesis phase report a maximum frequency higher than 100MHz.

In 5.9 is shown the speed-up between the FPGA and CPU implementation, the code was run on q Intel Xeon 2 GHz Quadcore.

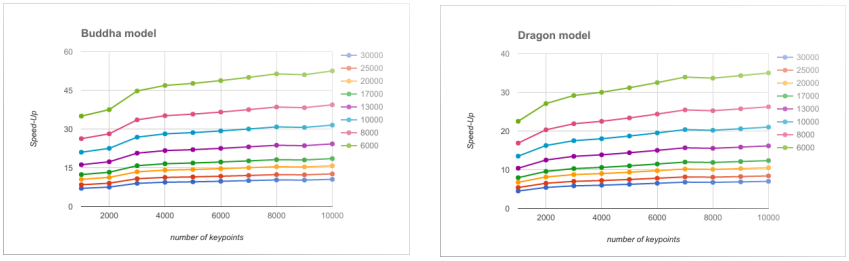

Figure 5.9: Speed-Up obtained comparing the PCL CPU implementation and the FPGA results.

5.5.1 Accuracy comparison

We compared the accuracy of our implementation against the SHOT implementation that can be found in the PCL [54] library. We ran our circuit over all points of the Mario views in the Bologna dataset Fig. 5.8.c. Using a fixed point arithmetic has implied an acceptable loss in the accuracy of the description. Running the experiment we compared the accuracy of the circuit and the PCL implementation using the relative error formula averaged on the whole feature descriptor, let $F_{fix} = \{f_1, \ldots, f_n\}$ and $F_{float} = \{g_1, \ldots, g_n\}$ be respectively the descriptor obtained by our fixed point circuit and the floating point result of PCL, the averaged relative error is defined as:

$$E = \frac{1}{n} \sum_{f_i \in F_{fix}, g_i \in F_{float}} \frac{|f_i - g_i|}{g_i}$$

Our circuit has obtained an average error of 10^{-4} all over the Mario models using 36 bit registers.

5.5.2 Performance comparison

We have compared our performance against the GPU version of the descriptor [65, 66], those results were obtained using a Tesla C2075 GPU. Comparisons

are not simple and of course the FPGA circuit is far more power efficient than the GPU implementation. Concerning the speed performance, in Fig: 5.7.a is shown the computation time of the feature vector for different set sizes of key points. The key points are extracted from 2 point clouds (Happy Buddha, Dragon, composed of, respectively, 32328 and 100250 points Fig: 5.8.a, 5.8.b) taken from the Stanford 3D Scanning Repository. We have chosen these point clouds because they do not have color information since our circuit is based on the colorless version of SHOT. In Fig: 5.7.b is shown how the performances change using different numbers of key points and different sizes of the neighborhood. Our circuit has a total pipeline latency of 165 clock cycles, after this interval the computation became directly proportional to the number of neighbors passed to the modules ($\mathcal{O}(|N_p|)$), thus if we use a neighborhood of 1000 points we only need to wait 1165 clock cycles before the computation ends. This result scale linearly with size of the key points set. Using the radius of the neighborhood search R is possible to define a window around the key point which encompass for sure any neighbor that must be included in the computation, using this window as a reference is possible to greatly reduce the number of points to send to the circuit. Thus our circuit is not influenced by the size of the point cloud as long as it is possible to define an upper bound to the number of neighbors to check. No information about the neighborhood size is provided in [65], but our results show that the two implementations have similar performance and, as previously said, our circuit is far more cheaper in terms of power consumption.

5.6 Future works

We presented an FPGA circuit capable of processing SHOT descriptor at a very high rate depending on the neighborhood size, furthermore the application is very power efficient consuming just 0.27 W. Future work will encompass the full structure of the descriptor implementing local reference frame computation through the eigenvalue decomposition and experiment

about accuracy and recall in cluttered scene will be conducted.

