Master Thesis

- Implementation of Stereo Vision Engine -

Project Report

Group 1072

Aalborg University
Electronics and IT

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Implementation of Stereo Vision Engine

Theme:
Master Thesis in Signal Processing and Computing

Project Period:
Spring Semester 2016

Project Group:
1072

Participant(s):
Tomas Brandt Trillingsgaard

Supervisor(s):
Peter Koch

Copies: 1

Page Numbers: 78

Date of Completion:
December 16, 2016

Abstract:

In this report, a fast high-precision stereo vision engine for implementation on an FPGA is studied. Different aspects of stereo vision and obstacles within this area are explored. Two stereo vision algorithms: *Efficient Edge Preserving Stereo Matching* and *Fast Cost-Volume Matching* have been studied and implemented in Python. A discussion concerning the algorithms computational complexity and stereo matching quality results in the choice of *Efficient Edge Preserving Stereo Matching* for further implementation. A final implementation was not achieved but the challenges of implementing an exponential function and the challenges with memory usage with large images were studied and solutions were found.

Contents

Preface **vii**

1 Introduction **1**
1.1 Introduction to stereo vision . . . 1
1.2 HSA Systems . . . 2
1.3 Motivation . . . 3
1.4 Problem description . . . 3
1.5 Delimitation . . . 3
1.6 Report Structure and Design Process . . . 4

2 Application Analysis **7**
2.1 The Basic Principle of Stereo Vision . . . 7
2.2 Epipolar Geometry . . . 10
2.3 Color space and grayscale . . . 11
2.4 Disparity precision . . . 12
2.5 Occlusions . . . 14
2.6 Occlusions filling result . . . 18
2.7 Wrap-up . . . 18

3 Design and test specification **19**
3.1 Requirement specification . . . 20
3.2 Test specification . . . 20

4 Algorithm design **23**
4.1 Efficient Edge Preserving Stereo Matching (EEPSM): . . . 24
4.2 Fast Cost-Volume Matching (FCV): . . . 26
4.3 Simulation and comparison . . . 31
4.4 Theoretical Complexity . . . 35
4.5 Choosing an algorithm . . . 36
4.6 Wrap-up . . . 37

5 Platform analysis **39**
5.1 Platform trade-off . . . 39
5.2 Zynq Z-7020 . . . 40
5.3 Hardware utilization . . . 41
5.4 Wrap-up . . . 43

6 Design methodology **45**
6.1 Wrap-up . . . 48

7 Architecture design **49**
7.1 Parallelism Analysis . . . 50
7.2 Allocation and Scheduling . . . 52
7.3 Assignment . . . 55
7.4 Implementation challenges . . . 56
7.5 Wrap-up . . . 63

8 Acceptance test **65**

9 Conclusion **67**

Bibliography **71**

A Allocation test **73**
A.1 General procedure . . . 73
A.2 Adder . . . 74
A.3 Subtract . . . 74
A.4 Adder/Subtract . . . 74
A.5 Multiplier - LUT . . . 74
A.6 Multiplier - DSP48 . . . 74
A.7 Divider . . . 75

B Middlebury data set **77**

Preface

This thesis has been written by Tomas B. Trillingsgaard, who is a student at the Signal Processing and Computing Master's Program, Aalborg University. The thesis serves as documentation for the authors master project, which have been devised in the period from February 1 to September 30, 2016. The theme for this project is *Signal Processing and Computering* where the subject *Implementation of a Stereo Vision Engine* is chosen.

Reading Instructions

The thesis is addressed to supervisors, students and other with interest in the field of Electronics and Information Technology. By extension the thesis presumes the reader have a basic knowledge within this field.

The thesis contains references following the IEEE citation style; [i]. These references points to the bibliography in the back of the thesis on page 72. The bibliography contains information about the source like author, title, year of release, etc.

Figures and tables are numbered according to the location in the thesis. List of figures and tables can be also be found in the back of the thesis on 72. Unless otherwise described, the figures and tables in the thesis have been produced by the author. When referring to these, their numbers will be used which will be followed the location if they are not on the same page.

Formulas and calculations are also numbered according to their location and referenced in the same way as figures and tables.

Aalborg University, December 16, 2016

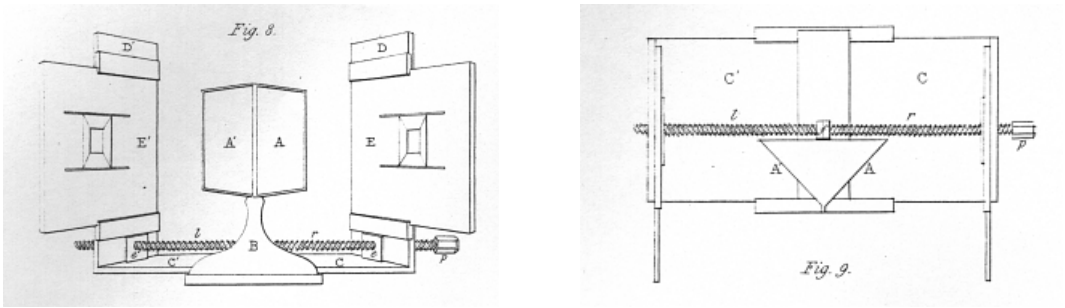
Tomas Brandt Trillingsgaard
<ttrill10@student.aau.dk>

Chapter 1

Introduction

In this chapter, the project is introduced and motivated. Furthermore, a brief description is presented for stereo vision and the use for it at HSA Systems, the company with whom the work has been conducted. Lastly, this chapter also describes a delimitation of the project and report.

1.1 Introduction to stereo vision



(a) Wheatstones stereoscope seen from the front

(b) Wheatstones stereoscope seen from above

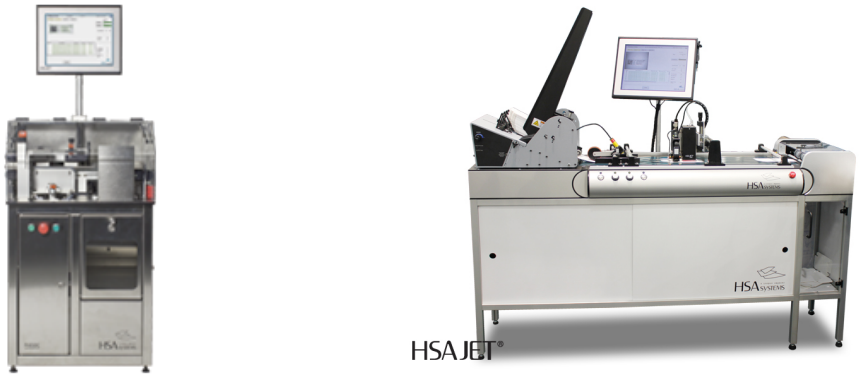
Figure 1.1: Illustration of Wheatstone's stereoscope from [23]

In 280 A.D the greek mathematician Euclid discovered that the perception of depth is caused by each eye receiving a dissimilar image of the same object. Throughout history, different people have been working on this concept. In 1833 Sir Charles Wheatstone began to mimic depth perception and forced the perception of depth by developing the stereoscope and his work on this is discussed in [23]. Figure 1.1 shows some illustrations of Wheatstones stereoscope. This stereoscope functions by using either two drawings or photographs where the point of view is displaced horizontally by a short distance. These

images are placed on the two surfaces marked E′ and E on figure 1.1a (p. 1). A′ and A on the same figure is two mirrors angled at 45° which reflect the images to the viewer. When the viewer move close enough to the mirrors A′ and A then the images will be isolated to each eye. This mimics the normal human depth perception and should trigger the brain to accept the images as a single 3D image. [22]

Around 1970, computer vision began appearing and a significant part of this research area is focused on stereo vision: the measurement of depth mimicking the human vision using two cameras [21]. The ability to measure depth enables a computer to distinguish between objects and hence to better interact with and react to the world. This project adds to this research by discussing various aspects of obtaining real time execution without neglecting the quality of the stereo matching.

1.2 HSA Systems



(a) PV650C[11] **(b)** Pharma lay-flat media [11]

Figure 1.2: Some products from HSA Systems

HSA Systems is a danish company with an R&D department in Aalborg, which develops and manufactures high-resolution inkjet printer systems and a range of other products. Figure 1.2 shows two examples of these printer systems. These systems print labels, bar codes etc. on packages and verify the quality print.

HSA Systems wishes to track packages going through their printer systems. A strategically placed stereo vision camera will provide knowledge of how many and where these packages are in the printer system. In case of errors and the like, the printer system is then able to notice when the conveyor belt in their system is empty and ready to reset. This is a significant improvement to the current solution at HSA Systems where no knowledge of locations of packages is available except for a single camera scanning labels. Currently in

case of errors the conveyor belt is set to run for a set time until it is assumed that the belt is empty.

Currently some products exist which can produce real time depth images such as the Asus Xtion PRO [3] but these 3D sensors have a low depth resolution due to using small image resolution i.e. VGA (640×480).

For detecting packages a high depth precision is not needed since packages normally are larger boxes (above $2 \times 2 \times 2$ cm) but HSA Systems would like to develop a stereo vision camera which also can be used for future assignments where depth precision requirement is higher. Hence HSA Systems have given the requirement for the stereo vision system to have very precise depth measurements of ≤ 5 mm at distances between 0.5-1.5 m .

1.3 Motivation

The area of stereo vision has been researched thoroughly and accurate algorithms have been developed but most algorithms are very computationally complex. Some real time algorithms have been developed but these focus on low resolution images. As described HSA Systems wishes for a stereo vision system which can execute in real-time (10 fps) for current assignments while also having a high depth precision for the future assignments.

1.4 Problem description

As mentioned earlier HSA Systems wishes for stereo vision system which can track packages going through their printing systems and for future assignments. This project will attempt to develop a stereo vision system which can execute real time and have a high depth precision. This system should be implemented on an FPGA.
It is therefore essential that the following questions are asked, and in part we will try to answer them:

- What obstacles occur within stereo vision?
- Which stereo vision algorithms exist, both being computationally efficient and at the same time providing good vision results?
- How can an architecture be designed and optimized for executing a stereo vision algorithm?

1.5 Delimitation

This project is mainly concerned with the design and implementation of a hardware architecture on an FPGA. Therefore, we will not focus on developing a new stereo vision

algorithm. Limitations and issues with stereo vision algorithms will be analyzed but a simpler solution will be used for most obstacles.

1.6 Report Structure and Design Process

The A^3 model is a basic design model originally suggested by teaching staff at AAU and is illustrated on figure 1.3. The model consists of three design domains which can be explored and the report is structured after this model. These domains are Application, Algorithm, and Architecture.

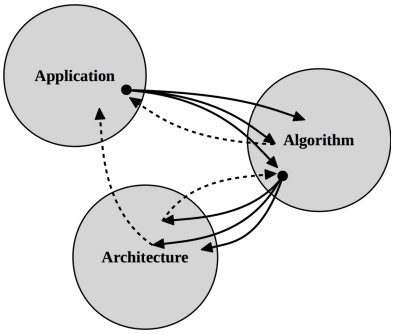


Figure 1.3: A^3 model

The search for a solution starts in the application domain where the problem, the application and the specifications are explored. Chapter 2 (p. 7): Application Analysis will explore the Application domain and the resulting specification for the application are presented in chapter 3 (p. 19). The problem and application - which has been specified by HSA Systems - is specified in this chapter and will be explored further in chapter 2 (p. 7).

As represented by the solid arrows on figure 1.3, there are several possible algorithms that may match a specific application and specification. Similarly, several architectures may be used to implement a given algorithm. Exploration of the algorithm is described in chapter 4 (p. 23) .

Chapter 6 (p. 45) will describe a methodology to traverse from an algorithm in the algorithm domain to some hardware architecture in the architecture domain.

Notice the dashed arrows in figure 1.3. These arrows show that when exploring one domain new information might arise which calls for changes in a former domain, and requiring a new iteration through the domains. This basically illustrates that the design process is iterative and thus typically requires several synthesis/evaluation loops before a satisfactory solution is found. In this context "satisfactory" means either compliance with

the specification or "best possible".

Chapter 8 (p. 65) specifies how the testing is done to determine whether the found solution complies with the requirements in chapter 3 (p. 19). Chapter 9 (p. 67) concludes the project and its findings.

Chapter 2

Application Analysis

This chapter will explore the application domain in the A^3 model described in section 1.6 (p. 4) and it is the domain marked on figure 2.1. First, the basic principles of stereo vision will be described and then other aspects such as color versus grayscale etc. are analyzed. The results from this chapter will be used in chapter 3 (p. 19).

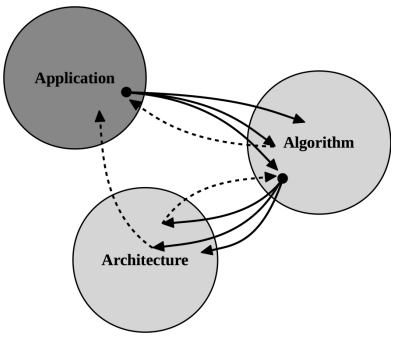
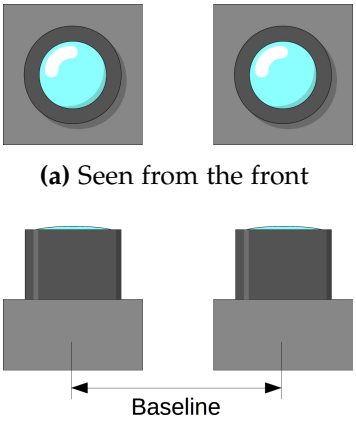


Figure 2.1: A^3 model with the application domain marked

2.1 The Basic Principle of Stereo Vision

A standard stereo vision setup consists of two similar cameras placed horizontally at a specified distance from each other. This distance is called the baseline. Figure 2.2 (p. 8) shows an example of this setup.

Figure 2.3 (p. 8) shows how a scene is seen by the camera, is inverted in the optical center and projected onto the image sensor in the camera. The original image plane is located at the position of the image sensor but it is inverted compared to the scene captured. To simplify comparisons to the real world, an image plane can be placed opposite of the optical center at the same distance from the center and this image plane will not be

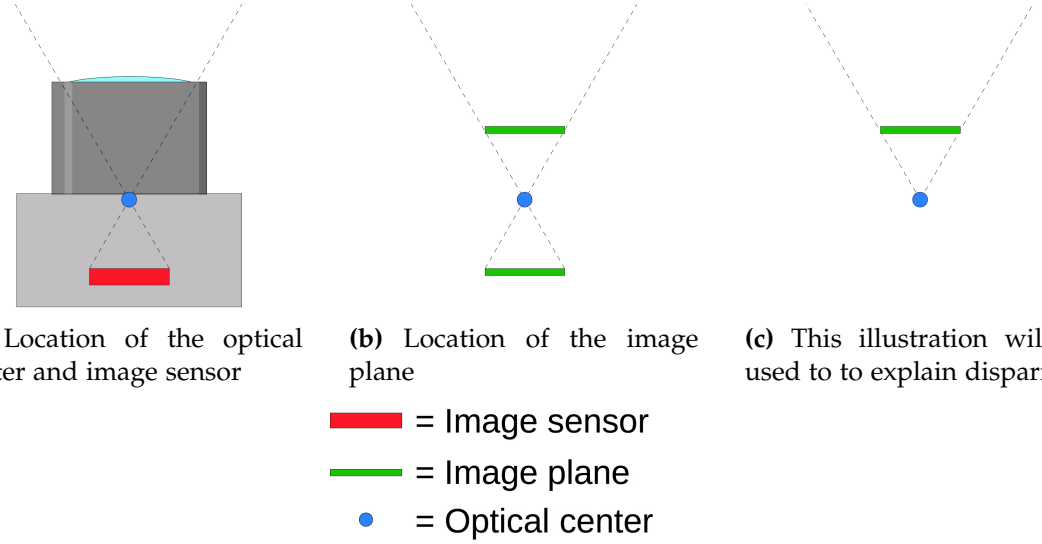


(a) Seen from the front

(b) Seen from above

Figure 2.2: Illustration of a standard stereo vision setup

inverted.



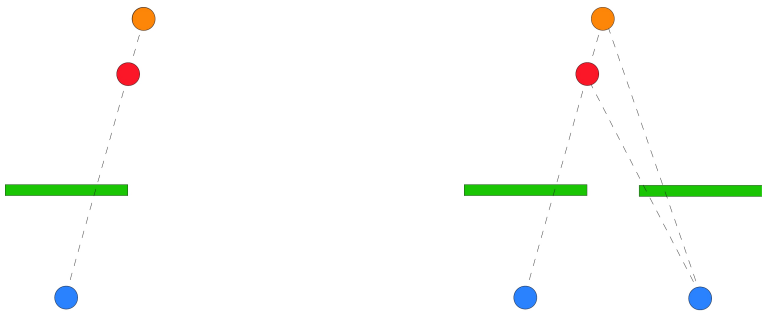
(a) Location of the optical center and image sensor

(b) Location of the image plane

(c) This illustration will be used to to explain disparity

Figure 2.3: Illustration of going from camera to image plane

Figure 2.4a (p. 9) shows that a single camera is not able to differentiate between two points at the same angle from the optical center at different distances. Figure 2.4b (p. 9) illustrates how adding the second camera allows differentiating between the two points.
Figure 2.5 (p. 9) shows how the distance to a point can be calculated from the difference in x-positions on the image plane (the disparity). Figure 2.5a (p. 9) and 2.5b (p. 9) shows how the disparity changes depending on the distance to the point. Figure 2.5c (p. 9) shows the point in the scene (p), where line of sight crosses the image planes (i_1 and i_2) and the optical centers (c_1 and c_2). From these points two similarly angled triangles can be created.