Chapter 6

Conclusion

In this work, a stereo vision architecture with real time processing is proposed alongside a 3D descriptor hardware implementation. Both architectures can be used in a Visual Search pipeline for object recognition. The stereo vision architecture is shown in a prototype. The implementation is fully embedded in a working system using two cameras and one VGA socket. The final architecture can process in real time (>30fps) XGA frame size consuming less than 2 Watts. On the other hand, FPGA-SHOT is capable of processing points with no delay and a fully pipelined architecture allows to analyze the point stream at a frequency higher than 100MHz with a power consumption of less than 0.2 Watts.

With the low cost robotics becoming so pervasive, in the future years more and more low cost and low power hardware architectures will be proposed in the literature. My main goal was to show the improvement that can be achieved in this field using cheap devices like the Zedboard. With these architectures it is possible to obtain really good performance with low power usage.

Bibliography

[1] M. Vigliar, L. Puglia, M. Fratello, and G. Raiconi, "Sascr2: Enhanced hardware string alignment coprocessor for stereo correspondence," in *Embedded Computing (MECO), 2014 3rd Mediterranean Conference on*. IEEE, 2014, pp. 56–61.

[2] L. Puglia, M. Vigliar, and G. Raiconi, "Sascr3: a real time hardware coprocessor for stereo correspondence," in *International Conference Image Analysis and Recognition*. Springer, 2014, pp. 383–391.

[3] F. Tombari, S. Salti, L. Puglia, G. Raiconi, and L. Di Stefano, "A Radial Search Method for fast Nearest Neighbor Search on Range Images," *Recovering 6D Object Pose, Workshop of European Conference on Computer Vision, ECCV*, 2016.

[4] L. Puglia, M. Ionica, G. Raiconi, and D. Moloney, "Passive Dense Stereo Vision On The Myriad2 VPU," *Hot Chips: A Symposium on High Performance Chips*, 2016.

[5] L. Puglia, M. Vigliar, and G. Raiconi, "Real-Time Low-Power FPGA Architecture for Stereo Vision," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2016.

[6] ——, "FPGA-SHOT: an HLS architecture for 3D descriptors computation," *Journal of Real-Time Image Processing*, In Revision.

[7] H. Sunyoto, W. Van der Mark, and D. M. Gavrila, "A comparative study of fast dense stereo vision algorithms," in *Intelligent Vehicles Symposium, 2004 IEEE.* IEEE, 2004, pp. 319–324.

[8] D. Scharstein, R. Szeliski, and R. Zabih, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," in *Stereo and Multi-Baseline Vision, 2001.(SMBV 2001). Proceedings. IEEE Workshop on.* IEEE, 2001, pp. 131–140.

[9] T. Kanade, H. Kano, S. Kimura, E. Kawamura, A. Yoshida, and K. Oda, "Development of a video-rate stereo machine," in *Proceedings of International Robotics and Systems Conference (IROS95)*, 1995.

[10] O. Faugeras, B. Hotz, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin *et al.*, "Real time correlation-based stereo: algorithm, implementations and applications," Inria, Tech. Rep., 1993.

[11] A. Ansar, A. Castano, and L. Matthies, "Enhanced real-time stereo using bilateral filtering," in *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on.* IEEE, 2004, pp. 455–462.

[12] R. K. Gupta and S.-Y. Cho, "A correlation-based approach for real-time stereo matching," in *International Symposium on Visual Computing.* Springer, 2010, pp. 129–138.

[13] O. Veksler, "Fast variable window for stereo correspondence using integral images," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1. IEEE, 2003, pp. I–I.

[14] N. Baha and S. Larabi, "Accurate real time disparity map computation based on variable support window," *International Journal of Artificial Intelligence & Applications (IJAIA)*, vol. 2, no. 3, 2011.

[15] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 467–474.

[16] D. Scharstein and R. Szeliski, "Middlebury Stereo Dataset." [Online]. Available: http://vision.middlebury.edu/stereo/data/

[17] R. Kalarot and J. Morris, "Comparison of fpga and gpu implementations of real-time stereo vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*. IEEE, 2010, pp. 9–15.

[18] G. Yulan, B. Mohammed, S. Ferdous, L. Min, W. Jianwei, and K. Ngai, Ming, "A Comprehensive Performance Evaluation of 3D Local Feature Descriptors," *International Journal of Computer Vision*, 2015.

[19] A. E. Johnson and M. Hebert, "Surface matching for object recognition in complex three-dimensional scenes," *Image and Vision Computing*, vol. 16, no. 9, pp. 635–651, 1998.

[20] ——, "Using spin images for efficient object recognition in cluttered 3d scenes," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 21, no. 5, pp. 433–449, 1999.

[21] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik, "Recognizing objects in range data using regional point descriptors," *Computer vision-ECCV 2004*, pp. 224–237, 2004.

[22] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *European Conference on Computer Vision*. Springer, 2010, pp. 356–369.

[23] Y. Guo, M. Bennamoun, F. A. Sohel, J. Wan, and M. Lu, "3d free form object recognition using rotational projection statistics," in *Applications of Computer Vision (WACV), 2013 IEEE Workshop on.* IEEE, 2013, pp. 1–8.

[24] Y. Guo, F. Sohel, M. Bennamoun, M. Lu, and J. Wan, "Rotational projection statistics for 3d local surface description and object recognition," *International journal of computer vision*, vol. 105, no. 1, pp. 63–86, 2013.

[25] ——, "TriSI: A distinctive local surface descriptor for 3D modeling and object recognition," *8th International Conference on Computer Graphics Theory and Applications*, 2013.

[26] H. Chen and B. Bhanu, "3d free-form object recognition in range images using local surface patches," *Pattern Recognition Letters*, vol. 28, no. 10, pp. 1252–1262, 2007.

[27] ——, "Human ear recognition in 3D," *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 2007.

[28] J. J. Koenderink and A. J. Van Doorn, "Surface shape and curvature scales," *Image and vision computing*, vol. 10, no. 8, pp. 557–564, 1992.

[29] A. Flint, A. Dick, and A. Van Den Hengel, "Thrift: Local 3d structure recognition," *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*, pp. 182–188, 2007.

[30] A. Flint, A. Dick, and A. Van den Hengel, "Local 3d structure recognition in range images," *IET Computer Vision*, vol. 2, no. 4, pp. 208–217, 2008.

[31] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on.* IEEE, 2008, pp. 3384–3391.

[32] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3212–3217.

[33] M. Jin and T. Maruyama, "A real-time stereo vision system using a tree-structured dynamic programming on fpga," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 21–24.

[34] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-time high-quality stereo vision system in fpga," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 10, pp. 1696–1708, 2015.

[35] L. Zhang, K. Zhang, T. S. Chang, G. Lafruit, G. K. Kuzmanov, and D. Verkest, "Real-time high-definition stereo matching on fpga," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2011, pp. 55–64.

[36] R. Dieny, J. Thevenon *et al.*, "Bioinformatics inspired algorithm for stereo correspondence," *Computer Vision Theory and Applications (VISAPP), International Conference on*, 2011.

[37] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[38] D. Hoang and D. Lopresti, "Fpga implementation of systolic sequence alignment," *Field-Programmable Gate Arrays: Architecture and Tools for Rapid Prototyping*, pp. 183–191, 1993.

[39] R. J. Lipton and D. Lopresti, "A systolic array for rapid string comparison," in *Proceedings of the Chapel Hill Conference on VLSI*, 1985, pp. 363–376.

[40] Y. Shan, Z. Wang, W. Wang, Y. Hao, Y. Wang, K. Tsoi, W. Luk, and H. Yang, "Fpga based memory efficient high resolution stereo vision

system for video tolling," in *Field-Programmable Technology (FPT), 2012 International Conference on*. IEEE, 2012, pp. 29–32.

[41] S. K. Gehrig, F. Eberli, and T. Meyer, "A real-time low-power stereo vision engine using semi-global matching," in *International Conference on Computer Vision Systems*. Springer, 2009, pp. 134–143.

[42] Y. Shan, Y. Hao, W. Wang, Y. Wang, X. Chen, H. Yang, and W. Luk, "Hardware acceleration for an accurate stereo vision system using mini-census adaptive support region," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, p. 132, 2014.