(a) Seen from a single camera **(b)** Seen from two cameras

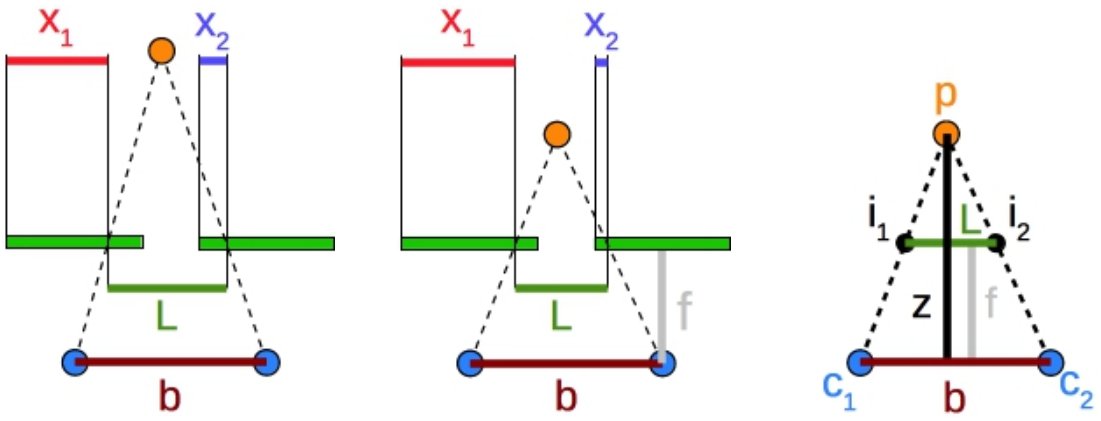
Figure 2.4: Example of two points in a scene at different depths

One between p, i_1 and i_2 and the other triangle between p, c_1 and c_2. For similarly angled triangles the ratio between the height and the bottom width is the same for each triangle and hence the following equation can be formed:

$$\frac{b}{z} = \frac{L}{z-f} = \frac{b-(x_1-x_2)}{z-f} \tag{2.1}$$

$x_1 - x_2$ is also called the disparity, d, and equation 2.1 can be simplified:

$$z = \frac{b \cdot f}{d} \tag{2.2}$$



(a) Disparity for a point far way **(b)** Disparity for point close **(c)** Illustration of triangles used for calculating the disparity

Figure 2.5: Illustration of how to calculate depth from disparity

In equation 2.2 b and f are known (b is the baseline and f is the focal length) hence only the disparity is needed to find z. So to find the distance to a point, the location of the point

in the two images should be found and the displacement may be computed.

This explains the basics of stereo vision. The rest of this chapter will venture into other areas of stereo vision and describe the difficulties and solutions for each area.

2.2 Epipolar Geometry

Section 2.1 assumes that the stereo image planes are ideal, align exactly and being parallel with the baseline but in a real scenario the cameras will have small imperfections and variations which will make the image planes not perfectly align.

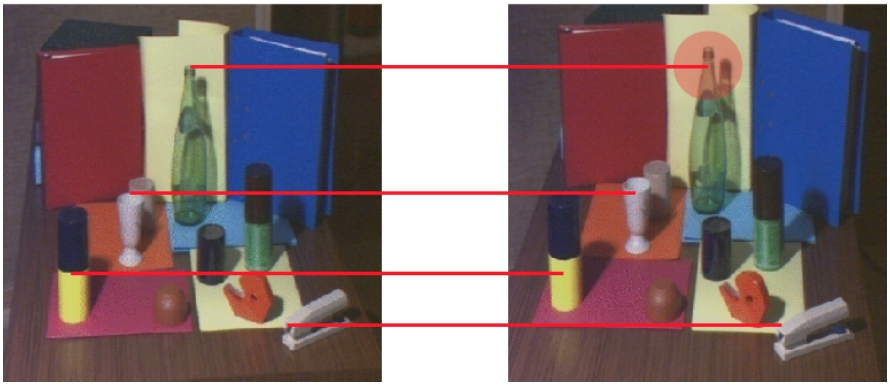


Figure 2.6: Non-rectified stereo pair [14]

Figure 2.6 shows a pair of stereo images. As seen when searching for a corresponding point in the second camera (e.g. the top of the bottle) then a 2D search area is needed. To simplify the search epipolar geometry can be used.

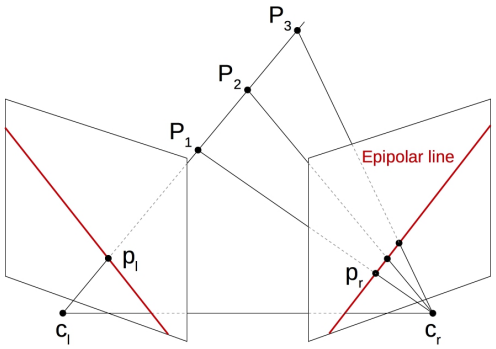


Figure 2.7: Illustration of epipolar geometry

Epipolar geometry occurs when a scene is seen from two different views. Figure 2.7 illus-

trates epipolar geometry. An epipole is the projection, on one camera's image plane, of the optical center of the other camera and is noticed on figure 2.7 (p. 10) as the points where the line between the centers crosses the image planes. The red line going through p_l (the projection of point P on the image plane) and the epipole is called an epipolar line and a corresponding epipolar line can be found on the other image plane. When searching for the corresponding point in the other image the search can be simplified from a 2D search to a 1D search along the epipolar line. To simplify the search further, the image can be rectified.

Rectification will transform the stereo images to remove lens distortion and make them into standard form. The standard form is helpful since all epipolar lines then will be horizontal and this simplifies the search for corresponding points to search along the x-axis. Figure 2.8 shows the stereo image pair from figure 2.6 (p. 10) but rectified. As seen, the corresponding points can now be found by following the horizontal lines or the x-axis.

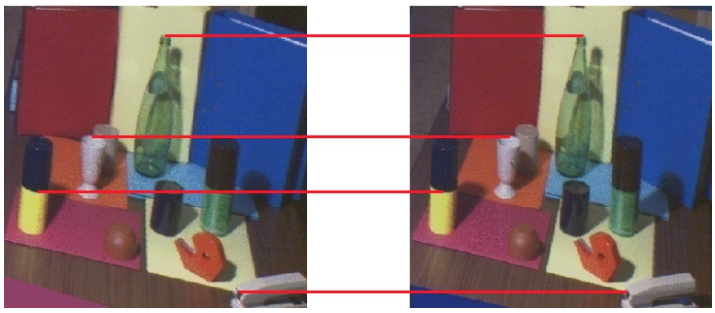


Figure 2.8: Rectified stereo pair [14]

The issue with rectification is that it is difficult to get a perfect match with the stereo setup since every camera and its lens will be unique and require an all new manual calibration. HSA Systems theorizes that a system can be developed which instead of rectifying the image will find the epipolar lines for each epipole and feed this information to the stereo cameras.

In this project stereo image pairs from Middlebury Vision Test sets [18] will be used. These image pairs are rectified, and rectification of image will not be discussed further in this project.

2.3 Color space and grayscale

Colors can be represented in many different ways digitally using color spaces. Two of the most common color spaces are grayscale and RGB. Some studies have investigated what impact different color spaces can have on the result from a stereo vision algorithm.

Measure	Type	Correct [%]	Time
Ncc	G	52.3	52
	C	55.2	141
D_1	G	49.5	63
	C	51.6	140
PRATT	G	29.1	86
	C	45.2	225
ISC	G	44.9	126
	C	52.6	245
$SMPD_2$	G	49.9	569
	C	56.5	2109

Table 2.1: Part of table containing results from [5]

For this project the impact of color spaces have not been studied but instead the findings in [5] are used. This study describes the impact of color spaces on stereo matching by investigating 9 color spaces and 3 different methods. Table 2.1 shows parts of the results from [5]. In this table the *Measure* column is the algorithm used, *Type* specifies whether it is grayscale (G) or color (C) with the best color space used, the *Correct* column is the percentage of correct matches and *Time* is the execution time. The article concludes that using color always results in better matching, but from table 2.1 it is seen that using grayscale lowers the execution time significantly compared to using color. In most cases without significant impact on the matching results.

Since the main focus of the project is on a fast stereo vision algorithm and minding the results from table 2.1, it is decided to use grayscale images if the tradeoff is not too significant.

2.4 Disparity precision

The depth resolution depends on different things in the stereo camera setup: the camera resolution, the focal length, the baseline etc. HSA Systems requires that the system has a $\leq$5 mm depth resolution between 0.5 m and 1.5 m.
The sensor which will be used for a final product is a *Sony IMX264* since this sensor is used in the company's smart camera products. This sensor has a resolution of 2464×2056, a pixel size of 3.45 µm and a frame rate of 35.7 fps. Full specifications of the sensor can be found in [19]. HSA Systems requires that the baseline is $\leq$10 cm.

To calculate the precision equation 2.2 (p. 9) is used. This equation is repeated here:

$$z = \frac{b \cdot f}{d} \tag{2.3}$$

The disparity, d, of equation 2.3 is expressed in pixels and may be converted to mm by using the the pixel size (p_{size}) of the camera sensor. A focal length f should be chosen so the scene at max distance will be 1.5×1.5 m and figure 2.9 shows the relationship between scene size and focal length.

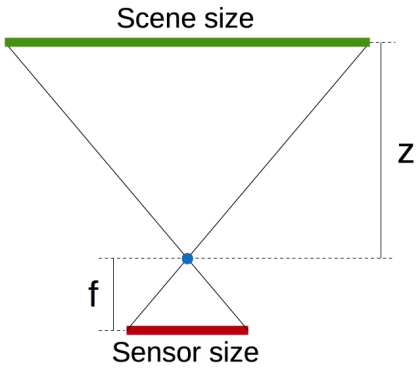


Figure 2.9: Illustration of the relationship between sensor size, scene size, focal length and distance

Since the two triangles in the figure are similar the focal length can be determined using:

$$f = \frac{z \cdot s_{sensor}}{s_{scene}} \tag{2.4}$$

Where z is the distance, s_{sensor} is the size of the sensor which depends on pixel size and resolution and s_{scene} is the size of the scene. Since the camera resolution is not square a focal length for both horizontal and vertical scene size have to be calculated and the lowest focal length is chosen.

$$f_h = \frac{1500 \cdot 2464 \cdot 0.00345}{1500} = 8.5mm \tag{2.5}$$

$$f_v = \frac{1500 \cdot 2056 \cdot 0.00345}{1500} = 7.09mm \tag{2.6}$$

A focal length of 7.09 mm is chosen and this results in a scene size of 1798×1500 mm. With the baseline and focal length determined, it can be calculated when the disparity precision is ≤5 mm.
The derivative of z with respect to d gives the precision at a specific disparity.

$$z' = -\frac{b \cdot f}{p_{size} \times (d^2)} \tag{2.7}$$

Inserting the known values and the wanted precision the disparity d is found to be 202,73 but since the disparity is an integer this value should be rounded up to 203. This results in a disparity precision of -4.99 mm and the distance where this precision is achieved is at 1012.35 mm. This does not comply with the requirement from HSA Systems. Using sub-pixel refinement a precision of 10 mm can be used to achieve the same precision. This precision is achieved at a disparity of 144 and the precision is then -9.91 mm at the distance 1427 mm. This still doesn't comply with the requirement.
To improve the precision further either the resolution, the baseline or the focal can be changed but each requires another requirement to fail. The resolution requires a new camera sensor. The baseline need to be at least 11 mm larger. The focal length needs to at least 0.73 mm higher but this results in a scene size of 1631×1361 mm.

It is chosen to reduce the scene size to 1631×1361 mm and use a focal length of 7.82 mm. This results in a precision of 10 mm at $\approx$1500 mm so sub-pixel refinement has to be used. The resulting disparity range will be 303. Figure 2.10 shows the disparity precision at different distances with the chosen baseline, focal length, and sensor.

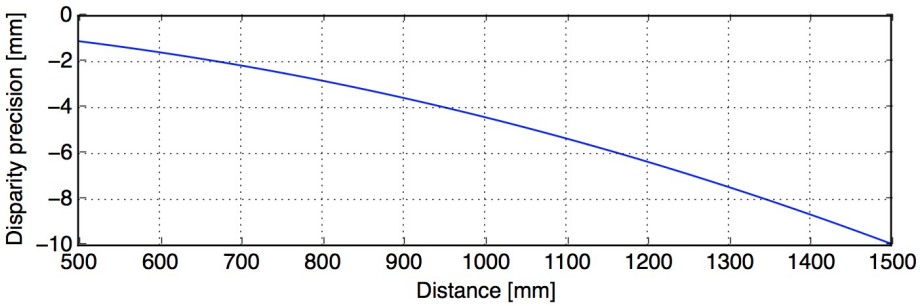
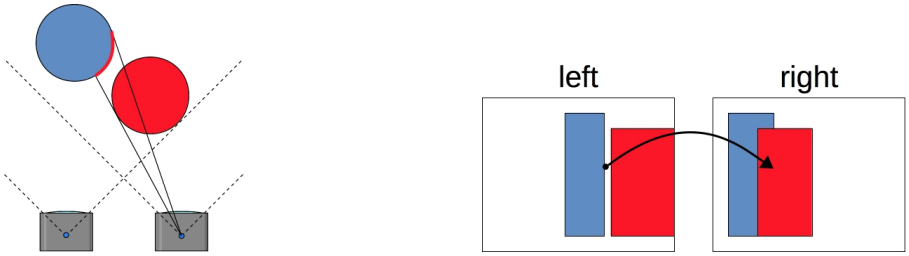


Figure 2.10: Disparity precision in the distance range 0 5−1.5 m

2.5 Occlusions

Occlusions occur when an object closer to the camera setup entirely or partially blocks an object behind it. Figure 2.11 (p. 15) illustrates two cylinders seen by a stereo camera setup where occlusion occurs. Figure 2.11a (p. 15) shows that the blue cylinder is only partially visible to the right camera because the red cylinder hides some of the blue cylinder (marked with red) while the left camera can see the whole of both cylinders. When searching for corresponding points in the occluded area issues occur. As illustrated by the arrow on figure 2.11b (p. 15), the edge of the blue cylinder can not be found and it will result in calculating a wrong disparity value.

There exist different types of occlusions: partial occlusions, self-occlusions, border occlu-



(a) Stereo camera setup with two cylinders in the scene seen from above

(b) Image seen by each camera in figure 2.11a

Figure 2.11: Illstration of occlusions

sions and total occlusions. Partial occlusions are when an object is only partially obstructed as seen on figure 2.11. Self-occlusion occurs on round surfaces such as faces, and balls and as seen on figure 2.12a the surfaces marked with red can be seen by one camera but not the other camera. Border occlusions occur when an object or part of an object is outside the view of one camera but not the other camera as seen with the blue object on figure 2.12b. Total occlusion is when an object is completely hidden by an object in the view of one camera but not in the view of the other camera and an example of this is seen with the red object on figure 2.12b.

(a) Illustartion of self occlusion seen from above

(b) Illustration of total and border occlusions seen from above

Figure 2.12: Illustration of different types of occlusions

Occluded points can be found by running stereo matching again but switching which camera is used as the reference and then compare the disparity values found for each direction. When occluded areas are found these can be filled using different methods. For this project the findings from [12] is used which will be described in section 2.5.1 (p. 16) to 2.5.4 (p. 17). The methods will only be briefly described in this report and a more thorough description can be found in [12]. All these methods assume that the stereo matching is using the left image as reference i.e. the points in the left image are searched for in the right image.

2.5.1 Neighbor's Disparity Assignment (NDA)

This is one of the simpler methods to fill occlusions. It functions by selecting an occluded point, p_L, then finds then nearest non-occluded point, q_L, to the left when filling non-border occlusion. With border occlusion, the nearest point to the right is found instead. This method assumes that this non-occluded point is part of the same surface as the occluded point (this can be seen on figure 2.11) and the disparity value from q_L can be assigned to p_L. This method has some issues. In cases of total occlusions (see figure 2.12b) wrong disparity values will be given to the total occluded object since it is not a part of the nearest surface with non-occluded points to the left. In cases with self-occlusions the occluded area should have disparity values close to the disparity values of the non-occluded points to the right (This will be the area of the surface which is in view of both cameras) but using NDA will give the occluded area disparity values corresponding to the background.

2.5.2 Diffusion in Intensity Space (DIS)

This method is inspired by diffusion. Diffusion is the movement of molecules or atoms from a high concentration region to a low concentration region.

After detecting occluded regions with cross-checking during stereo matching, the diffusion energy for the region is approximated. This method is dependent on the stereo matching algorithm because it uses the energy from the last iteration to determine initial diffusion energy for the area. But a change to the method can be made to make it independent from the stereo matching. The initial diffusion energy will be 0. Then the diffusion energy for an occluded point $E(p_L)$ is calculated by basically integrating the energies of non-occluded points with the same disparity value in the neighborhood of the occluded point p_L. The disparity value which results in the lowest diffusion energy will be given to p_L. This is repeated for each occluded point until every point have been filled.

2.5.3 Weighted Least Squares (WLS)

In this method, all the non-occluded and filled occluded neighbors in a neighborhood around the occluded point is considered valid points and is used as control points in interpolation. But since the neighborhood contains both foreground points and background points and the occluded point is expected to be a part of the background then the background points should have higher influence than foreground points.

Therefore a weighted least square problem is set up where the residual error term is weighted so non-occluded points belonging to foreground objects will have a lower influence on the least square problem than points belonging to the background. To distinguish between foreground points and background points it can be assumed that the color inten-

sity between objects is significantly different.

2.5.4 Segmentation-based Least Squares (SLS)

This method is an alteration of the WLS method. The biggest difference between WLS and SLS is that SLS only uses non-occluded points as control points. The control points is a subset of the non-occluded neighboring points. The control points are segmented from the neighborhood by applying different constraints: visibility constraint, disparity gradient constraint, and color similarity cues.

First find occluded points which have at least one non-occluded neighboring point. Then these points are sorted by a priority which is found by checking the homogeneity i.e color similarity between the occluded point, p_L and the neighboring non-occluded points.

When the occluded point with the highest priority is found then some initial control points from the neighborhood are needed. First only points from the background is wanted therefore only points in the neighborhood which have a disparity value lower than the foreground. The nearest non-occluded point to the left of the occluded point is assumed to be part of the background, $l_{p_{Lb}}$, and the nearest non-occluded point to the right is assumed to be part of the foreground, $l_{p_{Lf}}$. But narrow objects in the foreground will make the non-occluded point on either side of the occluded area be part of the background. So some constraints are setup to get all the background points. The constraints will find every point which has a disparity value close to the background points $l_{p_{Lb}}$ or lower than the foreground points $l_{p_{Lf}}$.

All points in the neighborhood, which satisfies these constraints, is assumed to be part of the background but might contain points from multiple background surfaces. It is assumed, from looking at the data sets from [18], that the neighborhood only contains points from either one background surface or two background surfaces. If the difference between highest and lowest value is too high then this new neighborhood contains two objects and have to be divided into two neighborhoods. This is done by grouping all points with disparity values close to the lowest disparity and all points with disparity values close to the highest disparity. Then it can be determined which group the occluded point belongs to by looking at color distance to each group.

With these control points found a least square problem can be set up using these control points for finding the disparity value of the occluded point.

2.6 Occlusions filling result

In [12] it is found that SLS gives the best result followed by DIS, NDA, and WLS but the runtime SLS is slower than NDA and WLS and faster than DIS. Since the project focuses on implementing a fast stereo algorithm and focuses more on the stereo algorithm itself, NDA is chosen as the method to fill occlusions since it is the better performing of the two faster methods.

2.7 Wrap-up

To conclude this chapter the findings for each area in stereo vision will be examined and it will be specified which solution will be used from this point.

Rectification of Images

This project will not delve further into the subject of rectification. Instead data sets from [18], will be used in this project as they are already rectified.

Color Space

The result from [5] shows that color spaces in most cases are slightly better than using grayscale images but the computational complexity is much higher. This result depends on the algorithm used therefore when an algorithm has been chosen a test should check if grayscale images perform much worse than using color images and what the impact on run time is.

Resolution and Disparity Precision

To have a disparity precision of 2 mm at 1500 mm using *Imaging Source DMK 72BUC02* cameras with a resolution of 2592×1944 and a pixel size of 2.2 µm a lens with a focal length of 10 mm should be used together with subpixel refinement.

Occlusions

4 different methods to fill occlusions were described and from these NDA is chosen since it is the better performing of the two faster algorithms.

Chapter 3

Design and test specification

This chapter elaborates on the requirements for the system and derives a test specification which will describe how to test for the requirements.

A way to identify the performance of a design using different design metrics is the cost function. The cost function specifies the overall cost of a design and is a function of the different design metrics:

$$C = f(A, T, P, N, S) = a_1 \cdot A + a_2 \cdot T + a_3 \cdot P + a_4 \cdot N + a_5 \cdot S \tag{3.1}$$

where A is area, T is time, P is power, N is numerical properties, S is the stereo matching result from Middlebury test and a_i tells the importance of the associated metric.
It is the task of the system designer to minimize this cost. And due to a_i the cost function can be changed to fit the priorities of the application.

For this thesis, the number one priority is the time metric since it is required that the algorithm should be executable in real-time. The stereo matching metric is also important since it describes the quality of the resulting disparity map i.e. the number false disparity values. Numerical properties are 3rd most important metric since this can affect the result from the algorithm since the cost value may have added noise. Area is not as important because if the target FPGA is too small then HSA Systems will use a larger FPGA but larger area often equals more expensive hardware. Power is not important either since the stereo setup is intended as a part of industrial systems hence power is easily accessible.

An ordered list of priorities for the design metrics in this project:

1. Time
2. Stereo matching result
3. Numerical properties
4. Area
5. Power

In this thesis, the cost function will be used when e.g comparing the different algorithms found in chapter 4 (p. 23).

3.1 Requirement specification

This section will contain a table with the requirements for the system. Each parameter will be given a number, a value which should be met, the unit for the value, additional information for the requirement and a reference to where the discussion for the requirement can be found.

No.	Parameter	Value	Unit	Additional Information	Source
1	Frame rate	≥ 10	fps		Section 1.3 (p. 3)
2	Disparity precision	≤ 5	mm	• Either directly or using subpixel refinement • In the range 0.5-1.5 m	Section 1.2 (p. 3)
3	Camera resolution	2464×2056	pixels		Section 2.4 (p. 12)
4	Pixel size	3.45	µm		Section 2.4 (p. 12)
5	Focal length	7.09	mm		Section 2.4 (p. 13)
6	Scene size	1.5×1.5	m		Section 2.4 (p. 13)
The algorithm should be implemented on a *Xilinx Zynq Z7020*					

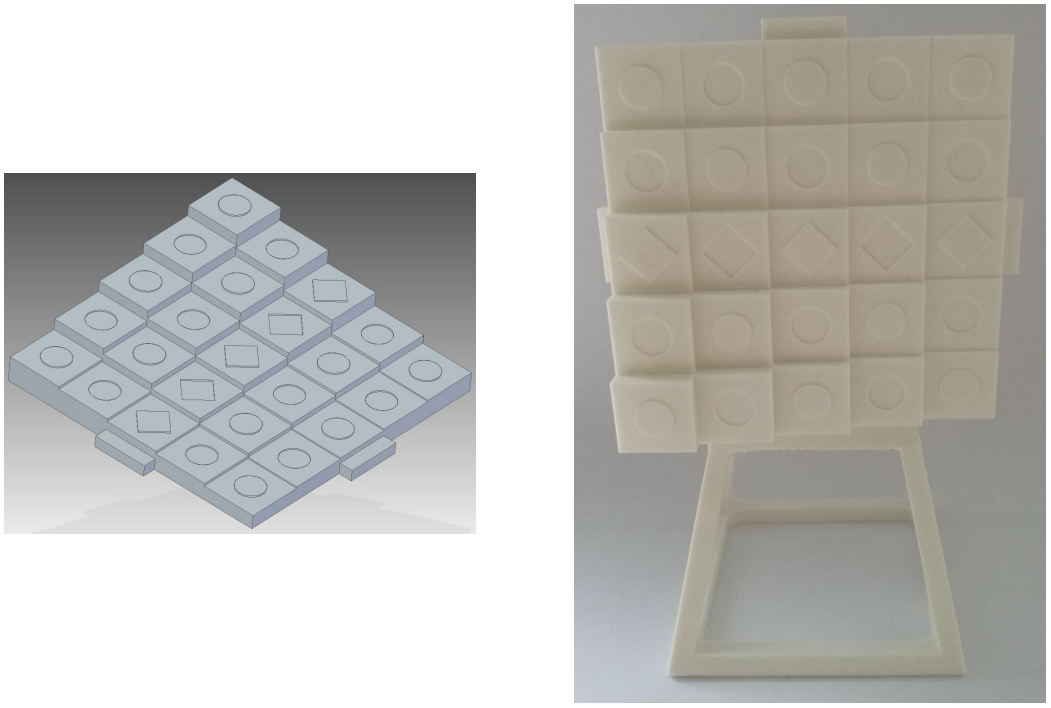
3.2 Test specification

This section will describe how the system can be tested in order to ensure that the requirements are fulfilled.

Requirement 1: frame rate can be tested by running the finalized implementation and measure the runtime which should be $\leq$100 ms.

Requirement 2: disparity precision. This requirement can be tested if a working stereo camera setup is developed.
HSA Systems has produced a depth precision test object with small depth differences. The

object is shown on figure 3.1. The object consists of five sets of steps with each stair having different depth increment. Looking at 3.1b the set of steps at the bottom have a depth difference of 5 mm between each step. The set of steps just above have a depth difference of 4 mm between each step. This pattern continues and results in the top set of steps to have a depth difference of 1 mm. The small shapes in the middle of each step have a depth difference of 0.5 mm alternately protrude from or recess into the steps.



(a) 3D model of depth precision test object

(b) Picture of 3D printed depth precision test object

Figure 3.1: Depth precision test object

Placing this object at a specific distance the depth precision can be tested. If the steps in lowest set of steps can be distinguished from each other in the disparity map then the depth precision is $\leq$5 mm.
If the implementation can distinguish between the lowest set of steps and the object is placed at 1.5 m from the camera setup then the requirement is fulfilled. The rest of the steps help determine if the camera have a better precision than required.
The Middlebury test set does not contain images where specific areas contain depth increments of 2 mm and therefore these images can not be used for testing this requirement and calculations 2.4 (p. 12) are used instead.

Requirement 3: camera resolution is fulfilled by choosing the correct camera hardware but

is essential to be fulfilled for requirement 2 to be fulfilled.

requirement 4: pixel size is fulfilled by choosing the correct camera hardware but is essential to be fulfilled for requirement 2 to be fulfilled.

requirement 5: focal length is fulfilled by choosing the correct camera hardware but is essential to be fulfilled for requirement 2 to be fulfilled.

Chapter 8 (p. 65) will perform the available tests.

Chapter 4

Algorithm design

This chapter will explore the algorithm domain in the A^3 model. It is the domain marked on figure 4.1. This chapter will describe the two stereo vision algorithms, *Efficient Edge Preserving Stereo Matching* (EEPSM) and *Fast Cost-Volume Matching* (FCV). Lastly, a simulation of each algorithm is conducted and described and the results of these simulations are compared and from this, an algorithm is chosen.

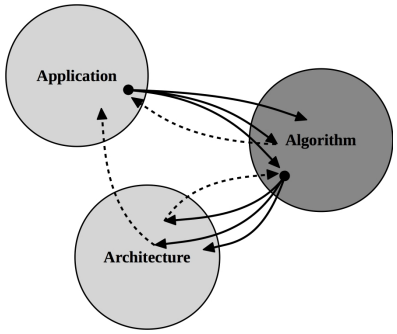


Figure 4.1: A^3 model with the algorithm domain marked

After the application analysis, a search for a fitting stereo vision algorithm can begin. During an internship at HSA systems a standard normalized cross-correlation (NCC) stereo matching algorithm was developed. An issue with the NCC algorithm is that it is not accurate near edges resulting in false matching around edges. For this project HSA systems wishes to find an algorithm which can be fast and is edge preserving. After some research, two algorithms were found. These algorithms are *Efficient Edge Preserving Stereo Matching* and *Fast Cost-Volume Matching*. The description of these algorithm comes from [6] and [10]. The pseudo code for the EEPSM is created by us while the FCV pseudo code comes from [9] with some details added by us.

4.1 Efficient Edge Preserving Stereo Matching (EEPSM):

This algorithm is described in [6]. This algorithm first calculates a cost for each pixel and disparity. This cost is a combination of the sum of absolute differences (SAD) and hamming distance of the census transform around each pixel. First the SAD is calculated:

$$C_d^{SAD}(x,y) = \sum_{i=1}^{3} |I_l(x,y,i) - I_r(x+d,y,i)| \tag{4.1}$$

where i is the color (either red, green, or blue), $I_l(x,y,i)$ is a pixel in the left image or the reference image, and $I_r(x+d,y,i)$ is a pixel shifted by the disparity, d, in the right image or the target image.
This cost only uses the differences in color in a single pixel from each image and not a window around the pixel. Next the cost for a census transform is calculated.

$$C_d^{CENSUS}(x,y) = Ham(CT_l(x,y), CT_r(x+d,y)) \tag{4.2}$$

Where $Ham(x_1,x_2)$ is the hamming distance between two bit strings and $CT(x,y)$ is the census transform around the pixel at x,y of the grayscale version either the left image or the right image. The census transform converts the pixels in window around a center pixel into a bit string. Figure 4.2 shows a 3×3 census transform. If the intensity (grayscale value) of a pixel is higher than the center then it is converted to 1 and 0 otherwise and then bits are inserted into a bit string.

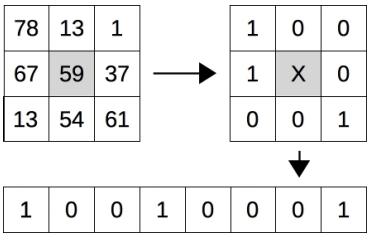


Figure 4.2: Illustration of census transform

When the two cost values have been calculated they can be combined to a single cost value.

$$C_d(x,y) = \alpha \cdot C_d^{SAD}(x,y) + (1-\alpha) \cdot C_d^{CENSUS}(x,y) \tag{4.3}$$

With the cost calculated the next step in the algorithm is to aggregate the cost. This is done in 3 steps. First step is to calculate permeability weights. Permeability is known from biomedicine and describes the ability to transfer molecules through a cell membrane. The permeability weights are inspired by this and describes how well the color transfers from one pixel to another pixel and is expressed as:

$$\mu(x,y) = \min(e^{\frac{-\Delta R}{\sigma}}, e^{\frac{-\Delta G}{\sigma}}, e^{\frac{-\Delta B}{\sigma}}) \tag{4.4}$$

Where ΔR, ΔG, and ΔB is the difference in the specified color between two neighboring pixels and σ is a smoothing factor.

The permeability weights should be calculated for each direction: up, down, left, and right and an example of the permeability of the downwards direction is shown in equation 4.5.

$$\mu_D(x,y) = \min(e^{\frac{-|I_l(x,y,1) - I_l(x,y-1,1)|}{\sigma}}, e^{\frac{-|I_l(x,y,2) - I_l(x,y-1,2)|}{\sigma}}, e^{\frac{-|I_l(x,y,3) - I_l(x,y-1,3)|}{\sigma}}) \tag{4.5}$$

The next step is to aggregate the cost in equation 4.3 (p. 24) horizontally and this is achieved by using successive weighted summation (SWS) of the cost values along the left and right direction. The weights used in this summation will be the permeability weights for the right and left directions. The SWS for the right direction is expressed as:

$$C_d^R(x,y) = C_d(x,y) + \mu_R(x-1,y)C_d^R(x-1,y) \tag{4.6}$$

$$C_d^R(x,y) = C_d(x,y) + \sum_{i=1}^{x-1}\left(C_d(x-i,y)\prod_{j=i}^{i}\mu_R(x-j,y)\right) \tag{4.7}$$

From equation 4.7 it is noticed that the cost, $C_d^R(x)$, will be affected by cost values from pixels to the left of point x but the permeability weight ensures that only pixels close to point x and with similar color will have a large influence. When there are large changes in color it is assumed that the pixels will belong to a new object and therefore it ensures that only pixels from the same object have an influence on the cost.

When aggregating the right SWS the algorithm starts in the left side and equation 4.6 is used as an update rule while moving in the right direction. A similar update rule exists for the left SWS. When the left SWS and the right SWS have been performed then these can be combined to a horizontal SWS:

$$C_d^H(x,y) = C_d^R(x,y) + C_d^L(x,y) \tag{4.8}$$

When the horizontal aggregation have been completed then vertical aggregation can be performed. The vertical aggregation is similar to horizontal aggregation but instead of using the cost $C_d(x)$ it uses the cost from horizontal aggregation, $C_d^H(x,y)$, and moves in the vertical directions, up and down. The two update rules for vertical aggregation and the combined vertical SWS is shown below:

$$C_d^U(x,y) = C_d^H(x,y) + \mu_U(x,y-1)C_d^U(x,y-1) \tag{4.9}$$

$$C_d^D(x,y) = C_d^H(x,y) + \mu_D(x,y+1)C_d^D(x,y+1) \tag{4.10}$$

$$C_d^V(x,y) = C_d^U(x,y) + C_d^D(x,y) \tag{4.11}$$

The cost from vertical aggregation, $C_d^V(x,y)$, will be influenced by the cost from pixels belonging to the same object as pixel at (x,y) in greater or lesser degree. This cost can be used for minimization along disparity estimates with a winner take all approach. For occlusion perform a left-right cross check e.i. run the algorithm with the reference and target image changed around.

Through testing α in equation 4.2 (p. 24) is chosen to be 0.4, the windows size of the census transform is chosen to be 3×3 and σ in equation 4.4 (p. 24) is chosen to be 38.1. These values have been determined through empirical observation.

EEPSM psuedo code

Input:
left image: I_l
right image: I_r
disparity estimate: d
Output:
filtering output: q
Steps:

1. $C_d^{SAD} = f_{SAD}(I_l, I_{r,d})$
   $C_d^{CENSUS} = f_{Ham}(f_{CENSUS}(I_l), f_{CENSUS}(I_{r,d}))$
2. $C_d = \alpha C_d^{SAD} + (1-\alpha) C_d^{CENSUS}$
3. $\mu_D = f_{perme}(I_l, down)$
   $\mu_U = f_{perme}(I_l, up)$
   $\mu_L = f_{perme}(I_l, left)$
   $\mu_R = f_{perme}(I_l, right)$
4. $C^L = f_{SWS}(C_d, left)$
   $C^R = f_{SWS}(C_d, right)$
5. $C^H = f_{SWS}(C^L, C^R, horizontal)$
6. $C^U = f_{SWS}(C_d, up)$
   $C^D = f_{SWS}(C_d, down)$
7. $C^V = f_{SWS}(C^U, C^D, vertical)$

4.2 Fast Cost-Volume Matching (FCV):

This algorithm is described in [10]. As the algorithm in section 4.1 (p. 24) this algorithm calculates an initial cost and then aggregate this cost. The initial cost for this algorithm

is also a combination of the sum of absolute differences and another cost. Instead of a cost based on census transform and hamming distance this algorithm uses difference in the gradient along the horizontal axis. The following equations show the different cost function for this algorithm and equation 4.14 shows the combined initial cost. First the SAD cost is calculated and this equation is similar to equation 4.1 (p. 24).

$$C_d^{SAD}(x,y) = \sum_{i=1}^{3} |I_l(x,y,i) - I_r(x+d,y,i)| \tag{4.12}$$

Then the gradient cost is calculated.

$$C_d^{Grad}(x,y) = \nabla_x I_l^g(x,y) - \nabla_x I_r^g(x+d,y) \tag{4.13}$$

Where ∇_x is the gradient along the x-axis and I_l^g and I_r^g is the grayscale version of each image. When these cost values have been calculated they can be combined to a single cost value:

$$C_d(x,y) = \alpha \cdot C_d^{SAD}(x,y) + (1-\alpha) \cdot C_d^{Grad}(x,y) \tag{4.14}$$

When the initial cost has been found it will be aggregated. The fast cost-volume algorithm will aggregate the cost values using a guided image filter.

$$C'_d(x_i,y_i) = \sum_{x_j,y_j} W_{(x_i,y_i),(x_j,y_j)}(I) C_d(x_j,y_j) \tag{4.15}$$

Where $C'_d(x_i,y_i)$ is the aggregated cost in pixel x_i, y_i with disparity estimate, d, x_j, y_j is pixels in a square window around x_i, y_i, and $W_{(x_i,y_i),(x_j,y_j)}(I)$ is the filter weights based on a guidance image, I. Section 4.2.1 will describe how the filter weights are found.

This aggregate cost can then be used for minimization along the disparity estimates with a winner takes all approach. For finding occlusion a left-right cross check will also be used here.

4.2.1 Guided Image Filter

This filter is described in [8] and [9] and this section simply summarizes their findings.
To describe the guided image filter a standard linear translation-variant filtering process is defined:

$$q_i = \sum_j W_{i,j}(I) p_j \tag{4.16}$$

Where i and j are pixel indexes, q the filter output, $W_{i,j}(I)$ is a filter kernel which is function of a guidance image I, and p is a input image.

The guided image filter is defined as a linear model between a guidance image, I, and a filtering output, q:

$$q_i = a_k I_i + b_k, \forall i \in \omega_k \tag{4.17}$$

where i and k is pixel indexes, ω_k is a square window centered at k, and a_k and b_k is linear coefficients which are assumed to be constant in the window, ω_k.
How to determine a_k and b_k is described in [9] and the solution is given by the following equations:

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon} \tag{4.18}$$

$$b_k = \bar{p}_k - a_k \mu_k \tag{4.19}$$

Where $|\omega|$ is the number of pixels in the window ω_k, μ_k is the mean of I in window ω_k, $\bar{p}_k$ is the mean of input image p in window ω_k, σ_k^2 is the variance of I in the window and ϵ is a regularization parameter which will penalize large a_k.

With a_k and b_k determined the filter output can be calculated:

$$q_i = \frac{1}{|\omega|}\sum_{k|i\in\omega_k} (a_k I_i + b_k) \tag{4.20}$$

$\sum_{k|i\in\omega_k} a_k = \sum_{k\in\omega_k} a_k$ because of symmetry in the square window and then the equation can be rewritten as

$$q_i = \bar{a}_i I_i + \bar{b}_i \tag{4.21}$$

Where $\bar{a}_i$ and $\bar{b}_i$ are the average coefficients for all windows that overlaps the pixel i and are expressed as $\bar{a}_i = \frac{1}{|\omega|}\sum_{k\in\omega_k} a_k$ and $\bar{b}_i = \frac{1}{|\omega|}\sum_{k\in\omega_k} b_k$.

The guided image filter is used for its edge preserving property. The edge preserving property can be explained with the case where $I = p$ then:

$$a_k = \frac{\sigma_k^2}{\sigma_k^2 + \epsilon} \tag{4.22}$$

$$b_k = (1 - a_k)\mu_k \tag{4.23}$$

And if $\epsilon = 0$ then $a_k = 1$ and $b_k = 0$ but if $\epsilon > 0$ then two cases can occur. If the pixel is in an area where I have a high variance in the window ω_k then $\sigma_k^2 \gg \epsilon$ and this results in $a_k \approx 1$ and $b_k \approx 0$. Instead if the pixel is in an area where I is flat in the window ω_k then $\sigma_k^2 \ll \epsilon$ and this results in $a_k \approx 0$ and $b_k \approx 1$.
When these values are averaged then if the pixel are in a high variance area then the output is $q \approx p$ and if it instead is in a flat area the output is the average of surrounding pixels $q \approx p$.

FCV - grayscale - psuedo code

Input:
left image: I_l
right image: I_r
disparity estimate: d
radius: r
epsilon: ϵ
Output:
filtering output: q
Steps:

1. $C_d^{SAD} = f_{SAD}(I_l, I_{r,d})$
   $C_d^{GRAD} = f_{GRAD}(I_l, I_{r,d})$
2. $C_d = \alpha C_d^{SAD} + (1-\alpha) C_d^{GRAD}$
3. $\mu_I = f_{mean}(I_l)$
   $\mu_p = f_{mean}(C_d)$
   $\rho_{II} = f_{mean}(I_l \cdot I_l)$
   $\rho_{Ip} = f_{mean}(I_l \cdot C_d)$
4. $\sigma_I = \rho_{II} - \mu_I \cdot \mu_I$
   $cov_{Ip} = \rho_{Ip} - \mu_I \cdot \mu_p$
5. $a = cov_{Ip}/(\sigma_I + epsilon)$
   $b = \mu_p - a \cdot \mu_I$
6. $\mu_a = f_{mean}(a)$
   $\mu_b = f_{mean}(b)$
7. $q = \mu_a \cdot I_i + \mu_b$

The guided image filter described in this section and shown in the FCV psuedo code above is using grayscale images. To use color images then some changes have to be added to the guided image filter. Equation 4.24 will then become:

$$q_i = \mathbf{a}_k^T \mathbf{I}_i + b_k, \forall i \in \omega_k \tag{4.24}$$

Where $\mathbf{I}_i$ is a 3×1 color vector and $\mathbf{a}_k$ is 3×1 coefficient vector. Then the guided image filter will be:

$$\mathbf{a}_k = (\Sigma_k + \epsilon U)^{-1} \left(\frac{1}{|\omega|} \sum_{i \in \omega_k} \mathbf{I}_i p_i - \mu_k \bar{p}_k \right) \tag{4.25}$$

$$b_k = \bar{p}_k - \mathbf{a}_k^T \mu_k \tag{4.26}$$

$$q_i = \bar{\mathbf{a}}_k^T \mathbf{I}_i + \bar{b}_i \tag{4.27}$$

where σ_k is the 3×3 covariance matrix of $\mathbf{I}$ in window, ω_k and U is the 3×3 identity matrix.

FCV - color - psuedo code

Input:
left image: I_l
right image: I_r
disparity estimate: d
radius: r
epsilon: ϵ
Output:
filtering output: q
Steps:

1. $C_d^{SAD} = f_{SAD}(I_l, I_{r,d})$
   $C_d^{GRAD} = f_{GRAD}(I_l, I_{r,d})$
2. $C_d = \alpha C_d^{SAD} + (1-\alpha) C_d^{GRAD}$
3. $\mu_{I,r} = f_{mean}(I_{l,r})$
   $\mu_{I,g} = f_{mean}(I_{l,g})$
   $\mu_{I,b} = f_{mean}(I_{l,b})$
   $\mu_p = f_{mean}(C_d)$
   $\mu_{Ip,r} = f_{mean}(I_{l,r} \cdot C_d)$
   $\mu_{Ip,g} = f_{mean}(I_{l,g} \cdot C_d)$
   $\mu_{Ip,b} = f_{mean}(I_{l,b} \cdot C_d)$
4. $\sigma_{I,r,r} = f_{mean}(I_{l,r} \cdot I_{l,r}) - \mu_{I,r} \cdot \mu_{I,r}$
   $\sigma_{I,r,g} = f_{mean}(I_{l,r} \cdot I_{l,g}) - \mu_{I,r} \cdot \mu_{I,g}$
   $\sigma_{I,r,b} = f_{mean}(I_{l,r} \cdot I_{l,b}) - \mu_{I,r} \cdot \mu_{I,b}$
   $\sigma_{I,g,g} = f_{mean}(I_{l,g} \cdot I_{l,g}) - \mu_{I,g} \cdot \mu_{I,g}$
   $\sigma_{I,g,b} = f_{mean}(I_{l,g} \cdot I_{l,b}) - \mu_{I,g} \cdot \mu_{I,b}$
   $\sigma_{I,b,b} = f_{mean}(I_{l,b} \cdot I_{l,b}) - \mu_{I,b} \cdot \mu_{I,b}$
   $cov_{Ip,r} = \mu_{Ip,r} - \mu_{I,r} \cdot \mu_p$
   $cov_{Ip,g} = \mu_{Ip,g} - \mu_{I,g} \cdot \mu_p$
   $cov_{Ip,b} = \mu_{Ip,b} - \mu_{I,b} \cdot \mu_p$
5. $\Sigma = f_\Sigma(\sigma_{I,r,r}, \sigma_{I,r,g}, \sigma_{I,r,b}, \sigma_{I,g,g}, \sigma_{I,g,b}, \sigma_{I,b,b})$
   $cov_{Ip} = [cov_{Ip,r}, cov_{Ip,g}, cov_{Ip,b}]$
   $a = cov_{Ip} \cdot f_{inv}(\Sigma + epsilon \cdot U)$
   $b = \mu_p - a_r \cdot \mu_{I,r} - a_g \cdot \mu_{I,g} - a_b \cdot \mu_{I,b}$
6. $\mu_{a,r} = f_{mean}(a_r)$
   $\mu_{a,g} = f_{mean}(a_g)$
   $\mu_{a,b} = f_{mean}(a_b)$
   $\mu_b = f_{mean}(b)$

7. $q = \mu_{a,r} \cdot I_{i,r} + \mu_{a,g} \cdot I_{i,g} + \mu_{a,b} \cdot I_{i,b} + \mu_b$

Through testing α in equation 4.14 (p. 27) is chosen to be 0.11, the windows size of the guided image filter is chosen to be 19×19 and ϵ in equation 4.25 (p. 29) is chosen to be 0.0001. These values have been determined through empirical observation.

4.3 Simulation and comparison

With each algorithm described the algorithms have been implemented in Python for simulation and comparison. The code for FCV is inspired by example matlab code from [10]. The code for EEPSM is written by us using the description in this chapter. None of the Python implementations have been optimized further to ensure that run on multiple cores etc. It should also be noted that the simulations are using floating-point values.

The simulation were performed on a MacBook Pro, Retina, 15-inch (mid 2015). This machine have the following specifications.

- OS: OS X El Capitan - version 10.11.6
- Processor: Intel©Core^{TM} i7-4770HQ
- Memory: 16 GB 1600 MHz DDR3
- Python version: 2.7.12 | Anaconda 2.5.0
- Relevant Python packages: Numpy v1.10.4, matplotlib v.1.5.1

For testing data sets from [18] are used. The stereo pairs are: Tsukuba, Cones, Teddy and Motorcycle and they are shown on figure 4.3 (p. 32). More information about the data sets is seen in appendix B (p. 77).

The runtime and stereo matching quality results from the simulations are presented in table 4.1 (p. 34) and table 4.2 (p. 35) while figures 4.4 (p. 33) to 4.7 (p. 34) shows resulting disparity maps. The results are discussed starting by discussing the visual result for each test set. Afterwards the calculated results, i.e run-time and number of false estimates, are discussed

Looking at the Tsukuba test results on figure 4.4 (p. 33) the FCV result seems better than the EEPSM result. The background seems more smooth on EEPSM result compared to the FCV result. Looking at the edges of objects FCV performs better in this test set. This is especially seen with the lamp. The silhouette of the lamp is sharper on the FCV result than on the EEPSM result and the sticks holding the lamp is missing in some areas in the EEPSM result.

(a) Tsukuba [15]

(b) Cones [16]

(c) Teddy [16]

(d) Motorcycle [17]

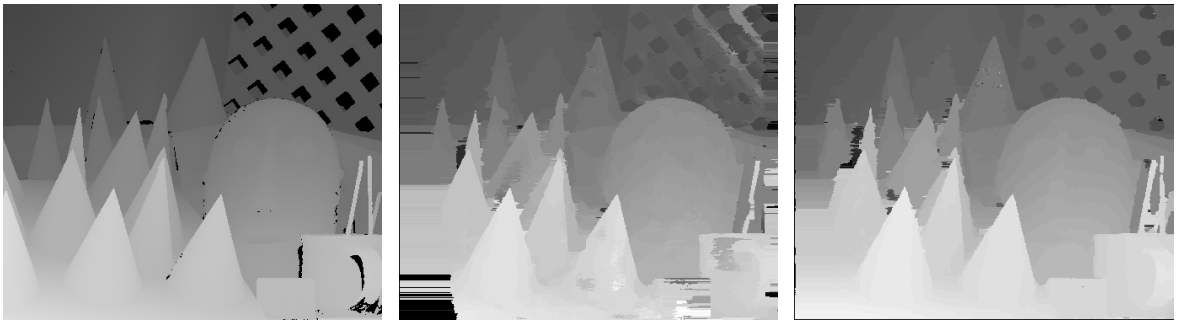
Figure 4.3: Middlebury data set - left images [18]

Looking at the Cones test results on figure 4.5 (p. 33) the EEPSM result seems better than the FCV result. The border occlusion filling in the left side are filled better on the EEPSM result. The occlusions are filled using the same method in both algorithms so any differences will come from the accepted matching results near the occluded area. Looking at the cones in the FCV result some of them have areas which jump in disparity value while the cones in the EEPSM result seems smooth. Some of the occlusions near the cones introduces more errors in the EEPSM result when compared to the FCV result. Looking at the box and cup in the lower right corner the result from EEPSM is better. The cup and box are smooth in EEPSM whereas in FCV there is some noise and in the EEPSM result, more of the sticks in the cup is found than in the FCV result. At last looking at the crisscross area in the right side of the background the EEPSM result is better. The holes in the surface are sharper in the FCV result but it have some errors when getting closer to the upper right corner whereas in the EEPSM result the whole surface is smooth.

Looking at the Teddy test results on figure 4.5 (p. 33) the EEPSM result seems better than the FCV result. Again the border occlusion in the left side is filled better in the EEPSM re-

(a) Ground truth **(b)** FCV **(c)** EEPSM

Figure 4.4: Tsukuba [15]

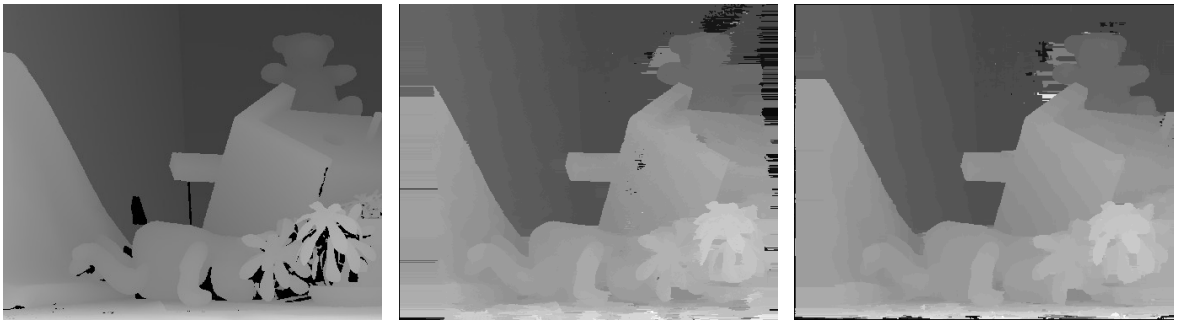


(a) Ground truth [16] **(b)** FCV **(c)** EEPSM

Figure 4.5: Cones [16]

sult when compared to FCV. Both seems to have equal results when comparing the teddy in front and the chimney on the house. The roof in the FCV result have an area with false disparities and the edge of the roof near the gable is not as sharp as in the EEPSM result. The area behind/around the teddy on top of the house have errors in both results. The area is prone to giving errors due to the repetitive texture on the wall. The area to the left of the teddy is better in the FCV result while the area to the right is better in the EEPSM result.

Looking at the Motorcycle test results on figure 4.5 the EEPSM result seems better than the FCV result. Once again the border occlusions have been filled better in the EEPSM algorithm. Looking at the ground, it is very smooth in the EEPSM result whereas in the FCV result large areas of the ground have errors like the lower left corner of the image. Looking at the background the FCV result has a large area with wrong disparity values while EEPSM seems smooth. The bench in the background is captured better in the FCV result where the holes and the silhouette are sharper when compared to the EEPSM result. Looking at the shelf unit in the background the FCV result has some areas with errors. Looking at the motorcycle is can be noticed that holes in the wheels and around the motor are present in either result. The side mirrors are captured better in the EEPSM result



(a) Ground truth [16] **(b)** FCV **(c)** EEPSM

Figure 4.6: Teddy [16]

(a) Ground truth [17] **(b)** FCV **(c)** EEPSM

Figure 4.7: Motorcycle [17]

where they are sharper and one of the side mirrors isn't present in the FCV result. The rest of the motorcycle are more smooth in the EEPSM result while FCV have small areas with errors.

Image	Resolution	FCV	EESPM
Tsukuba	384×288	180 s	86 s
Teddy	450×375	542 s	245 s
Cones	450×375	559 s	260 s
Motorcycle	741×497	1434 s	625 s

Table 4.1: Run time for different stereo pairs

Table 4.1 shows the runtime for each algorithm process each test set. From this, it is seen that EEPSM is more computationally efficient when compared with FCV. EEPSM is more than twice as fast as FCV at processing each test set. The main difference in runtime comes from the aggregation of the initial cost values. The SWS in EEPSM is less complex than the guided image filter in FCV.

Image	Resolution	FCV	EESPM
Teddy	450×375	8.4 %	5.8 %
Motorcycle	741×497	14.3 %	8.3 %

Table 4.2: Percentage false disparity estimates

Table 4.2 shows the number falsely estimated pixels. As seen the EEPSM algorithm performs better in this area too. One of the main contributors to the difference in percentage of errors seems to be the errors near borders and the ground in the motorcycle test set.

4.4 Theoretical Complexity

The simulations of the algorithms might have unknown factors which can affect the performance e.g. the programmers knowledge of the algorithms and programming skills. To ensure a more precise estimation of the performance of the algorithms the theoretical computational complexity have been calculated.

Table 4.3 shows the number of addition, multiplications, divisions and comparisons used per pixel.
It should be noted that the exponential function in the EEPSM algorithm is not a simple function to implement in hardware therefore it will be approximated here with a power series:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \tag{4.28}$$

In section 7.4.1 (p. 56) the challenges of implementing exponential function in hardware is discussed and from this section it is found that 23 terms of the power series is needed to achieve an approximation error ≤ 0.001 for values in the interval $[-\frac{255}{38.1}; \frac{0}{38.1}]$. The power series approximation with 23 terms is used in the calculation of the theoretical complexity instead of the exponential function. From the table it is noticed that the FCV requires a

	type	add/subtract	multiplication	division
EEPSM	C	$8768 \cdot N$	$5460 \cdot N$	0
	G	$7372 \cdot N$	$3133 \cdot N$	0
FCV	C	$296031 \cdot N$	$63630 \cdot N$	$303 \cdot N$
	G	$104838 \cdot N$	$16059 \cdot N$	$303 \cdot N$

Table 4.3: Complexity of each algorithm, where N is the number of pixels

lot more operations per pixel when compared to EEPSM. This is mainly due to the guided image filter which have a window size of 19×19 which results in a lot of addition and

multiplication for each single result. It is also seen that with EEPSM the change between color and grayscale doesn't reduce the number of operations by a large factor while with FCV this change reduces the number of operations by a lot. This is due to the EEPSM algorithm only using color during initial cost and finding permeability weights while in FCV the use of color makes the guided image filter require far more operations.

I/O communication can also have a influence on the performance hence the memory reads and writes for each algorithm have also been calculated and the result is shown in table 4.4. It should be noted that when calculating reads and writes for the FCV that the mean function is assumed to be a running window.

	type	reads	writes
EEPSM	C	$6402 \cdot N$	$2731 \cdot N$
	G	$5778 \cdot N$	$2731 \cdot N$
FCV	C	$9740 \cdot N$	$5766 \cdot N$
	G	$4861 \cdot N$	$3033 \cdot N$

Table 4.4: Memory reads and writes used by each of the algorithms, where N is the number of pixels

From table 4.4 it is seen that the change from color to grayscale in EEPSM doesn't change the number of reads by much and the number of writes remains the same. This is due to the EEPSM algorithm only using color in a few functions. On the other hand, it is noticed that the FCV algorithm reduces the number of reads and writes by much. This is because of color being used in a lot of functions in the guided image filter. When using color EEPSM uses less I/O communication when compared to FCV while when using grayscale FCV uses fewer reads than the EEPSM algorithm but still uses more writes. From the results in this table EEPSM generally performs better.

4.5 Choosing an algorithm

One of the algorithms described in this chapter should be chosen for further implementation on a FPGA. This choice is based on the results from section 4.3 (p. 31) and section 4.4 (p. 35).

Starting with the computational efficiency table 4.1 shows that the Python implementation of the EEPSM algorithm is more than twice as fast as the FCV algorithm (run-time is reduced by 52-56%). To ensure that these results are not affected by unknown factor e.g. the programmer's programming skills, the theoretical computational complexity is also used. Table 4.3 (p. 35) and 4.4 shows the number of operations and I/O communication per pixel. This shows the same result. The reason that EEPSM is faster is due to the ag-

gregation each uses. The SWS requires far fewer operations when compared to the guided image filter especially when using color.

The quality of the resulting disparity map should also be taken into consideration when choosing an algorithm. The visual inspection of the results shows that EEPSM algorithm generally results in better disparity maps in all except for one of the test sets which is the Tsubuka test set. But this test set is the least important since it is the set with the lowest resolution and the images for a prototype will have a much higher resolution. In some situations, FCV gives a bit better result than the EEPSM algorithm i.e. the holes in the bench in figure 4.7 (p. 34). Table 4.2 (p. 35) shows that the EEPSM algorithm results in disparity maps which are closer to the ground truth when compared to the FCV algorithm results (between 31-42% fewer false estimates).

Referring to the cost function from chapter 3 the two main factors to consider when choosing an algorithm is time and stereo matching quality with time having the highest priority. Considering these two factors and the results which are summarized in this section the EEPSM algorithm is chosen.

4.6 Wrap-up

Two edge preserving stereo vision algorithms have been described. The algorithms have been simulated using Python and floating point. To support the results from the simulations their theoretical complexities has been calculated. The results show that the EEPSM algorithm has a lower execution time and matches better than the FCV algorithm. Considering these results and the cost function in chapter 3 the EEPSM algorithm is chosen to be implemented on a FPGA.

Chapter 5

Platform analysis

In this chapter the hardware platform provided for this project is described. For this project, HSA systems have provided a Zedboard Development Board [4]. This board is used for this project since it contains a Zynq Z-7020 SoC from Xilinx, which HSA Systems uses for multiple products.

5.1 Platform trade-off

When designing embedded systems the choice of platform is important. Different platform exists each with their own advantages and disadvantages. Figure 5.1 illustrates the relation between design time and flexibility for some general platform types, inspired by [1].

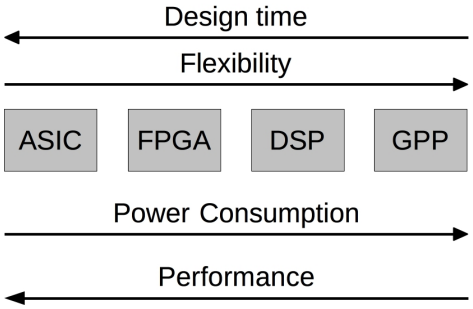


Figure 5.1: Relation between design time and flexibility for general platform types.

From figure 5.1 it is seen that there is a clear trade-off between a general-purpose processor (GPP) and an application specific integrated circuit (ASIC). The other platforms fall in between these two.

A GPP is a popular choice due to its short design time and high flexibility. It is optimized for data manipulation, control flow, and sequential performance. However in the area of signal processing applications usually have a high number of computations and a high level of inherent parallelism. This makes a GPP not the best platform for such applications. For these applications, more specialized platforms can be used such as a DSP or FPGA. A DSP includes optimized computational units which can be used for real-time signal processing. But DSPs require the designer to have a better knowledge of the specific DSP platform when designing for it. The DSP is still limited by the number of computational units on the platform. This is where an FPGA is strong. An FPGA consists of a high number of logic gates and interconnections between them and with this specialized architectures can be designed which can completely utilize the inherent parallelism in an algorithm.

In later years new platforms have emerged which combines some the general platforms. This is clearly seen with platforms such as Zynq Z-7020 which is a GPP combined with an FPGA.

HSA Systems has chosen that the platform for this project will be a Zedboard which contains a Zynq Z-7020 since this chip is used in other of the company's products. This project will mainly only use the programmable logic of the Zynq SoC since the project description states that the focus is on an FPGA implementation.

5.2 Zynq Z-7020

The Zynq SoC contains an ARM® Processing system and 7 series programmable logic (FPGA). Figure 5.2 (p. 41) shows an overview of the Zynq Z-7000 architecture. From this figure, it can be noticed that the *programmable logic* is located at the bottom and all the connection to the rest of the system is seen. In the upper right of the overview, the ARM® cores are located. In this project, the GPP part of the SoC will be used for OS and likewise assignments while the algorithm will mostly be implemented in the programmable logic.

Part	**Quantity**
Programmable logic cells	85,000
Look-Up Tables	53,200
Flip-flops	106,400
Block Ram	4.9 MB
Programmable DSP slices	220

Table 5.1: Programmable logic - Zynq 7020 [24]

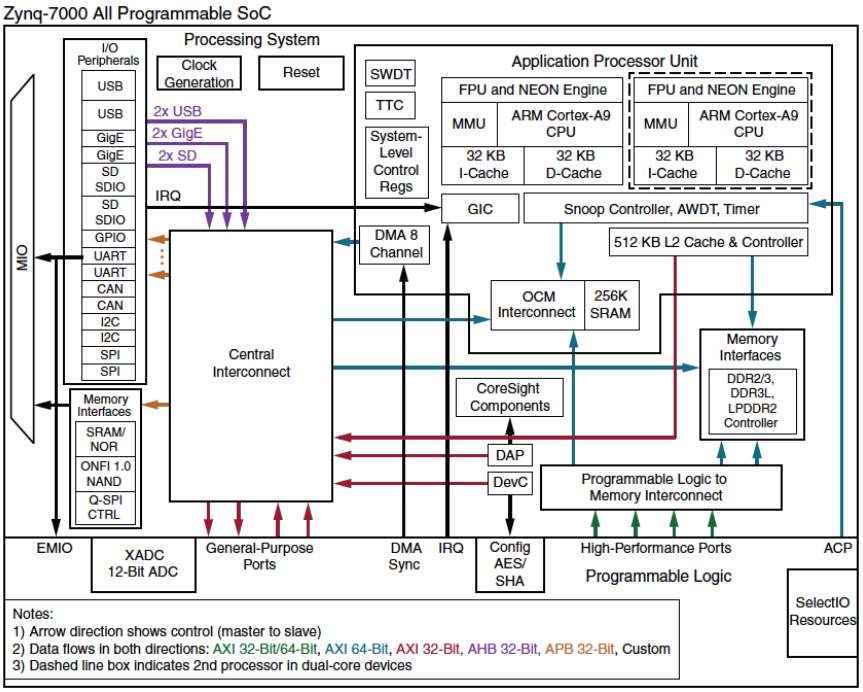


Figure 5.2: Architectural Overview [24]

In table 5.1 (p. 40) the logic elements available in the Zynq Z7020 are listed. Table 5.2 shows some additional specifications for the Zedboard platform. This information will be used when designing the hardware architecture in chapter 7 (p. 49).

Part	Quantity
Memory - DDR3	512 MB
Memory - QSPI	256 MB
Oscillator - PS	33.333 MHz
Oscillator - PL	100 MHz

Table 5.2: Additional specifications for Zedboard [4]

5.3 Hardware utilization

In this section, the hardware utilization of the Xilinx Vivado software and the Zedboard is explored. The software used is the Vivado HL Design Edition 2016.2. To get a simple approximation of the hardware utilization some simple systems have been generated in software, synthesized and implemented. Simple systems with a specified functional unit

have been created using the Block Design system in Vivado. 100 copies of the specified functional unit were added. The reasoning for 100 copies is to ensure that the contribution

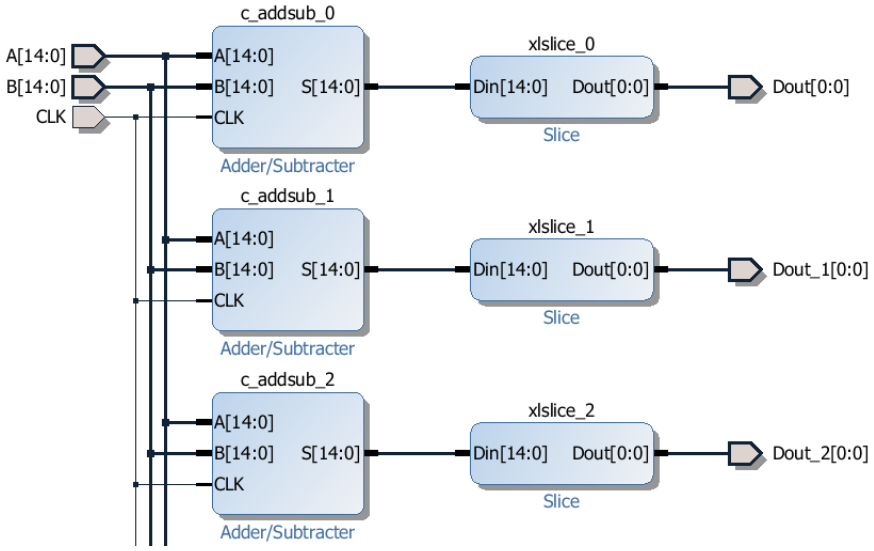


Figure 5.3: Example of block design in vivado

from control signals and wiring is minimized. With a block design generated it can be synthesized. In Vivado there are 2 steps. The first step is *synthesize* where the software figures out which hardware is needed for the system and then the next step is *implementation* where the software optimize the hardware use for the platform it is implemented on. To complete the *implementation* step the design have to fit the target hardware so the design must not exceed the number of available I/O ports etc. When generating the block design IP blocks from Xilinx were used and for most FUs inputs and outputs of 16 bits were used. Each output needs to be connected to its own port and this will result in the implementation step to fail since too many I/O ports are used. Connecting the outputs to temporary signals and not using these signals will result in the *implementation* step to optimize and remove all the adders since the output aren't used. Instead, a Slice IP can be used to strip every bit but MSB. This results in every block only having an output size of 1 bit and the I/O usage have been lowered enough for the *implementation* to succeed. Then the hardware utilization can be found in the software. These values are then divided by 100 to get a rough approximation of how much hardware each functional unit requires.
Appendix A (p. 73) describes more thoroughly the procedure for these tests and table 5.3 shows the result.
Looking at the results it is noticed that the multiplier implemented with LUT requires a lot of LUTs and FFs when compared to adder/subtracter while it only requires a single DSP if implemented using DSP48 elements. From this, it is decided that the 220 DSP48 elements should mainly be used for multiplication.

	LUT	FF	BRAM	DSP48
Adder	15	15	-	-
Adder/Subtracter	≈ 16	15	-	-
Subtract	15	15	-	-
Multiplier - LUT	352	36	-	-
Multiplier - DSP48	-	-	-	1

Table 5.3: Number of logic elements used in average for each FU

5.4 Wrap-up

The first section discusses the trade-off between platforms and which platform will be used in this project which is a Zedboard. The second section describes the specifications of the Zynq Z7020 and some additional important specification of the Zedboard. Lastly, the utilization of the hardware for different functional units using the Vivado software was analyzed. From this, it is noticed that the limited number of DSP48 elements should mainly be used for multiplications since it uses more logic elements than additions and subtractions.

Chapter 6

Design methodology

This chapter will describe a method for traversing between the algortihm domain and architecture domain in the A^3 model described in section 1.6 (p. 4) and the red lines in figure 6.1 shows where this movement occurs.

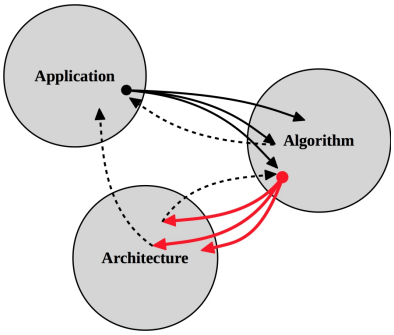


Figure 6.1: A^3 model with the movement from algorithm domain to architecture domain highlight

In [7] the Gajski-Kuhn Y-chart is described and it is illustrated on figure 6.2 (p. 46). This chart is a structured design method which can help organize the design processes for creating a dedicated hardware system. The chart consist of 3 domains: Behaviour, Structure, and Physical. The circles in the chart illustrates the different abstraction levels and following the arrows on the domain lines the abstraction levels increases.

The behavioral domain describes the functional behavior of the system. Going from the highest abstraction level to the lower abstraction level this domain starts being described by algorithms and at the lowest level the behavior is described with differential equations.

The structural domain describes the system in terms of components and their interconnection. Going from the highest abstraction level to the lower abstraction levels this domain is first described using larger components such as platforms or CPUs and at the lowest level

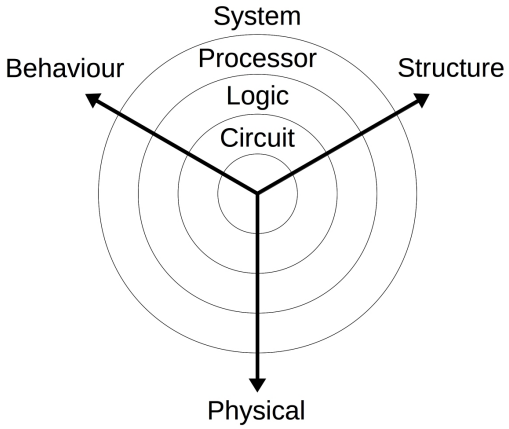
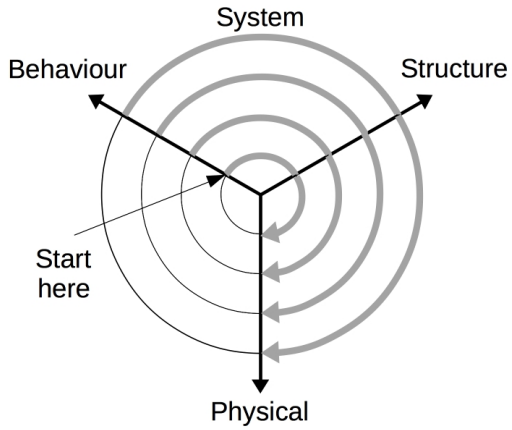


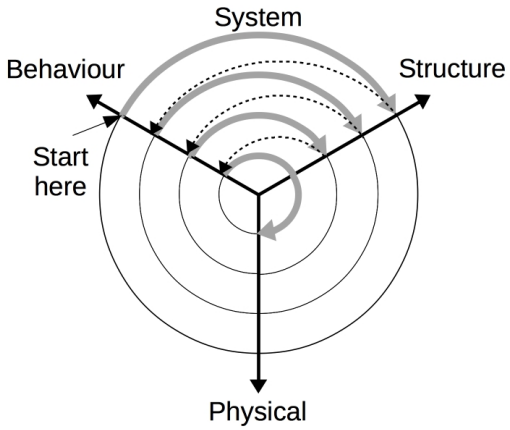
Figure 6.2: Illustration of the standard Gajski-Kuhn Y-chart [7]

it will be described with transistors.

The physical domain describes how the system is physically put together and is at the highest abstraction levels described using chips, boards etc. and at lower levels, it will be described using transistor layout.



(a) Illustration of the Gajski-Kuhn Y-chart showing the bottom-up methodology [7]



(b) Illustration of the Gajski-Kuhn Y-chart showing the bottom-up methodology [7]

Figure 6.3: Illustration of different methodologies

Using the Y-chart for structuring the design process can guide the design process through different design methodology. Two basic methodologies are bottom-up and top-down.

The bottom-up methodology starts at the lowest abstraction level and moves through the

3 domains designing the system. Then it goes up a level and uses the results from the lower level to design next level. The bottom-up methodology gives full control of even the smallest details and the final cost function is available earlier. But designing at the lowest abstraction level can quickly be overwhelming and time-consuming due to all the details available at the lowest level. The bottom-up methodology is shown on figure 6.3a (p. 46)

Top-down methodology instead starts at the highest abstraction level and designs the system going between the behavioral and structural domains and then go an abstraction level down and at the lowest level the system will be described in the physical domain. Since it starts at a higher abstraction level the design process is simpler and it is easier to optimize the system because amount of details to consider is less than at the lower abstraction levels i.e. it is easier designing a system connecting blocks/subsystems than it is connecting transistors. An issue with this methodology is that the exact cost function for the system is only available at the end of the design process when the design is finalized at the lowest abstraction level but is should be noticed that at higher abstraction levels an estimate of the cost function can be used. The top-down methodology is shown on figure 6.3b (p. 46).

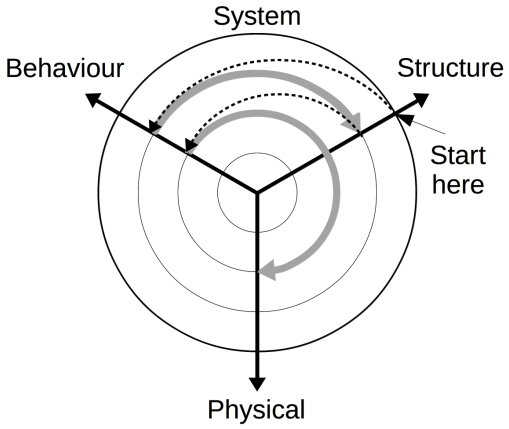


Figure 6.4: Illustration of the Gajski-Kuhn Y-chart showing the FPGA methodology [7]

Another methodology is the FPGA based methodology shown on figure 6.4 which is a variation of the top-down methodology. The methodology is based on the fabric of FPGA which consists of a large amount of Look-Up Tables (LUT). The top-down methodology is used at the system and processor abstraction levels where the processing and communication elements are expressed using LUTs. The design begins by mapping the application onto a platform and then custom components are expressed using LUTs. Once every component has been defined the whole design is flatten to LUTs and BRAMs and the tools provided by the FPGA supplier performs component placement and routing. This methodology have the same weaknesses as the normal top-down methodology and in addition to those weaknesses it is also unknown for the designer how the provided tools

maps and connects the elements.

The Y-chart has been used for many years and therefore is well known but technology have evolved and the Y-chart have become less applicable in some cases. As discussed in chapter 5 platforms for hardware and software co-design such as the Zynq platform have appeared. The Y-Chart is insufficient for these designs because the segregation of hardware and software is not naturally modeled on the chart. For these designs, new design models have been developed and one of these is the Rugby model [13]. The model is not described in this project since it will not be used. As discussed in chapter 5 this project will only focus on a FPGA design hence the GPP part of the Zynq platform is mostly ignored. Due to this the Y-chart and the FPGA methodology can and will be used for modeling the design for this project.

6.1 Wrap-up

The Y-chart and different design methodologies have been described and it is chosen to use the Y-chart and FPGA methodology to model the design for this project. If the GPP part of the Zynq were to be used in collaboration with the FPGA fabric another model such as the Rugby model should have been used but an FPGA implementation is the focus of this project and hence the Y-chart is used.

Chapter 7

Architecture design

This chapter will explore the architecture domain in the a^3 model described in section 1.6 (p. 4) and it is the domain marked on figure 7.1. The design process in this chapter starts by creating a block diagram of the chosen algorithm.

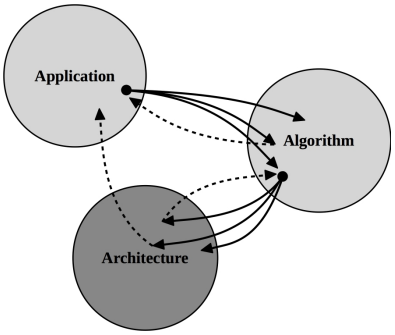


Figure 7.1: A^3 model with the architecture domain marked

A block diagram of the EEPSM algorithm is illustrated on figure 7.2 (p. 50). This block diagram is based on the pseudo code in section 4.1 (p. 26). The *init. cost* block contains step 1 and 2 of the pseudo code, the μ block contains step 3 of the pseudo code, the *horz. aggre* contains step 4 and 5, the *vert. aggre.* block contains step 6 and 7 and the *minimization* block will contain the minimization of the cost values to find the disparity values. This block diagram is created to divide the system into smaller subsystems. This is equal to go down an abstraction level i.e. synthesize in the Structure domain of the Y-Chart. This division into subsystems help limit size of systems which the different analyses and design processes is used on.

To develop an FPGA hardware a finite state machine (FSM) and a data path are wanted. Before designing an FSM some processes and analyses should be performed. These processes are:

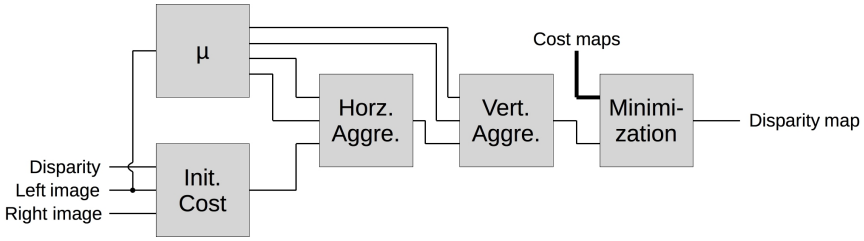
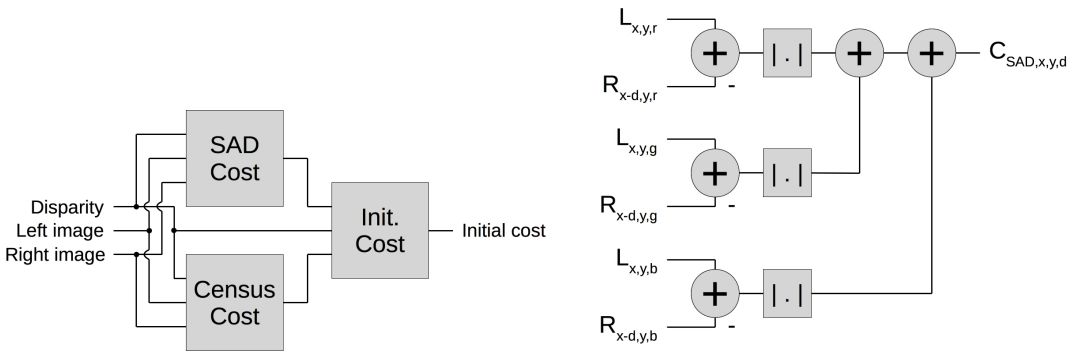


Figure 7.2: Block diagram of the EEPSM algorithm

- Parallelism analysis
- Allocation and scheduling
- Assignment

Due to time constraint for the thesis, a final implementation was not achieved but the design process is shown at a lower abstraction level with the *SAD cost* block which is a part of the *init. cost* block in figure 7.2.



(a) Init. Cost block diagram **(b)** Cost SAD block diagram

Figure 7.3: Block diagrams

7.1 Parallelism Analysis

The inherent parallelism of the system has been analyzed to find out which improvements can be made.

To find the parallelism in the system precedence graphs (PG) have to be created. With simple subsystems, the precedence graphs can be created by looking at the system from end to start and for each operator find out which signal is needed and have to be calculated before.

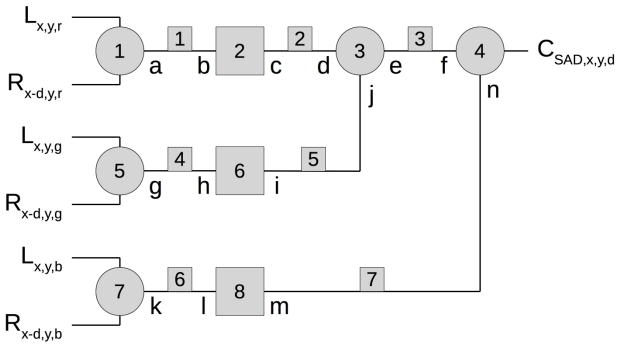


Figure 7.4: Synchronous data flow graph of the SAD cost block

Another method is to create a synchronous data flow graph (SDFG) and from the SDFG a matrix, $\underline{\underline{\Gamma}}$, can be created. This matrice expresses the relationship between the in- and outputs from each node in the SDFG and is called the *topology* matrix. An example of an SDFG is illustrated on in figure 7.4. In this example the topology matrix is then:

$$\underline{\underline{\Gamma}} = \begin{array}{c} \\ \text{arcs} \end{array} \overset{\text{nodes}}{\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix}} \tag{7.1}$$

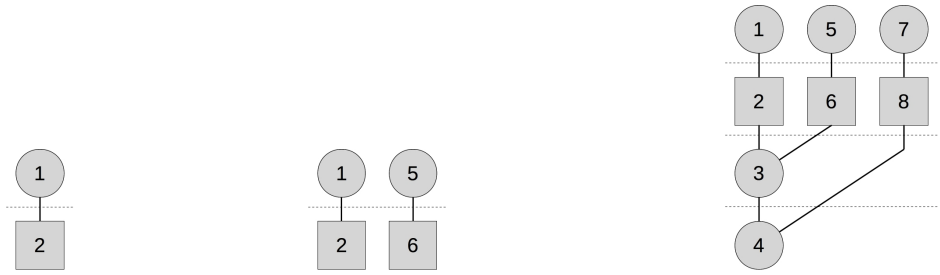
If $rank(\underline{\underline{\Gamma}}) \leq s - 1$ where s is the number of nodes then a positive integer vector $\underline{q}$ can be found such that $\underline{\underline{\Gamma}} \cdot \underline{q} = \underline{0}$. The resulting $\underline{q}$ is:

$$\underline{q} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{7.2}$$

This $\underline{q}$ vector expresses how many times each node have to be executed within a period.

With $\underline{\underline{\Gamma}}$ and $\underline{q}$ a periodic admissible sequential sequence (PASS) can be found. This tells a sequential sequence in which the nodes can be executed and with it, a periodic admissible

parallel sequence (PAPS) can be generated. To find the PASS generate a randomly ordered list of all the nodes, L. For each $\alpha \in L$ test if can be executed i.e. if data is available. If it can be executed then add it to the PASS, φ. This is repeated for each node and if every node have been scheduled in the PASS the number of times specified by $\underline{q}$ then continue to the next step.
For this example let $L = \{1,2,3,4,5,6,7,8\}$ then checking node 1 the data is available and is added to φ. After checking every node in L once, φ will be $\{1,2,5,6,7,8\}$. Since node 3 and 4 is missing in φ then repeat the process. After this iteration φ will be $\{1,2,5,6,7,8,3,4\}$. The PASS is only a sequential schedule but tells nothing about precedence relations and parallelism. For this, the precedence graph is used and it can be generated with the PASS found. Take one node at a time in the PASS and add it to the precedence graph and determine the precedence links. Figure 7.5 shows the progress towards a precedence graph.



(a) With only the 2 first nodes included

(b) With only the 4 first nodes included

(c) With all the nodes in the SAD cost block

Figure 7.5: Precedence graph of the SAD cost block

The first figure shows the precedence graph when only the two first nodes from φ. When the next two nodes are added it is noticed how these are added next to node 1 and 2 since they have no precedence links to those nodes. The last figure shows the final precedence graph which can be used as a PAPS.

With the precedence graph found the next step towards a hardware design can be fulfilled.

7.2 Allocation and Scheduling

To develop a Finite State Machine (FSM) we need to know which hardware is allocated and when each operation is scheduled. This enables us to define some states for the FSM. From section 7.1 some a precedence graph are found and this graph shows how many FUs can run in parallel.

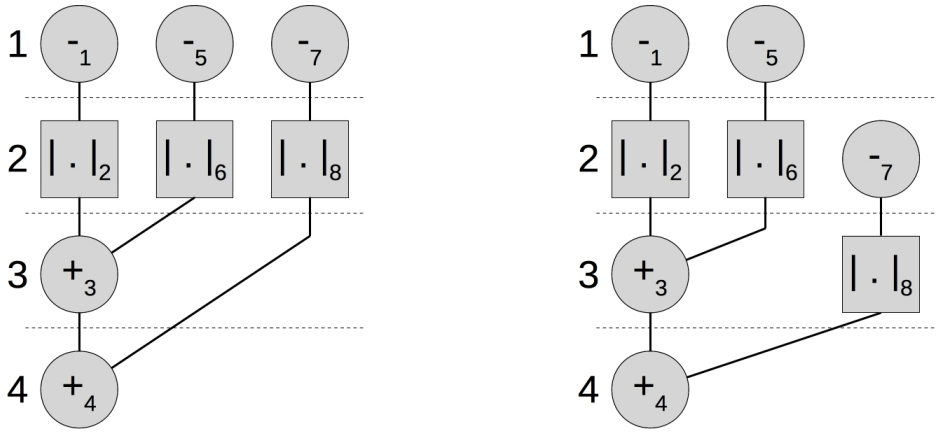
To create the FSM specification we need to introduce time into the precedence graphs to establish some states. To achieve this scheduling is used. There exist different methods for scheduling and some of these are:

- Resource Constrained (RC)
- Time Constrained (TC)

7.2.1 RC scheduling

For both methods the first step is to create as soon as possible (ASAP) and as late as possible (ALAP) schedules. Then for the RC scheduling a ready list is created. This list contains a list of operations which are ready for scheduling and sorted by mobility. The mobility, $M(op)$, expresses the difference between the states in which the operation, op, have been scheduled in ASAP and ALAP schedules, e.i. $S_{ALAP}(op) - S_{ASAP}(op)$.

Figure 7.6 shows an example of an ASAP and an ALAP schedule. As seen on the figure node 1-6 are part of the critical path and therefore mobility is 0 for each of these nodes. Nodes 7 and 8 both have a mobility of 1.



(a) Example of ASAP schedule **(b)** Example of ALAP schedule

Figure 7.6: Illustration of ASAP and ALAP schedules for SAD cost block

With mobility found for each node/operation then, a ready list can be generated. Add every ready node and then sort them by mobility. In the start the list would look like this: 1. node 1 ($M(-_1) = 0$), 2. node 5 ($M(-_5) = 0$) and 3. node 7 ($M(-_7) = 1$).

If the system has 2 ALU available then node 1 and 5 can be scheduled to state 1 since they are higher on the ready list and there is no free ALU for the remaining node. The scheduled nodes are removed and the ready list is updated with newly available nodes.

The list is still sorted by mobility and will now look like this: 1. node 2 ($M(|.|_2) = 0$), 2. node 6 ($M(|.|_6) = 0$) and 3. node 7 ($M(-_7) = 1$).

Then nodes 2 and 6 are scheduled into state 2 since they have lower mobility than the rest of the ready list. The list is updated again and will look like this: 1. node 3 ($M(+_3) = 0$) and 2. node 7 ($M(-_7) = 1$).

Nodes 3 and 7 are scheduled into state 3 and the list is updated again: 1. node 8 ($M(|.|_8) = 1$).

This is repeated until all nodes have been scheduled and the result is seen in figure 7.7. This schedule uses 1 states more than the ASAP and ALAP but it reduces the cost since it uses 1 less ALU compared to the ALAP and ASAP schedules.

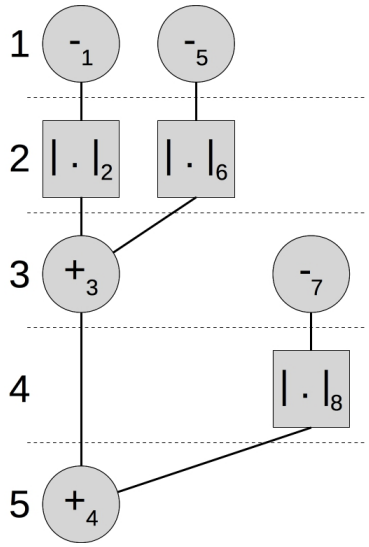


Figure 7.7: RC schedule of the SAD cost

With this scheduling the allocation is performed before scheduling since the scheduling is constrained by how many FUs have been allocated.

7.2.2 TC schedule

The RC schedule improves the area cost of the architecture while the TC intends to improve the computational performance of the implementation. First, a maximum number of states is decided and then ASAP and ALAP schedules are generated and mobility ranges are found. Then some probabilities are assigned to each operation. These probabilities express the probability for the specified operation to be scheduled in each state in its mobility range

e.g. node 7 in 7.6 (p. 53) have 1/2 probability for being scheduled in each of state 1-2 if the maximum number of states is kept at 4. Figure 7.8 shows the probabilities for each node in figure 7.6 (p. 53).

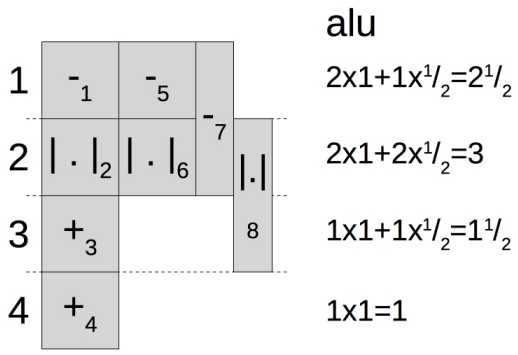


Figure 7.8: Probabilities for each operation in figure 7.6 (p. 53)

These probabilities can then be used to generate a force-directed schedule. With this scheduling the allocation of FUs is done after the schedule is found.

7.3 Assignment

When a schedule have been found and some hardware have been allocated the hardware can be assigned. During the assignment process, the nodes are assigned to the allocated FUs. A lifetime analysis can also be performed the allocate register and assign the variables to these registers.
For the example, this step is skipped and it is chosen to directly implement an FSM from the ALAP schedule and let the Vivado software optimize the design. The resulting RTL architecture is shown on figure 7.9. Due to time constraint the design process will stop at this point.

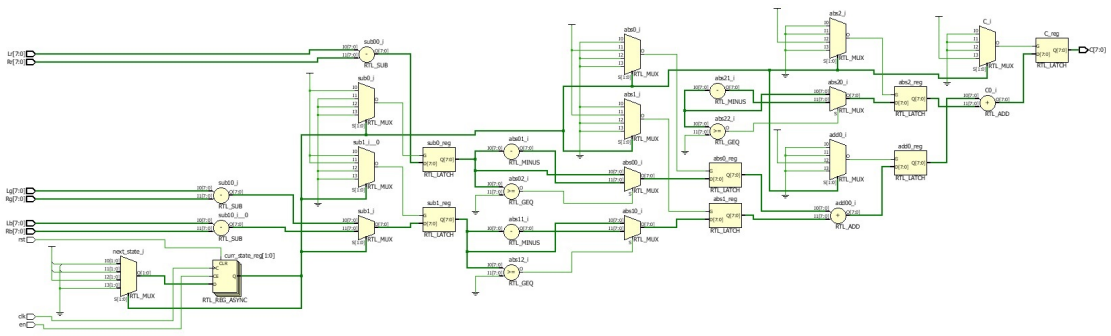


Figure 7.9: SAD cost - RTL schematic

7.4 Implementation challenges

Some challenges were noticed at the higher abstraction levels and these will be described in this section.

7.4.1 Exponential function

The exponential function is a high computational complex function which can be hard to implement. VHDL does not have an easy function to implement it and Xilinx Vivado does not have an IP core block with an exponential function. So another way to implement this function has to be found.

The exponential function can be defined by a power series:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \tag{7.3}$$

An approximation of the exponential function can be found by using a finite number of terms from power series definition. More terms results in a better approximation.

The input values for the exponential function will be in the range of $\left[-\frac{255}{\sigma}; \frac{0}{\sigma}\right]$ where σ through experiments have been chosen to be 38.1 so the values will be in the interval $[-6.69; 0]$.

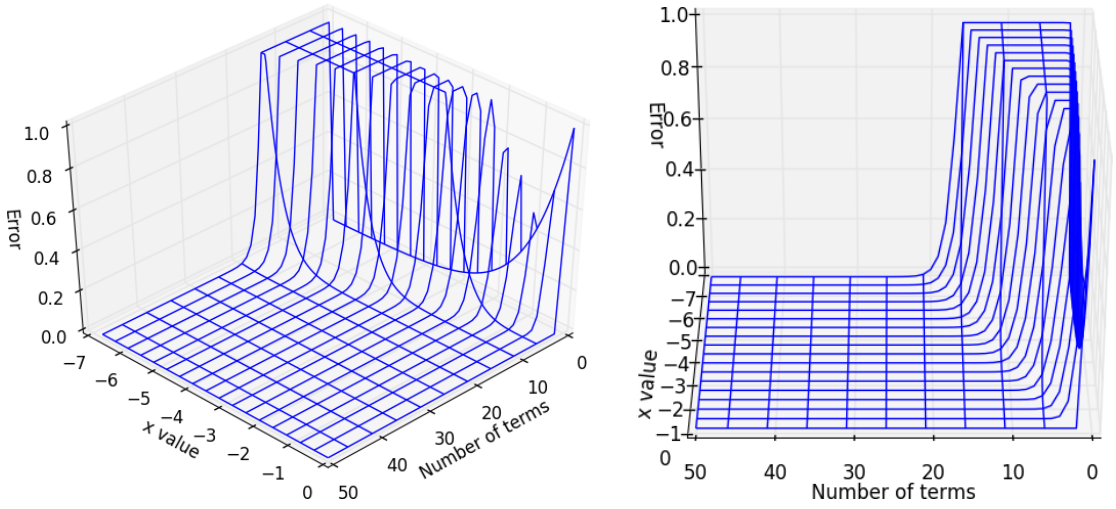


Figure 7.10: Approximation error with respect to x value and number of terms seen from two views. Error values have been limited to the range $[0; 1]$

Figure 7.10 shows the absolute error between the correct exponential value and the approximation with respect to the x value and number of terms used. It is noticed that a

higher absolute value of x requires a higher amount of terms i.e. an x value of -1 requires at least 7 terms to have an error ≤ 0.001 while an x value of -6 requires at least 21 terms to achieve same precision.

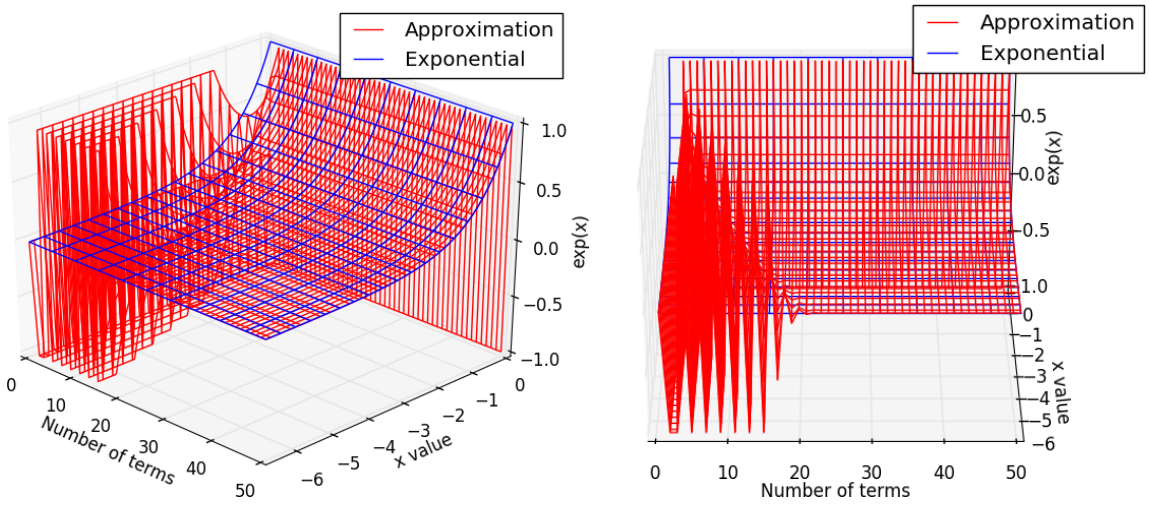


Figure 7.11: Exponential values for approximation and the correct value with respect to x value and number of terms seen from two views. The values have been limited to the range $[-1; 1]$

Figure 7.11 shows the result from the approximation (red surface) and the correct exponential values (blue surface). From this, it is noticed that the approximation at a low number of terms starts alternating between high positive and negative numbers when increasing the number of terms. The alternating positive and negative are due to the definition of x^n in equation 7.3 and x being negative, and the large difference between the approximation and the correct result at a low number of terms is due to x^n in the same equation starts out larger than the $n!$ but $n!$ grows quicker than x^n when increasing the number of terms used.

To ensure that the error is ≤ 0.001 at an x value of -6.69 then 23 terms have to be used and then the error is 0.00029. This power series could be implemented in hardware as a series of multipliers and adders. The $\frac{\cdot}{n!}$ of equation 7.3 can be implemented as multiplication with $\frac{1}{n!}$. But this implementation introduces an issue. Looking at the last term where $n = 23$ and using the x value -6.69 the term is: $(-6.69)^{23} \cdot \frac{1}{23!}$. This term alone will use 24 multipliers and the result from $(-6.69)^{23}$ requires 65 bits for representing the final result as an integer and $\frac{1}{23!}$ requires 75 bits for representing it as a fixed-point value. This seems unfeasible to implement in hardware so alternative implementations have to be found.

There exist different methods to implement an exponential function. It can be implemented using a CORDIC engine and there is an IP core block of CORDIC available through Vivado. A CORDIC engine can calculate hyperbolic and trigonometric functions using it-

erative operations working on one bit at a time. The exponential function can be expressed using hyperbolic functions:

$$e^x = \cosh x + \sinh \tag{7.4}$$

The strength of the CORDIC is that it requires a low amount of hardware since it is an iterative algorithm but looking at computation performance other methods are better [20]. One method which is better is using a look-up table containing the results. A look-up table can have a better computational performance but can require a high amount of memory [20]. The x value in e^x depends on the difference in intensity between two pixels hence the x value can only be 256 different values. A ROM containing these values needs address width of 8 bits. The only variable number which the exponential function depends on is the difference between intensity of two pixels. This difference can be used as the address to the memory cell containing the exponential value corresponding to that value i.e. if the difference is 3 then the memory cell at address 3 should contain $\exp(-3/\sigma)$. The exponential values can be calculated before and just be inserted into the look-up table. To ensure that the lowest result, which is $e^{-6.69} = 0.0012$, will not be 0 at least 11 bits are needed. A look-up table of that size seems feasible to implement in hardware and therefore this solution is chosen.

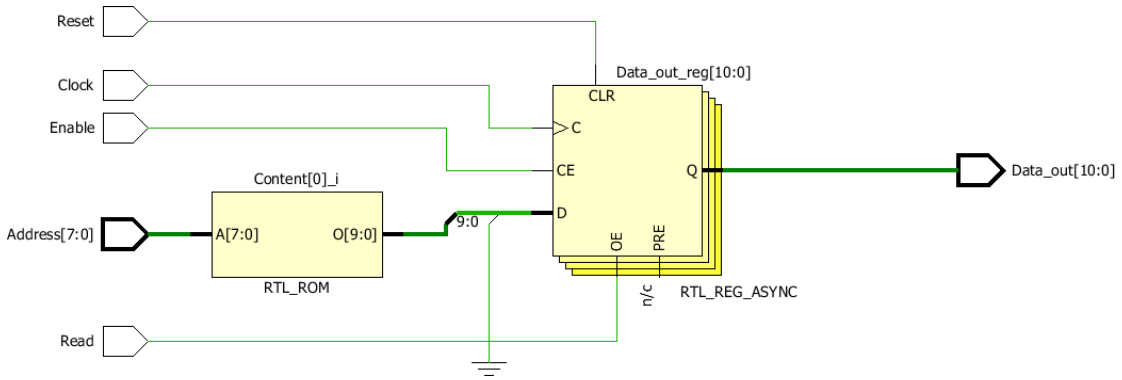


Figure 7.12: Schematic of ROM containing look-up table containing exponential values

A ROM containing the exponential values have been described in VHDL and synthesized for the Zedboard platform using Xilinx Vivado. Figure 7.12 shows a schematic of the implementation and table 7.1 (p. 59) shows how much hardware the implementation utilizes. From this, it is seen that a feasible implementation can be given with a look-up table.

7.4.2 Memory Issue in the EEPSM algorithm

The EEPSM algorithm has a challenge with large images and disparity ranges such as those in this project in that it requires a large amount of memory for containing the cost of each pixel at each disparity value. Considering all the variables in the pseudocode in

logic element	Utilization [·]	Utilization [%]
LUT	32	0.06
FF	11	0.01

Table 7.1: Hardware utilization for ROM containing exponential values

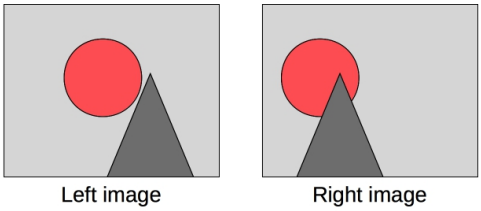
section 4.1 (p. 26) and a variable size of 1 byte, keeping all these variables in memory will require 13.8 GB.

Algorithm steps	1	2	3	4	5	6	7
Right image							
Left image							
SAD cost							
Cen. cost							
Combined cost							
Permeability weights							
Left aggregation							
Right aggregation							
Horizontal aggregation							
Top aggregation							
Bottom aggregation							
Vertical aggregation							

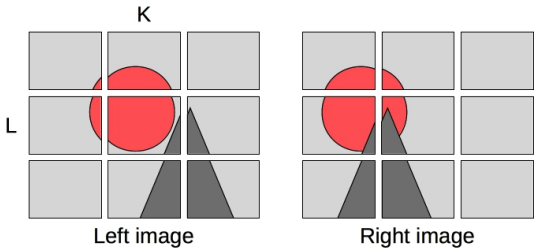
Table 7.2: Lifetime for variables in the EEPSM algorithm

Table 7.2 shows the lifetime of variables in the EEPSM algorithm. The algorithm steps in the table correspond to the steps in the pseudocode. From this, it can be seen that a maximum of 5 variables is needed to be kept in memory at the same time. This is at step 2 and this step requires 4.6 GB. It is not feasible to implement with this memory requirement since the Zedboard only has 512 MB DDR3 memory and 256 Quad-SPI Flash memory. The algorithm has to be changed to require less memory. Referring to the A3 model with this new knowledge the design process will go back to the algorithm domain and modify the algorithm to reduce memory usage. Referring to the Y-chart the design process will perform some iterations in the same abstraction level in the behavioral domain. Two methods to decrease the memory usage have been considered.

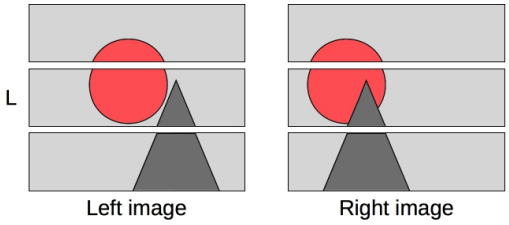
The first method suggests cutting the input images into smaller sub-images. This is illustrated on figure 7.13 (p. 60). Figure 7.13b (p. 60) shows an example where the images are divided along each axis and results in $K \times L$ sub-images. Performing the algorithm on one sub-image at a time reduces the memory usage by a factor proportional to $K \times L$ so in the



(a) Full images



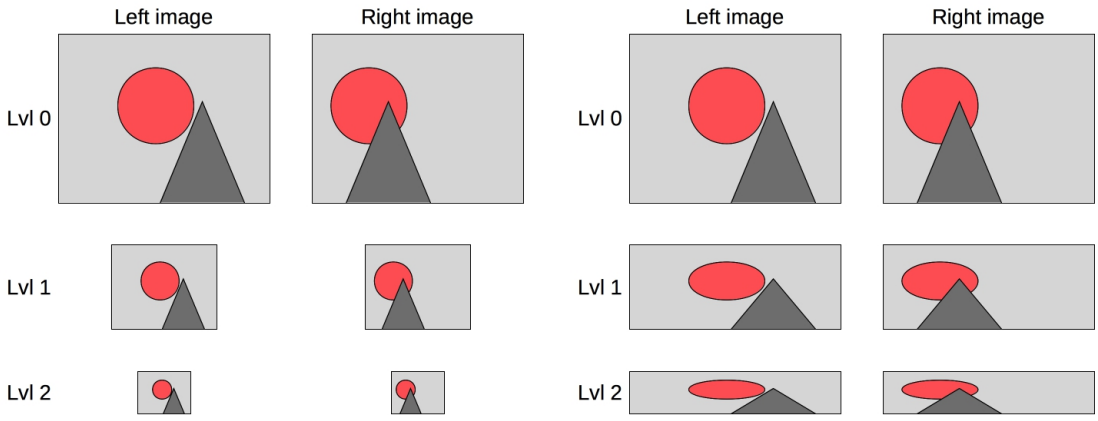
(b) Dividing the images along both y and x axes



(c) Dividing the images along the y axis

Figure 7.13: Illustration of dividing the images into smaller sub images

case of $K = L = 3$ the memory usage could be reduced to 515 MB. But some problems can occur when dividing this way. Looking at the triangle object in figure 7.13b it is seen that the top of the triangle in the left image is in the rightmost column of sub-images while it is in the middle column of sub-images in the right image. This will result in a false matching when processing the sub-image containing the triangle top. To solve this issue the original image can instead be divided along the y-axis as illustrated on figure 7.13c. Since the stereo matching only occurs along the x-axis, this way of dividing the image will not affect the matching the same way. The triangle top is, as seen in the figure, in the same sub-image. But it will require smaller rows to acquire the same reduction in memory as if dividing on both axes, since the reduction factor is proportional to L. Using the example in figure 7.13c, where $L = 3$, the memory will only be reduced to 1.5 GB so L have to be higher. It should be noted that with this method the stereo matching near borders between sub-images will be a bit worse than if working on full images since the aggregation will not include the cost values across the border.



(a) Along both axes **(b)** Along y-axis

Figure 7.14: Image pyramid downsampling

The second method to reduce memory use is using an image pyramid. An image pyramid is a down- and upsampling method which is often used in the computer vision field. It works by convolving the image with a gaussian kernel and then remove the even rows and columns which provide an image with a size reduced by a factor 4. This process can be repeated to downsample further. Each downsampled image is called a level in the pyramid where level 0 is the original size of the image, level 1 is the image downsampled once, etc. The original image size is reduced by a factor $2^{level+1}$ at each level. Figure 7.14a illustrates an image pyramid with 2 levels. A strength of this method is that beside memory usage reduction, fewer pixels are processed and this will result in the algorithm being faster. Using the figure as an example the memory usage at level 2 is reduced to 579.4 MB.

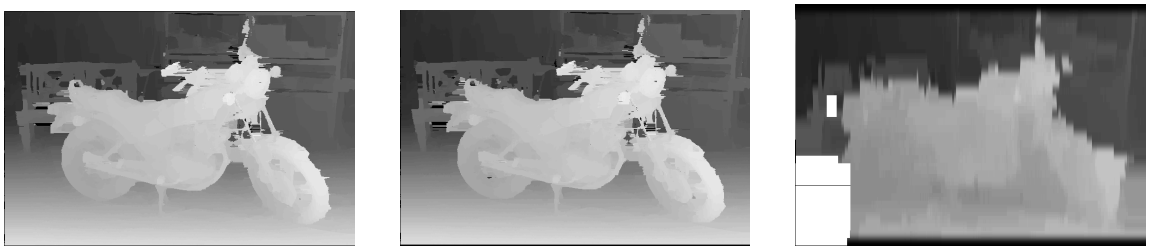
Since pixels are removed then data is lost and the disparity precision will be lower than with the original images. The most important data are those along the x-axis since the stereo matching occurs along this axis. To avoid losing this data the image pyramid could be modified to only downsample along the y-axis. This is illustrated on figure 7.14b. With this modified image pyramid the size of the original image is reduced by a factor 2^{level} at each level. Using the example on figure 7.14b at level 2 the memory usage is reduced to 1.16 GB. This method introduces some pre- and post-processing. The images have to be upsampled again after the algorithm has processed the image. The standard method for upsampling using an image pyramid is to double the size of the current image by inserting rows and columns of 0 next to each pixel or in the case of the modified version only rows will be inserted. Then the image is filtered with a gaussian kernel. An issue with this method is that since the upsampling is performed on the disparity map the calculated disparity values can be changed by the filtering. This is unwanted and instead of filtering the image, it has chosen to interpolate the rows containing 0 using linear interpolation.

Comparison of methods

First, it is calculated how much the images should be divided or downsampled to reduce the memory usage below 512 MB and then the python simulation created in section 4.3 have been modified to use one of the two methods. The changes in stereo matching and runtime are presented in table 7.3 and figure 7.15 shows the resulting disparity maps.
For memory usage of method 1 to go below the available memory on the Zedboard, it has to be divided into at least 10 sub-images and the memory usage will then be 463.5 MB. It should be noted that the image division results in the lower and upper border of each sub-image the disparity is equal to 0. This is resolved by setting the lower border equal to the row just above it and the upper border equal to the row just below it. This equal to a nearest neighbor interpolation.
For method 2 to go below the available memory it have to be downsampled 4 times and then the memory usage is equal to 289.7 MB.

Method	**false matches [%]**	**Δ false matches**	**runtime [s]**	**Δ runtime [s]**
None	8.3	-	625	-
1	8.2	-0.1	593	-32
2	27.9	+19.6	44	-581

Table 7.3: Runtime og matching changes from each memory usage reduction method when using Motorcycle image set.



(a) No method used **(b)** Method 1 used **(c)** Method 2 used

Figure 7.15: Resulting disparity maps using different memery usage reducing methods

From the results, it can be seen that method 2 reduce the runtime by a large margin but the blurring in the image is too extensive and results in a lot of false matches. It also introduces some large blocks of errors seen in the lower left corner of the disparity map. Downsampling by only one level results in a fine disparity map with false matches equal to 9.5% but already at level 2 the disparity map quality has degraded a lot and results in false matches to be 15.2%. Method 1 doesn't change neither the runtime or matching by much and therefore this method is chosen to reduce the memory usage. The decrease in false matches comes mainly from the top and bottom border of the disparity map since

when using no method the algorithm doesn't change these borders and they are equal to zero whereas method 1 copies the line just above the bottom and below the top. If the same functionality is added to the original algorithm the false matches will be 7.9%. The reduction in run-time is believed by us to come from the python script processing smaller matrices.

7.5 Wrap-up

A final FPGA implementation was not achieved due to time constraint for the thesis. But the basic process for design an FPGA hardware implementation is described.
Some challenges emerged during design at higher abstraction levels. One of these challenges were the implementation of exponential which was implemented using a lookup-table since the range of values were low. And another of these challenges is the memory usage due to large images and a high disparity range. The reduce the memory it were found that dividing the images into sub-images were feasible.

Chapter 8

Acceptance test

A final implement were not achieved due to time constraint for the thesis project so a final acceptance test can not be preformed. This chapter will be left blank for this reason.

Chapter 9

Conclusion

This chapter concludes on the project. In chapter 1 (p. 1) it was specified that the goal of this project was to design and implement a stereo vision algorithm and some questions were to be researched:

- What obstacles occur within stereo vision?
- Which stereo vision algorithm exist, both being computationally efficient and at the same time providing good vision results?
- How can an architecture be designed and optimized for executing a stereo vision algorithm?

The findings for each question will be discussed for each question below. In the end of chapter 1 (p. 1) the A^3 model were described and in chapter 6 (p. 45) the Gajski-Kuhn Y-chart were described. These models were used for structuring the report and project. In the end of this chapter the use of these models will be discussed.

1 What obstacles occur within stereo vision?

In chapter 2 (p. 7) the basics of stereo vision were researched along with some challenges stereo vision brings along with it. With occlusions being a significant subject when discussing stereo vision, occlusions and the different types of occlusions were discussed and different methods to find and fill these occlusions were researched. From this research, it was found that some methods for filling occlusions exists. Four methods were looked at in [12]. These methods where neighbor's disparity assignment, diffusion in intensity space, segmentation-based least square and weighted least square. The results from that article were used to conclude that a simple occlusion filling method (neighbor's disparity assignment) would be used for this project since the focus is on execution time and the improvement in matching was not high enough for neglect the runtime of the simpler method.

Another subject researched in chapter 2 (p. 7) was the depth precision. HSA Systems requires a high depth precision for future assignments. HSA Systems specified a sensor they wanted to use for the stereo vision system and it was calculated whether the wanted precision where possible to achieve. It was found that the required depth precision was possible to achieve by sacrifice the vertical scene size and using sub-pixel refinement.
Other subjects researched where rectification of images and the impact from color spaces. The challenges with rectification were described but for this project, it was not researched further since the test images used are already rectified. For the subject of color spaces the foundings in [5] were used. From this, it was concluded to use grayscale images if the impact on stereo matching quality were not too significant since grayscale images can speed up the runtime but will always have worse stereo matching quality.

2 Which stereo vision algorithm exist, both being computationally efficient and at the same time providing good vision results?

There exists a lot of stereo vision algorithms but HSA Systems reduced the search to algorithms with a focus on edge preserving algorithms to ensure a good distinction between objects. Two efficient algorithms which preserve the edge were found and these were: Efficient Edge Preserving Stereo Matching and Fast Cost-Volume. In chapter 2 the algorithms were described and then an algorithm were chosen to be used in this project. For choosing an algorithm both algorithms were simulated in Python and the run-time of each algorithm and the quality of the resulting disparity map were compared. To ensure that the programming skills of the author don't affect the choice of algorithm to use further in the project the theoretical computational complexity of each algorithm were calculated. Both the simulation and the theoretical complexity calculations show that the Efficient Edge Preserving Stereo Matching algorithm is better than the Fast Cost-Volume both when comparing the computational complexity and the quality of the resulting disparity map. Efficient Edge Preserving Stereo Matching was chosen due to the results of the comparison.

3 How can an architecture be designed and optimized for executing a stereo vision algorithm?

With an algorithm chosen, the design process towards an architecture could begin. The platform given by HSA Systems was a Zedboard which contains a Zynq Z7020 SoC. The Zynq Z7020 SoC contains both a general purpose processor and an FPGA. This project only focuses on an FPGA implementation hence the GPP part of the platform were not used since this was stated in the project description from HSA Systems. The GPP part was reserved for HSA Systems in case they wanted to use it for other applications in their products.

A final FPGA implementation was not achieved in this project but the steps towards an implementation such as scheduling and allocation were described in 7. It is concluded the architecture design and optimization process for the EEPSM algorithm can follow the general procedure for FPGA design. In the design process, we went through in this project some challenges emerged: the implementation of exponential function and memory usage.

The EEPSM algorithm uses exponential function and this function isn't trivial to implement. Different estimates and approximations were looked at: using a power series, using CORDIC and using a lookup table. The power series implementation was unfeasible since it required too many terms and too large constants. Among the CORDIC implementation and the Lookup table implementation, it was found that the lookup table would fit this project the best. This is due to CORDIC being an iterative algorithm and hence will result in higher runtime while lookup table only required an acceptable amount of logic elements since the quantity of numbers used in the exponential function is limited.

Another challenge which emerged was a significant memory usage. The algorithm requires saving a lot of copies of the cost images due to the aggregation step. This resulted in the algorithm to require above the available memory on the platform. The algorithm was changed to reduce the memory usage. Two different alterations were considered: Cutting the images into sub-images or down-sampling using image pyramids. The alteration was compared and it were found that the down-sampling altered the resulting disparity map too much to be acceptable while the sub-image alteration didn't affect the algorithm much and hence it were chosen to cut the original images into sub-images.

It can be concluded that to implement a stereo vision algorithm using large image sizes then memory usage can be a challenge and have to be optimized. The large images required for a high depth precision also introduces a high disparity range.

If the project were to be recreated we would ask to change the project description to include hardware/software co-design implementation instead of only a hardware implementation. This would require learning new theory about hardware/software co-design and use other design models such as the Rugby model [13]. The ARM® Processing system contains a NEON engine which is a general-purpose SIMD engine [2]. The NEON engine works with its own pipeline, register and execution hardware hence it can execute operations parallel with the GPP. The instruction set for the NEON engine include instructions such as MAC by vector, vector minimization, vector multiplication etc. These operations can be multiple parts of the algorithm such as aggregation, SAD, and minimization when finding the disparity value. The inclusion of a GPP can also simplify the memory management. Large images and high disparity ranges require a lot of memory management. The GPP could handle the memory management by controlling pointers to pixels which can be sent

to architecture in the FPGA.

Evaluation of Design Models

To structure the report the A^3 model were used. This model works in three domains and helps limit focus in each domain. i.e. in the first part of report only focus on the application, the second part focus on the algorithm and the last part focus on the architecture. The limiting of focus area can help simplify parts of the design process and limit some of the workload i.e. when researching solutions for occlusion filling the details of implementation in architecture is not considered. This helps to limit the details needed to be considered.

To structure the design process of a hardware architecture the Gajski-Kuhn Y-chart were used. This model can help to limit the design process into different domains and abstraction levels. The journey around in the model can be done in different ways i.e. starting at low abstraction levels and moving up in abstraction levels or start at high abstraction levels and moving towards lower levels. It was chosen to follow an FPGA methodology where the design process starts at highest abstraction levels when you have reached a wanted level of abstraction the FPGA software is used to synthesize the design and giving a final implementation.

The models helped to structure the project work and in chapter 7 (p. 49) at some occasion examples of information in later domains in the A^3 model requires the design process to backtrack into earlier domains to solve challenges.

The Gajski-Kuhn Y-chart were feasible for this project since it focuses on a hardware design but if the GPP part of the Zynq SoC were to be used then the Y-chart would be insufficient since the segregation between hardware and software is not naturally modeled in the Y-chart.

Bibliography

[1] Leon Adams and Strategic Marketing. "Choosing the right architecture for real-time signal processing designs". In: *Texas Instruments, Document Number SPRA879* (2002).

[2] ARM. *NEON - ARM*. `https://www.arm.com/products/processors/technologies/neon.php`. 2016.

[3] ASUS. *Xtion PRO | 3D sensor | ASUS Global*. `https://www.asus.com/3D-Sensor/Xtion_PRO/`. 2016.

[4] Avnet. *ZedBoard, (Zynq^{TM} Evaluation and Development), Hardare User's Guide*. `http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf`. Version 2.2. 2014.

[5] Sylvie Chambon and Alain Crouzil. "Colour correlation-based matching". In: *International Journal of Robotics and Automation* 20.2 (2005), pp. 78–85.

[6] Cevahir Çiğla and A Aydin Alatan. "Efficient edge-preserving stereo matching". In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 696–699.

[7] Daniel D. Gajski et al. *Embedded System Design - Modeling, Synthesis and Verification*. Springer, 2009.

[8] Kaiming He, Jian Sun, and Xiaoou Tang. "Guided image filtering". In: *European conference on computer vision*. Springer. 2010, pp. 1–14.

[9] Kaiming He, Jian Sun, and Xiaoou Tang. "Guided image filtering". In: *IEEE transactions on pattern analysis and machine intelligence* 35.6 (2013), pp. 1397–1409.

[10] Asmaa Hosni et al. "Fast cost-volume filtering for visual correspondence and beyond". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.2 (2013), pp. 504–511.

[11] HSA systems. *HSA systems*. `http://hsasystems.com/`. 2016.

[12] Shafik Huq, Andreas Koschan, and Mongi Abidi. "Occlusion filling in stereo: Theory and experiments". In: *Computer Vision and Image Understanding* 117.6 (2013), pp. 688–704.

[13] Axel Jantsch, Shashi Kumar, and Ahmed Hemani. "The Rugby model: A conceptual frame for the study of modelling, analysis and synthesis concepts of electronic systems". In: *Proceedings of the conference on Design, automation and test in Europe*. ACM. 1999, p. 54.

[14] Stefano Mattoccia. *Stereo Vision: Algorithms and applications*. `http://www.vision.deis.unibo.it/smatt/Seminars/StereoVision.pdf`. 2013.

[15] D. Scharstein and R. Szeliski. "A taxonomy and evaluaiton of dense two-frame stereo correspondence alogrithms". In: *International Journal of Computer Vision* 47.1/2/3 (2002), pp. 7–42.

[16] D. Scharstein and R. Szeliski. "High-accuracy stereo depth maps using structured light". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003)* 1 (2003), pp. 195–202.

[17] D. Scharstein et al. "High-resolution stereo datasets with subpixel-accurate ground truth". In: *German Conference on Pattern Recognition (GCPR 2014)* (2014).

[18] Daniel Scharstein, Richard Szeliski, and Heiko Hirschmüller. *vision.middlebury.edu/stereo*. `vision.middlebury.edu/stereo`. 2016.

[19] Sony. *IMX264LLR/LQR, IMX265LLR/LQR*. `http://www.sony-semicon.co.jp/products_en/IS/sensor0/img/product/cmos/IMX264_265_Flyer.pdf`. 2016.

[20] J Sudha et al. "A novel method for computing exponential function using cordic algorithm". In: *Procedia Engineering* 30 (2012), pp. 519–528.

[21] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[22] The Turing Institute. *The History of Stereo Photography*. `http://www.arts.rpi.edu/~ruiz/stereo_history/text/historystereog.html`. 1996.

[23] Charles Wheatstone. "Contributions to the physiology of vision.–Part the first. On some remarkable, and hitherto unobserved, phenomena of binocular vision". In: *Philosophical transactions of the Royal Society of London* 128 (1838), pp. 371–394.

[24] Xilinx Zynq. "Zynq-7000 All Programmable SoC Overview". In: *DS190 (v1. 10) September 27* 2 (2016).

Appendix A

Allocation test

This appendix will explain how the average allocation of logic elements for each functional unit is found.
The result is used in table 5.3 (p. 43) in chapter 7. The table is repeated here in table A.1

	LUT	FF	BRAM	DSP48
Adder	15	15	-	-
Adder/Subtracter	≈ 16	15	-	-
Subtract	15	15	-	-
Multiplier - LUT	352	36	-	-
Multiplier - DSP48	-	-	-	1
Divider	≈ 177	82	0.5	7

Table A.1: Number of logic elements used in average for each FU

A.1 General procedure

This section will describe the general procedure to find the utilization of logic elements for the specified FUs.
First a new project is created in Xilinx Vivado 2016.2. All the settings are set to standard except for the target which is set to ZedBoard Zynq Evaluation and Development Kit. Then a block design is created and a single IP of the wanted type is added. Then for each input and output a port is generated. The inputs are connected to the corresponding ports and between the output of the IP and the corresponding port a Xilinx Slice IP core is inserted which strips every bits except one (MSB) from the output and this IP is connected to the output and the port. This is done to not use to many I/O ports which will make . Using TCL commands the IP block is copied 99 times and for each copy a new slice and output port is generated and connected to the copy. The inputs are all connected

the original corresponding ports except for the divider FU where each copy will have its own input ports. When everything have been connected then Vivado is set to generate top-level HDL wrapper for the block design. Run synthesis and implementation and after completion check the utilization table under the project summary. Note these values, go to the block design and change one of the slices to full output width. Then run synthesis and implementation again and notices if there is a difference in LUT utilization. The LUT utilization is this value minus the LUT value from the former run + 1. These values should be divided by 100 and inserted into table A.1 (p. 73).
The following sections will describe the specific settings for each FUs.

A.2 Adder

This FU uses the Xilinx Adder/Subtracter v12.0 LogiCORE IP. The IP is set to implement using *Fabric*. The inputs are set to a width of 15 bits, mode is set to *add* and the output is set to 15 bits and latency is set to 1. All control signals are disabled.

A.3 Subtract

This FU uses the Xilinx Adder/Subtracter v12.0 LogiCORE IP. The IP is set to implement using *Fabric*. The inputs are set to a width of 15 bits, mode is set to *subtract* and the output is set to 15 bits and latency is set to 1. All control signals are disabled.

A.4 Adder/Subtract

This FU uses the Xilinx Adder/Subtracter v12.0 LogiCORE IP. The IP is set to implement using *Fabric*. The inputs are set to a width of 15 bits, mode is set to *add subtract* and the output is set to 15 bits and latency is set to 1. All control signals beside the ADD signal are disabled.

A.5 Multiplier - LUT

This FU uses the Xilinx Multiplier v12.0 LogiCORE IP. The IP is set to construct the multiplier using *LUTs* and is set to optimize for speed. The inputs are set to a width of 18 bits and the output is set to 36 bits and pipeline stages is set to 1. All control signals are disabled.

A.6 Multiplier - DSP48

This FU uses the Xilinx Multiplier v12.0 LogiCORE IP. The IP is set to construct the multiplier using *Mults* and is set to optimize for speed. The inputs are set to a width of 18

bits and the output is set to 36 bits and pipeline stages is set to 1. All control signals are disabled.

A.7 Divider

This FU uses the Xilinx Divider Generator v5.1 LogiCORE IP. The algorithm type is set to *High Radix*. The inputs are set to a width of 8 bits and the output fractional width is set to 8 bits and latency is set to 3. All control signals are disabled. With this only 5 instances of the Divider is used since each instance uses a lot of IO and to ensure the ability to run implementation without error.

Appendix B

Middlebury data set

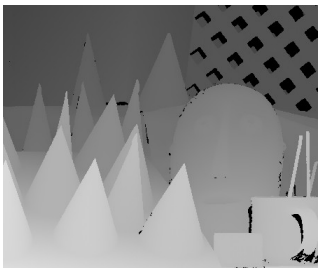
This appendix contains a brief description over the data sets from Middlebury. The computer vision department at Middlebury College have a large library of stereo pair with ground truth[18]. These data sets are used all around the world for evaluating stereo vision algorithms. This thesis uses a subset of these stereo pairs.
These are: Tsukuba, Cones, Teddy and Motorcycle. The three first data sets are used due to great knowledge of them from HSA systems. The last set, Motorcycle, is used due to it having the ground truth as an .pfm file. With a .pfm file then comparing is easier since the ground truth for other sets have scaling hence direct comparison is not possible. The four data sets are presented below.

(a) Left image **(b)** Right image **(c)** Ground Truth

Figure B.1: Tsukuba - 384 $\times$ 288 [15]

(a) Left image

(b) Right image

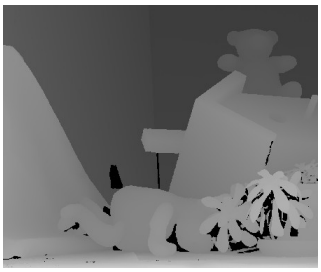


(c) Ground Truth

Figure B.2: Cones - 450 × 375 [16]

(a) Left image

(b) Right image

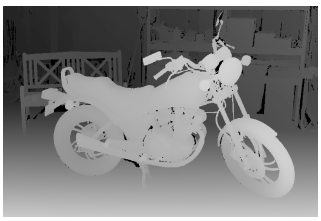


(c) Ground Truth

Figure B.3: Teddy - 450 × 375 [16]

(a) Left image

(b) Right image



(c) Ground Truth

Figure B.4: Motorcycle - 741 × 497 [17]