[43] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation," in *Embedded Computer Systems (SAMOS), 2010 International Conference on*. IEEE, 2010, pp. 93–101.

[44] M. Jin and T. Maruyama, "Fast and accurate stereo vision system on fpga," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 1, p. 3, 2014.

[45] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Robotics-DL tentative*. International Society for Optics and Photonics, 1992, pp. 586–606.

[46] W. a. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Communications of the ACM*, vol. 16, no. 4, pp. 230–236, 1973.

[47] K. Fukunaga and P. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *Computers, IEEE Transactions on*, no. July, pp. 750–753, 1975.

[48] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Foundations of*

Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on. IEEE, 2006, pp. 459–468.

[49] J. H. Freidman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[50] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on.* IEEE, 1997, pp. 1000–1006.

[51] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 891–923, 1998.

[52] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[53] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," *International Conference on Computer Vision Theory and Applications (VISAPP '09)*, pp. 1–10, 2009.

[54] "Point Cloud Library." [Online]. Available: http://pointclouds.org/

[55] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab, "Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2684–2689, 2012.

[56] F. Tombari, S. Salti, and L. Di Stefano, "Performance evaluation of 3D keypoint detectors," *International Journal of Computer Vision*, vol. 102, no. November 2011, pp. 198–220, 2013.

[57] Y. Zhong, "Intrinsic shape signatures: A shape descriptor for 3d object recognition," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 689–696.

[58] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Procedings of the Alvey Vision Conference 1988*, pp. 147–151, 1988.

[59] L. Chang, J. Hernández-Palancar, L. E. Sucar, and M. Arias-Estrada, "Fpga-based detection of sift interest keypoints," *Machine vision and applications*, vol. 24, no. 2, pp. 371–392, 2013.

[60] T. Krajník, J. Šváb, S. Pedre, P. Čížek, and L. Přeučil, "Fpga-based module for surf extraction," *Machine vision and applications*, vol. 25, no. 3, pp. 787–800, 2014.

[61] "Zynq-7000 All Programmable SoC." [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds187-XC7Z010-XC7Z020-Data-Sheet.pdf

[62] J. Knopp, M. Prasad, G. Willems, R. Timofte, and L. Van Gool, "Hough transform and 3d surf for robust three dimensional classification," in *European Conference on Computer Vision*. Springer, 2010, pp. 589–602.

[63] S. Salti, F. Tombari, and L. Di Stefano, "Shot: Unique signatures of histograms for surface and texture description," *Computer Vision and Image Understanding*, vol. 125, pp. 251–264, 2014.

[64] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[65] D. Palossi, F. Tombari, S. Salti, M. Ruggiero, L. Stefano, and L. Benini, "Gpu-shot: parallel optimization for real-time 3d local description," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2013, pp. 584–591.

[66] D. Palossi, M. Ruggiero, and L. Benini, "3d cv descriptor on parallel heterogeneous platforms," *ACM Trans. Embed. Comput. Syst.*, 2015.

[67] I. Bravo, P. Jiménez, M. Mazo, J. L. Lázaro, and A. Gardel, "Implementation in fpgas of jacobi method to solve the eigenvalue and eigenvector problem," in *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on.* IEEE, 2006, pp. 1–4.

[68] I. Bravo, M. Mazo, J. L. Lázaro, P. Jiménez, A. Gardel, and M. Marrón, "Novel hw architecture based on fpgas oriented to solve the eigen problem," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 12, pp. 1722–1725, 2008.

[69] "7 Series DSP48E1 Slice." [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[70] "AXI Reference Guide." [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/v13_4/ug761_axi_reference_guide.pdf

[71] J. Volder, "The cordic computing technique," in *Papers presented at the the March 3-5, 1959, western joint computer conference.* ACM, 1959, pp. 257–261.

[72] R. Andraka, "A survey of cordic algorithms for fpga based computers," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays.* ACM, 1998, pp. 191–200.

[73] "Zedboard (Zynq^{TM}Evaluation and Development) Hardware User's Guide." [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf