Master Thesis

- Implementation of stereo vision engine -

Project Report

Group 1072

Aalborg University
Electronics and IT

Here you can write something about which tools and software you have used for typesetting the document, running simulations and creating figures. If you do not know what to write, either leave this page blank or have a look at the colophon in some of your books.??

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: 66

Date of Completion:
October 31, 2016

Abstract:

In this report a fast high precision stereo vision engine for implementation on a FPGA is studied. Different aspects of stereo vision 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 results in the choice of *Fast Cost-Volume* for further implementation. A description of Gajski-Kuhn Y-chart follows with a discussion of pros and cons. Then implementation techniques are described and parts of the algorithm is implemented. In the conclusion it is discussed whether a fast high precision stereo vision engine is possible using the given specifications and the chosen algorithm.

Contents

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 58. 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 58. 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 to in the same way as figures and tables.

Aalborg University, October 31, 2016

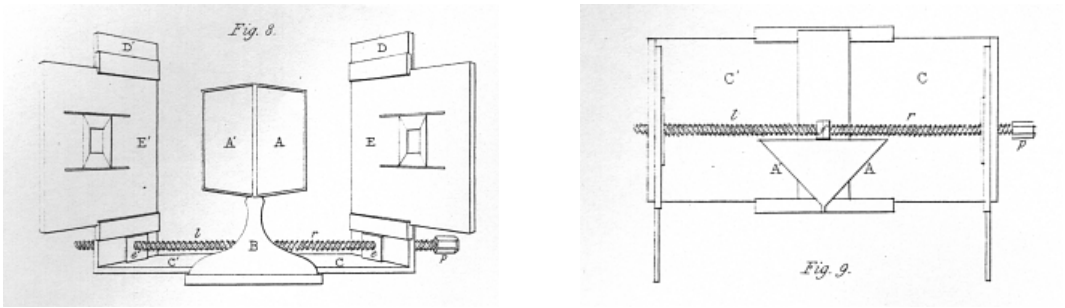
Tomas Brandt Trillingsgaard
<ttrill10@student.aau.dk>

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 [18]

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 paper on this is found in [18]. An illustration of the stereoscope is seen on figure 1.1. The stereoscope uses images taken by two cameras placed next to each other and the stereoscope then isolates the images so each eye only sees one image. This mimics

depth perception and the viewer would perceive depth in the images. [17]

Around 1970, computer vision began appearing and a significant part of this research area is stereo vision: the measurement of depth mimicking the human vision using two cameras [16]. The ability to measure depth enables a computer to distinguish between objects and hence better interact with and react to the world.

1.2 HSA Systems

(a) PV650C[8] **(b)** Pharma lay-flat media [8]

Figure 1.2: Some products from HSA Systems

HSA Systems is a danish company with a R&D department in Aalborg, which develops and manufactures high-resolution inkjet printer systems and a range of other products. Figure 1.2 shows some of these printer systems. These systems prints 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 enable them to know 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.

In future uses, HSA Systems would like the stereo vision system to have very precise depth measurements of $\leq$5 mm at distances between 0.5-1.5 m.

1.3 Motivation

The area of stereo vision has been researched thoroughly and precise algorithms have been found but most algorithms are very complex computational wise. HSA Systems wishes

for a stereo vision system which can run real-time (10 fps) and have a high precision of $\leq$5 mm between 0.5 m and 1.5 m .

1.4 Problem description

HSA Systems wishes to track packages going through their system. A strategically placed stereo vision camera will enable them to know how many and where these objects are in the system.
The following questions will be researched:

- What obstacles occur within stereo vision
- Which stereo 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 for an FPGA. This project will not focus on developing a new stereo vision algorithm. Limitations and issues with stereo vision algorithms will be analyzed but 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 (p. 4). 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.

The search for a solution starts in the application domain where the problem and application are being explored. Chapter 2 (p. 5): 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. 5).

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

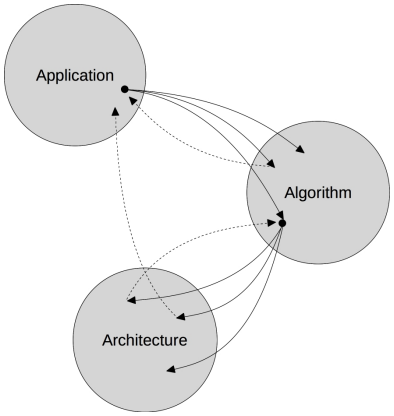


Figure 1.3: A^3 model

Chapter 6 (p. 39) 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.

Chapter 8 (p. 53) specifies how the testing is done to determine whether the found solution complies with the requirements in chapter 3 (p. 19). Chapter 9 (p. 55) 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. 3) 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).

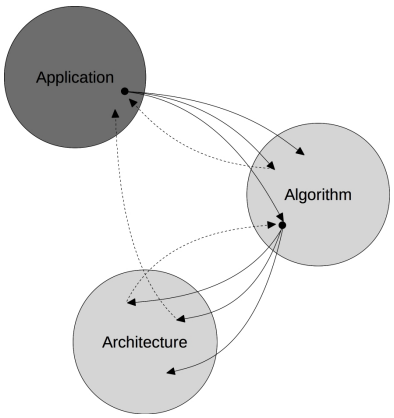
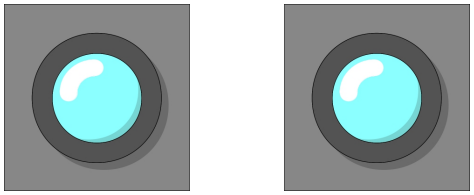


Figure 2.1: A^3 model with the application domain marked

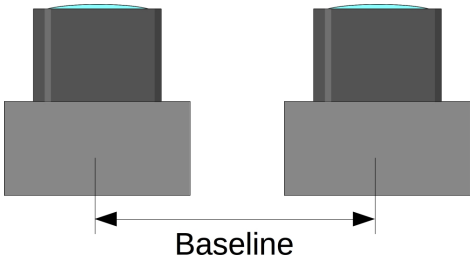
2.1 The Basic Principle of Stereo Vision

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. 6) shows an example of this setup.

Figure 2.3 (p. 7) 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.



(a) Seen from the front



(b) Seen from above

Figure 2.2: Illustration of a standard stereo vision setup

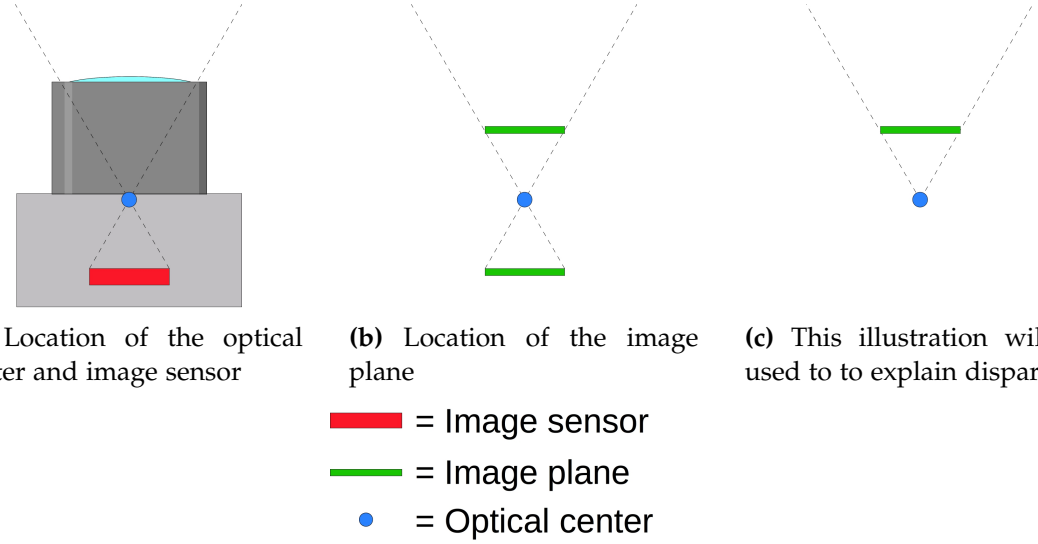
To simplify comparisons to the real world, an image plane can be placed opposite of the optical center at the same distance from the center and this image plane will not be inverted.
Figure 2.4a (p. 7) shows that a single camera is not able to differentiate between two points at the same angle from the optical center at different distances. Figure 2.4b (p. 7) illstrates how adding the second camera allows differentiating between the two points.
Figure 2.5 (p. 8) 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. 8) and 2.5b (p. 8) shows how the disparity changes depending on the distance to the point. Figure 2.5c (p. 8) 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. 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:

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

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.

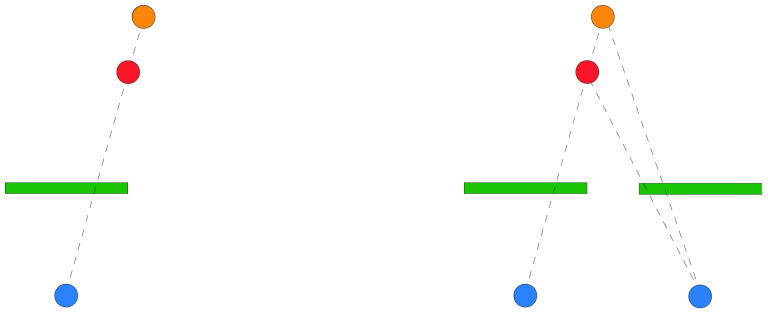


(a) Location of the optical center and image sensor

(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



(a) Seen from a single camera

(b) Seen from two cameras

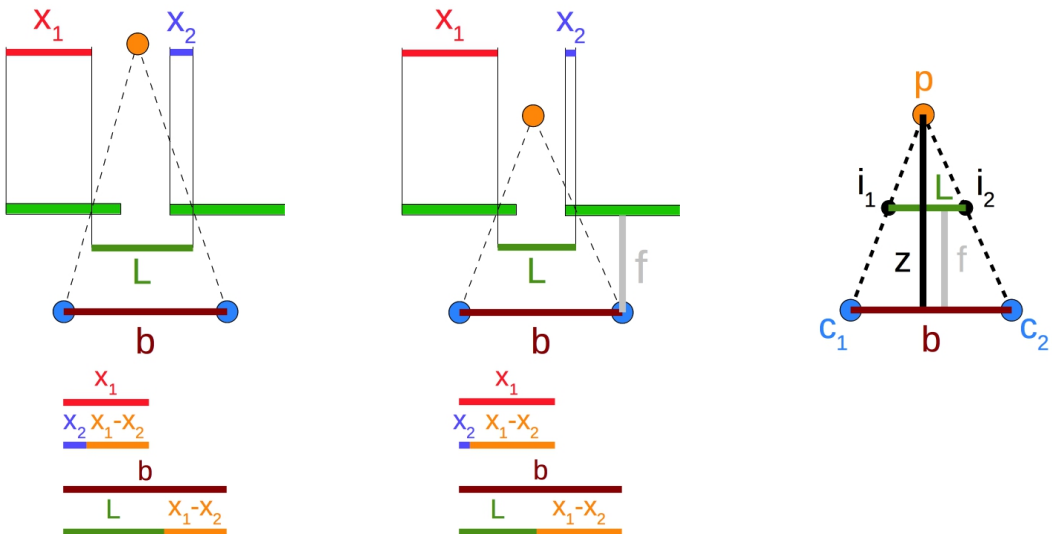
Figure 2.4: Example of two points in a scene at different depths

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 align perfectly.

Figure 2.6 (p. 8) 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.



(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

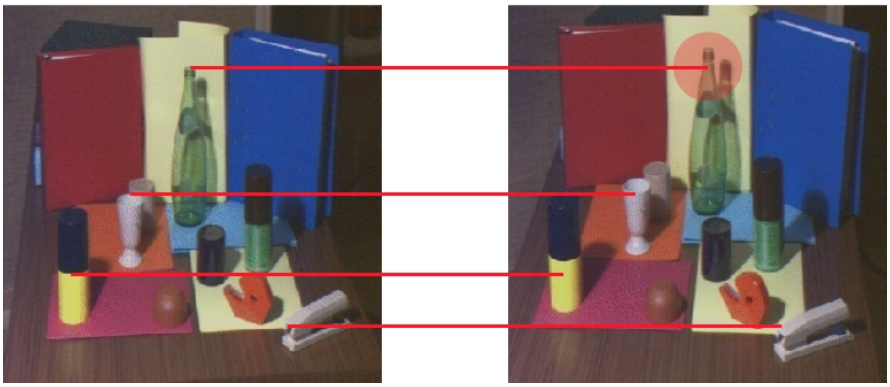


Figure 2.6: Non-rectified stereo pair [10]

Epipolar geometry occurs when a scene is seen from two different views. Figure 2.7 (p. 9) illustrates epipolar geometry. An epipole is the projection, on one cameras image plane, of the optical center of the other camera and is illustrated on figure 2.7 (p. 9) as e_1 and e_2. The red line going through p_1 (the projection of point P on the image plane) and e_1 is called an epipolar line and a corresponding epipolar line can be found on the other image plane going through p_2 and e_2. 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. To

simplify the search further, the image can be rectified.

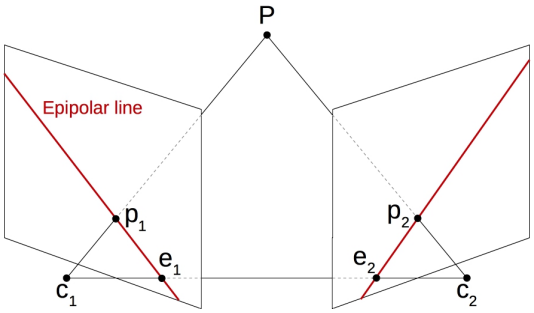


Figure 2.7: Illustration of epipolar geometry

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. 8) but rectified. As seen, the corresponding points can now be found by following the horizontal lines or the x-axis.

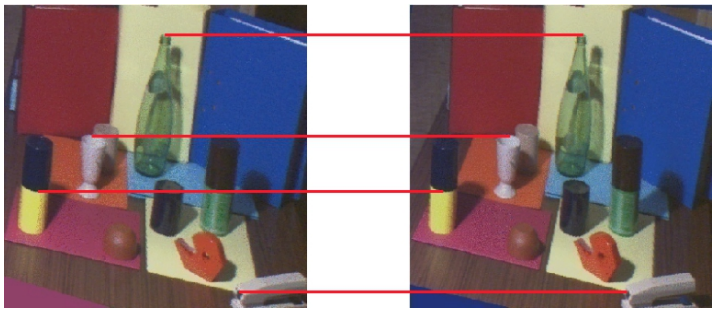


Figure 2.8: Rectified stereo pair [10]

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 and all new manual calibration. HSA Systems theorizes that a system can be developed which instead of rectifying the image will find the epipolar lines and feed this information to the stereo cameras.

In this project stereo image pairs from Middlebury Vision Test sets [14] will be used. These image pairs are rectified, and rectification of image will not be discussed further in this project.

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 [2]

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.

For this project the impact of color spaces have not been studied but instead the findings in [2] are used. This study descirbes 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 [2]. 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 to 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 used for a final product will be 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 [15]. HSA Systems requires that the baseline is ≤10 cm.
To calculate the precision equation 2.2 (p. 6) 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.

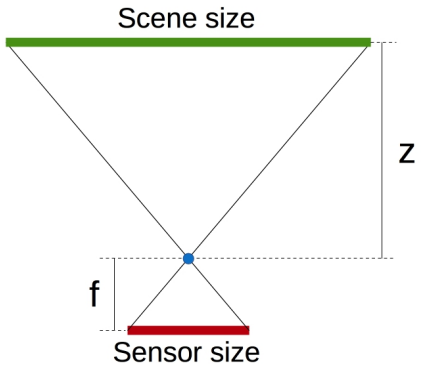


Figure 2.9: Illustration of the relationship between sensor size, scene size, focal length and distance

Since the two triangles in the figure are similar then 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 $\leq$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 doesn't not comply to 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 don't comply with the requirement.
To improve the precision further either the resolution, the baseline or the focal can be changed but each require an other requirement to fail. The resolution requires a new camera sensor. The baseline need to be at least 11 mm larger. The focal length need to at least 0.73 mm higher but this results in a scene size of 1631$\times$1361 mm.

It is chosen to reduce the scene size to 1631$\times$1361 mm and use a focal length of 7.82 mm. This result in a precision of 10 mm at $\approx$1500 mm so sub-pixel refinement have to be used. And the disparity range will be 275. Figure 2.10 shows the disparity precision at different distances with the chosen baseline, focal length and sensor.

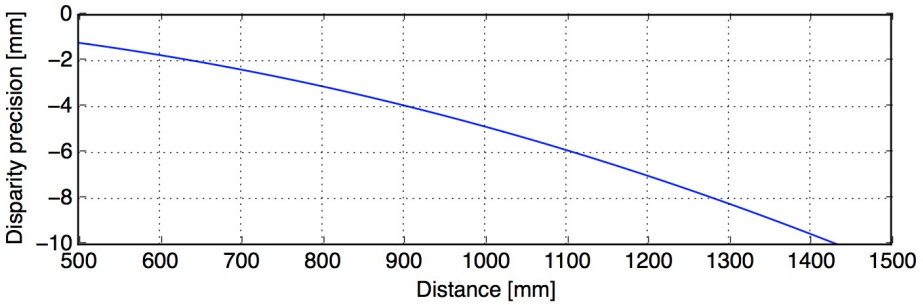
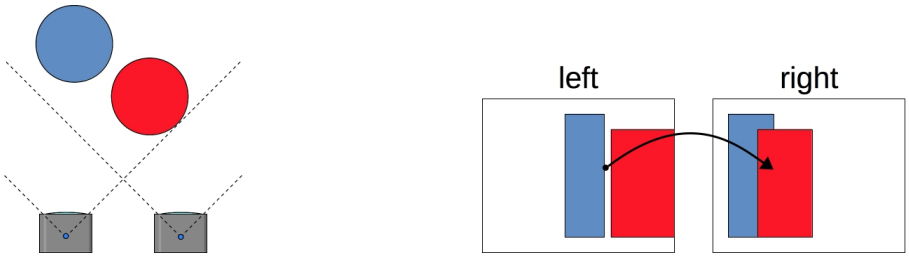


Figure 2.10: Disparity precision in the distance range 0.5$-$1.5 m

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. 13) illustrates two cylinders seen by a stereo camera setup where occlusion occurs. Figure 2.11a (p. 13) shows that the blue cylinder is only partially visible to the right camera because the red cylinder blocks the view 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. 13), the edge of

the blue cylinder can not be found and it will result in calculating a wrong disparity value.



(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

There exist different types of occlusions: partial occlusions, self-occlusions, border occlusions 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 blocked 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 [9] is used which will be described in section 2.5.1 (p. 14) 2.5.4 (p. 16)??. 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 use 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 energy will be 0. Then the diffusion energy for non-border occlusion is found by:

$$E(p_L) = \min_{l_{p_L}=\{0,\dots,l_{max}\}} \left(\frac{1}{2|q_L \in \mathcal{N}(p_L) \wedge l_{q_L=l_{p_L}}|} \sum_{q_L \in \mathcal{N}(p_L) \wedge l_{q_L=l_{p_L}}} (|\bar{I}(p_L) - \bar{I}(q_L)| + E(q_L)) \right) \tag{2.8}$$

And the diffusion energy for border occlusions are found by:

$$E(p_L) = \min_{l_{p_L}=\{0,\dots,l_{p_{Lf}}-2\}} \left(\frac{1}{2|q_L \in \mathcal{N}(p_L) \wedge l_{q_L=l_{p_L}}|} \sum_{q_L \in \mathcal{N}(p_L) \wedge l_{q_L=l_{p_L}}} (|\bar{I}(p_L) - \bar{I}(q_L)| + E(q_L)) \right) \tag{2.9}$$

The diffusion energy will be calculated for each occluded point and for each point the disparity which corresponds the minimum $E(p_L)$ is set as the disparity l_{p_L} for the occluded point.

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.

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. It is assumed that the color intensity between objects is significantly different and this property can be used to distinguish between foreground points and background points.

Each error term in the aggregated residual should be weighted so the foreground does not have much influence. The aggregated residual is then defined as:

$$\Delta = \sum_{q_L \in \mathcal{N}(p_L)} w_{q_L} (\hat{l}_{p_L}(p_L) - l_{p_L}(q_L))^2 \tag{2.10}$$

where $w_{q_L} = e^{-\mu_L |\bar{I}(p_L) - I(q_L)|}$ (the weight) is the likelihood of p_L with q_L under the assumption of an exponential distribution model of $|\bar{I}_(p_L) - I(q_L)|$. $\bar{I}(p_L)$ is the mean intensity of p_L and μ_L and is also called the decay rate. $\hat{l}_{p_L}(p_L)$ is the estimated disparity of p_L (will be estimated during interpolation) and $l_{p_L}(q_L)$ is the disparity of q_L.

$\bar{I}(p_L)$ is the mean intensity of p_L which can be obtained using mean shift algorithm in a window around p_L. To estimate this value then initialize the algorithm with $\bar{I}(p_L)$ equal to the intensity of p_L then the mean shift algorithm repeatedly picks those neighbors inside the window that satisfy $|\bar{I}(p_L) - I(q_L)| \geq 3\mu^{-1}$ and then assign the average of intensities of the selected neighbors to $\bar{I}(p_L)$ until $\bar{I}(p_L)$ converges to a fixed average. $|\bar{I}(p_L) - I(q_L)|$ has decay rate μ_L which is related to the decay rate μ of the variable $|I(p_L) - I(q_L)|$ by $\mu_L^2 = \mu$.

For this model a matrix containing the coordinates for all the control points is generated:

$$F = \begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \ddots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \tag{2.11}$$

and a vector with the corresponding labels for the coordinates in F is generated:

$$L = [l_1 \ \cdots \ l_N] \tag{2.12}$$

Then a Linear model can be expressed as:

$$l_{p_L} = a + bx(p_L) + cy(p_L) \tag{2.13}$$

Where $(x(p_L), y(p_L))$ is the coordinates of p_L and a, b and c are the model parameters.

The weights for the control points can be express in a vector as:

$$w = [w_{q_{L1}}\ w_{q_{L2}}\ \cdots\ w_{q_{LN}}]' \tag{2.14}$$

Then compute two new matrices, F_w and L_w:

$$F_w = diag(w)F \tag{2.15}$$
$$L_w = diga(w)L \tag{2.16}$$

The model parameter vector:

$$P = [\,a\ b\ c\,]' \tag{2.17}$$

By combining the equations above then the following equation is given:

$$P = (F_w^T F_w)^{-1} F_w^T L_w \tag{2.18}$$

With these equation the disparity of the occluded point can be estimated:

$$\hat{l}_{p_L} = [1\ x(p_L)\ y(p_L)]P \tag{2.19}$$

2.5.4 Segmentation-based Least Squares (SLS)

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. This section includes a brief description of SLS, while more details are available in [9].

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. In this case, equation 2.20 can be used to find the homogeneity. The lower result given by the equation the higher the homogeneity is.

$$\sum_{q_L \in N(p_L)} = \psi(p_L, q_L) \tag{2.20}$$

Where q_L is a non-occluded point, $N(p_L)$ is a set of neighboring non-occluded points and $\psi(p_L, q_L)$ is a cut-off function expressed as:

$$\psi(p_L, q_L) = \begin{cases} |\bar{I}(p_L) - I(q_L)| & \text{, if} |\bar{I}(p_L) - I(q_L)| \leq 3\mu_L^{-1} \\ 3\mu_L^{-1} & \text{, otherwise} \end{cases} \tag{2.21}$$

When the occluded point with highest priority is found then some initial control points from $N(p_L)$ are needed. First only points from the background is wanted therefore only points in $N(p_L)$ which satisfies $l_{q_L} < l_{p_{Lf}}$ where l_{q_L} is the disparity of a point, q_L in $N(p_L)$ and $l_{p_{Lf}}$ is the disparity of the nearest point from the foreground which is expected to be the nearest non-occluded point to the right of p_L. But a narrow object in the scene might result in the nearest non-occluded point to the right to be a part of the background therefore a second constraint is added. The nearest non-occluded point to the left is assumed to be a part of the background and therefore all points in $N(p_L)$ which satisfies $|l_{p_{Lb}} - l_{q_L}| \leq 1$ should also be part of the background. $l_{p_{Lb}}$ is the disparity of the nearest non-occluded point to the left. With these constraints a combined constraint can be expressed:

$$|l_{p_{Lb}} - l_{q_L}| \leq 1 \vee l_{q_L} < l_{p_{Lf}} \tag{2.22}$$

All points in $N(p_L)$, which satisfies this combined constraint, 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 [14], that $N(p_L)$ only contains points from either one background surface or two background surfaces.

First it is needed to calculate whether the points in the updated $N(p_L)$ belongs to one or two surfaces. For this the expression $l_{max} - l_{min} \leq 1$ can be used. l_{max} is the maximum disparity in $N(p_L)$ and l_{min} is the minimum disparity and if these disparities is close it is assumed that they belong to the same surface and all points in $N(p_L)$ can be used as controls points for finding the disparity for p_L. If $l_{max} - l_{min} > 1$ then $N(p_L)$ should be divided into two sets, $N_1(p_L)$ and $N_2(p_L)$.
If a point q_L in $N(p_L)$ satisfies $|l_{max} - l_{q_L}| \leq 1$ then it belongs to $N_1(p_L)$ or if it satisfies $|l_{min} - l_{q_L}| \leq 1$ then it belongs to $N_2(p_L)$. With $N(p_L)$ divided into two groups it is needed to determine which set p_L belongs to. For this the average truncated color distance is used and it is expressed as:

$$D(p_L, N_i(p_L)) = \frac{1}{|N_i(p_L)|} \sum_{q_L \in N(p_L)} \psi(p_L, q_L) \tag{2.23}$$

If $D(p_L, N_1(p_L)) < D(p_L, N_2(p_L))$ then p_L belongs to $N_1(p_L)$ else it belongs to $N_2(p_L)$. Then the set, which p_L, belongs to can used as control points for finding the disparity of p_L.

2.6 Occlusions filling result

In [9] 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 and data sets from [14], which are already rectified, will be used in this project.

Color Space

The result from [2] shows that color spaces in most cases is 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:

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

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 isn't important either since the stereo setup is intended being 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 choosing e.g when 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 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 1.3 (p. 3)
3	Camera resolution	2464×2056	pixels		Section 2.4 (p. 11)
4	Pixel size	3.45	µm		Section 2.4 (p. 11)
5	Focal length	7.09	mm		Section 2.4 (p. 11)
6	Scene size	1.5×1.5	m		Section 2.4 (p. 11)
The algorithm should be implemented on a *Xilinx Zynq Z7020*					

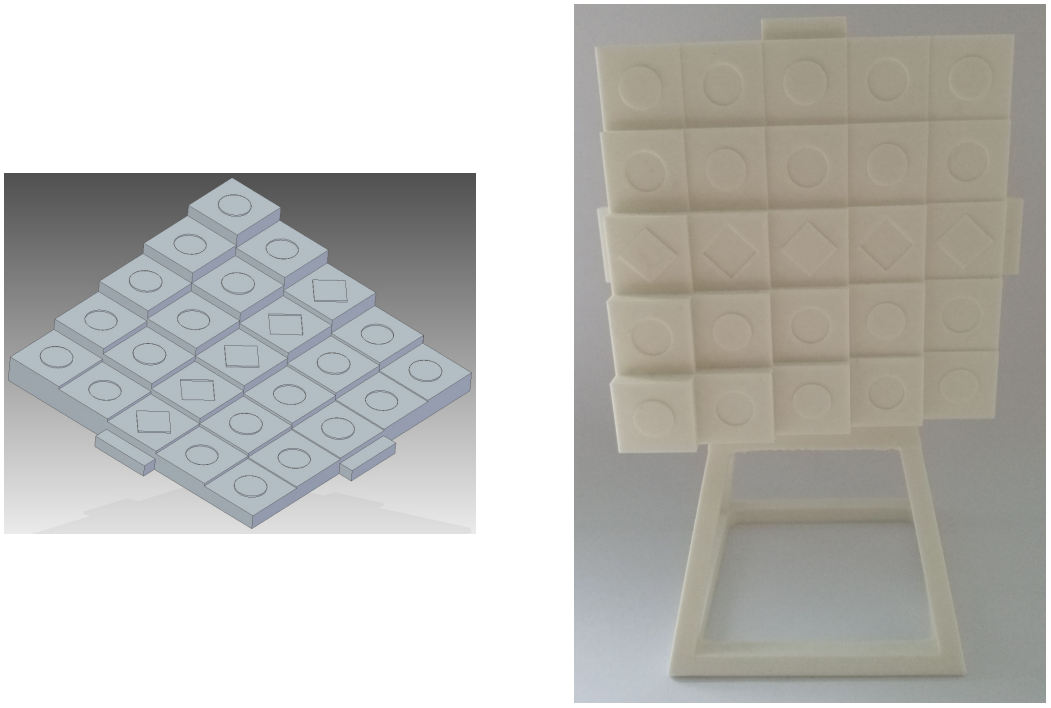
3.2 Test specification

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 run time which then 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 in 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 steps 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 precisison 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 distinguish 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. 10) 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. 53) will perform the available tests.

Chapter 4

Algorithm design

This chapter will explore the algorithm domain in the A^3 model described in section 1.6 (p. 3) and 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 created and the results of these simulations are compared and from this, an algorithm is chosen.

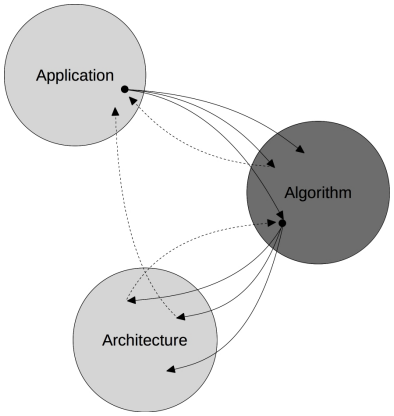


Figure 4.1: A^3 model with the algorithm domain marked

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 [3] and [7]. The pseudo code for the EEPSM is created by the author while the FCV pseudo code

comes from [6] with some details added by the author of this thesis.

4.1 Efficient Edge Preserving Stereo Matching (EEPSM):

This algorithm is described in [3]. 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:

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

$$\tag{4.2}$$

where i is the color (either red, green, or blue), $I_{left}(x,y,i)$ is a pixel in the left image or the reference image, and $I_{right}(x+d,y,i)$ is a pixel shifted by the disparity, d, in the right image or the target image.
It is noticed that 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.

$$C_d^{CENSUS}(x,y) = Ham(CT_{left}(x,y), CT_{right}(x+d,y)) \tag{4.3}$$

$$\tag{4.4}$$

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 for 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 transformer. 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.

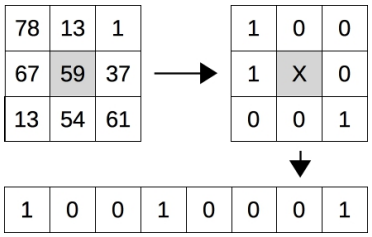


Figure 4.2: Illustration of census transform

When the two cost values are calculated then they can be combined to a single cost value.

$$C_d(x,y) = \alpha \cdot C_d^{SAD}(x,y) + (1-\alpha) \cdot C_d^{CENSUS}(x,y) \tag{4.5}$$

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:

$$\mu(x,y) = \min(e^{\frac{-\Delta R}{\sigma}}, e^{\frac{-\Delta G}{\sigma}}, e^{\frac{-\Delta B}{\sigma}}) \tag{4.6}$$

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 4.7

$$\mu_D(x,y) = \min(e^{\frac{-(R(x,y)-R(x,y-1))}{\sigma}}, e^{\frac{-(G(x,y)-G(x,y-1))}{\sigma}}, e^{\frac{-(B(x,y)-B(x,y-1))}{\sigma}}) \tag{4.7}$$

The next step is to aggregate the cost in equation 4.5 (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:

$$C_d^R(x) = C_d(x) + \mu_R(x-1)C_d^R(x-1) \tag{4.8}$$

$$C_d^R(x) = C_d(x) + \sum_{i=1}^{x-1}\left(C_d(x-i)\prod_{j=i}^{i}\mu_R(x-j)\right) \tag{4.9}$$

From equation 4.9 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 the only pixel 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 using equation 4.9 can be 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:

$$C_d^H(x) = C_d(x) + \mu_R(x-1)C_d^R(x-1) + \mu_L(x+1)C_d^L(x+1) \tag{4.10}$$

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, $C_d^H(x)$, and moves in the vertical directions, up and down.

The cost from vertical aggregation, $C_d^V(x)$, will be influence by the cost from pixels belonging to the same object as pixel x 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.

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 [7]. 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 an combination of the sum of absolute differences and another cost. Instead of the cost based on census transform and hamming distance this algorithm uses difference in the gradient in the horizontal direction. The following equations shows the different cost function for this algorithm and equation 4.13 (p. 27) shows the combined initial cost. First the SAD cost is calculated and this equation is similar to equation 4.1 (p. 24).

$$C_d^{SAD}(x,y) = \sum_{i=1}^{3} |I_{left}(x,y,i) - I_{right}(x+d,y,i)| \tag{4.11}$$

Then the gradient cost is calculated.

$$C_d^{Grad}(x,y) = \nabla_x I_{left}^g(x,y) - \nabla_x I_{right}^g(x+d,y) \tag{4.12}$$

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

$$C_d(x,y) = \alpha \cdot C_d^{SAD}(x,y) + (1-\alpha) \cdot C_d^{Grad}(x,y) \tag{4.13}$$

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.

$$C'_d(x_i,y_i) = \sum_{x_j,y_j} W_{(x_i,y_i),(x_j,y_j)}(I) C_d(x_j,y_j) \tag{4.14}$$

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 described how the filter weights are found.
This aggregate cost can then be used for minimization along the disparity estimates with a winner take 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 [5] and [6] and this section is mostly repeats what these articles describes.
To describe the guide image filter a standard linear translation-variant filtering process is defined:

$$q_i = \sum_j W_{i,j}(I) p_j \tag{4.15}$$

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:

$$q_i = a_k I_i + b_k, \forall i \in \omega_k \tag{4.16}$$

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 [6] and the solution is shown in the following equations:

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon} \tag{4.17}$$

$$b_k = \bar{p}_k - a_k \mu_k \tag{4.18}$$

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:

$$q_i = \frac{1}{|\omega|} \sum_{k|i \in \omega_k} (a_k I_i + b_k) \tag{4.19}$$

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

$$q_i = \bar{a}_i I_i + \bar{b}_i \tag{4.20}$$

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:

$$a_k = \frac{\sigma_k^2}{\sigma_k^2 + \epsilon} \tag{4.21}$$

$$b_k = (1 - a_k)\mu_k \tag{4.22}$$

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 chapter and therefore the FCV psuedo code is for a grayscale guidance image. For a color guidance image small changes have to be added. Equation **??** (p. ??) will then become:

$$q_i = \mathbf{a}_k^T \mathbf{I}_i + b_k, \forall i \in \omega_k \tag{4.23}$$

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:

$$\mathbf{a}_k = (\Sigma_k + \epsilon U)^{-1} \left(\frac{1}{|\omega|} \sum_{i \in \omega_k} \mathbf{I}_i p_i - \mu_k \bar{p}_k \right) \tag{4.24}$$

$$b_k = \bar{p}_k - \mathbf{a}_k^T \mu_k \tag{4.25}$$

$$q_i = \bar{\mathbf{a}}_k^T \mathbf{I}_i + \bar{b}_i \tag{4.26}$$

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. $\mu_{I,r} = f_{mean}(I_{l,r})$
   $\mu_{I,g} = f_{mean}(I_{l,g})$
   $\mu_{I,b} = f_{mean}(I_{l,b})$
   $\mu_p = f_{mean}(C_d)$
   $\mu_{Ip,r} = f_{mean}(I_{l,r} \cdot p)$
   $\mu_{Ip,g} = f_{mean}(I_{l,g} \cdot p)$
   $\mu_{Ip,b} = f_{mean}(I_{l,b} \cdot p)$
4. $\sigma_{I,r,r} = f_{mean}(I_{l,r} \cdot I_{l,r})/N - \mu_{I,r} \cdot \mu_{I,r}$
   $\sigma_{I,r,g} = f_{mean}(I_{l,r} \cdot I_{l,g})/N - \mu_{I,r} \cdot \mu_{I,g}$
   $\sigma_{I,r,b} = f_{mean}(I_{l,r} \cdot I_{l,b})/N - \mu_{I,r} \cdot \mu_{I,b}$
   $\sigma_{I,g,g} = f_{mean}(I_{l,g} \cdot I_{l,g})/N - \mu_{I,g} \cdot \mu_{I,g}$
   $\sigma_{I,g,b} = f_{mean}(I_{l,g} \cdot I_{l,b})/N - \mu_{I,g} \cdot \mu_{I,b}$
   $\sigma_{I,b,b} = f_{mean}(I_{l,b} \cdot I_{l,b})/N - \mu_{I,b} \cdot \mu_{I,b}$
   $cov_{Ip,r} = \mu_{Ip,r} - \mu_[I, r] \cdot \mu_p$
   $cov_{Ip,g} = \mu_{Ip,g} - \mu_[I, g] \cdot \mu_p$
   $cov_{Ip,b} = \mu_{Ip,b} - \mu_[I, b] \cdot \mu_p$
5. $\Sigma = f_\Sigma(\sigma_{I,r,r}, \sigma_{I,r,g}, \sigma_{I,r,b}, \sigma_{I,g,g}, \sigma_{I,g,b}, \sigma_{I,b,b})$
   $cov_{Ip} = [cov_{Ip,r}, cov_{Ip,g}, cov_{Ip,b}]$
   $a = cov_{Ip} \cdot f_{inv}(\sigma_I + epsilon \cdot U)$
   $b = \mu_p - a_r \cdot \mu_{I,r} - a_g \cdot \mu_{I,g} - a_b \cdot \mu_{I,b}$
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. $q = \mu_{a,r} \cdot I_{i,r} + \mu_{a,g} \cdot I_{i,g} + \mu_{a,b} \cdot I_{i,b} + \mu_b$

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 [7]. The code for EEPSM is written by the author of this thesis using the description in this

chapter. None of the Python implementations have been optimized further.

The simulation were running 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™ 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 [14] is used. The stereo pairs are: Tsukuba, Cones, Teddy and Motorcycle and they are shown on figure 4.3. More information about the data sets is seen in appendix C (p. 65).

(a) Tsukuba [11]

(b) Cones [12]

(c) Teddy [12]

(d) Motorcycle [13]

Figure 4.3: Middlebury data set - left images [14]

Figures **??** (p. ??) to **??** (p. ??) shows resulting disparity maps.

Image	Resolution	EESPM	FCV
Tsukuba	384×288	337 s	180 s
Teddy	450×375	1147 s	534 s
Cones	450×375	1105 s	545 s
Motorcycle	741×497	2471 s	1378 s

Table 4.1: Run time for different stereo pairs

Image	Resolution	EESPM	FCV
Motorcycle	741×497	46 %	14 %

Table 4.2: Procentage false disparity estimates

4.4 Theoretical Complexity

The simulations of the algorithms have to many 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 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 have be approximated with a power series:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \tag{4.27}$$

Then it should be determined how many terms should be used for achieving sufficient precision. The exponential function is used in the calculation of permeability weights and the values all is in the interval $[-\frac{0}{25}; -\frac{255}{25}]$

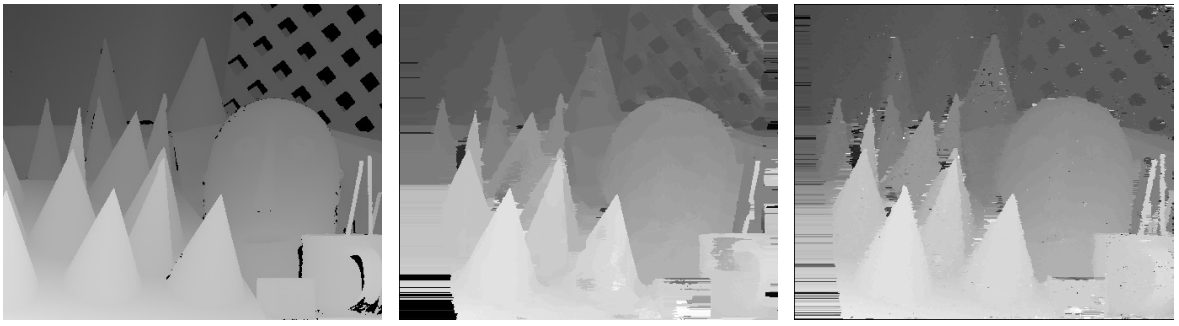
	type	add/subtract	multiplication	division	comparisons
EEPSM	C	$10320 \cdot N$	$9340 \cdot N$	0	$6624 \cdot N$
	G	$8940 \cdot N$	$4305 \cdot N$	0	$6624 \cdot N$
FCV	C	$268675 \cdot N$	$57750 \cdot N$	$275 \cdot N$	0
	G	$95150 \cdot N$	$14575 \cdot N$	$275 \cdot N$	0

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

I/O communication can also have a significant influence on the performance hence the memory usage for each algorithm have also been calculated and the result is shown in table 4.4 (p. 33).

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

Figure 4.4: Tsukuba [11]



(a) Ground truth [12] **(b)** FCV **(c)** EEPSM

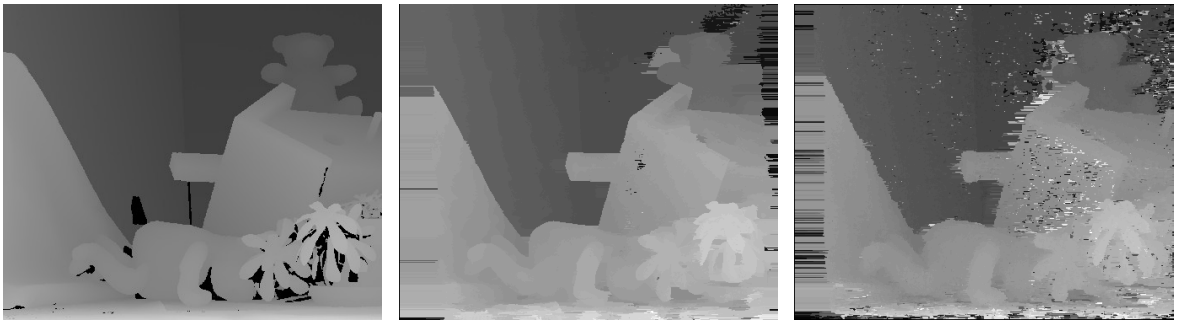
Figure 4.5: conkuba

	type	reads	writes	Memory [B]
EEPSM	C	$7751 \cdot N$	$1375 \cdot N$	$1381 \cdot N$
	G	$7189 \cdot N$	$1375 \cdot N$	$1377 \cdot N$
FCV	C	$22280 \cdot N$	$275 \cdot N$	$281 \cdot N$
	G	$11278 \cdot N$	$275 \cdot N$	$277 \cdot N$

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

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. 30). Looking at table 4.1 it is seen that the Python implementation of the FCV algorithm has a far lower runtime than the EEPSM algorithm. The results might be misleading since the code for the FCV algorithm was inspired by an matlab example while the code for the EEPSM algorithm was written from the description.



(a) Ground truth [12] **(b)** FCV **(c)** EEPSM

Figure 4.6: Teddy

(a) Ground truth [13] (Is dark due to scaling) **(b)** FCV **(c)** EEPSM

Figure 4.7: Motorcycle

Looking at table 4.2 (p. 32) it is seen that the FCV algorithm is much better at matching than the EEPSM algorithm.

Referring to cost function described in chapter 3 (p. 19) it is noticed that time constraint and matching quality is the most important design metrics. This results in FCV to have a better estimated cost than EEPSM. From this the FCV algortihm is chosen.

It should also be noticed that the EEPSM algorithm requires a lot of memory since for aggregation of each pixel has to be kept saved in memory whereas the FCV algorithm can function like a running window.

Since the result for EEPSM is significantly worse than FCV it could indicate that the implementation of the EEPSM in python have errors but FCV is still chosen due to time constraint for the thesis.

4.6 Wrap-up

Two edge preserving stereo vision algorithms have been described. The algorithms have been simulated using Python. The results shows that the FCV algorithm have a lower execution time and matches better than the EEPSM algorithm. But the results may indicate that the author have a better understanding of the FCV algorithm which gives this algorithm an edge. Regardless the FCV algorithm is chosen.

Chapter 5

Platform analysis

This chapter the hardware platform provided for this is described. For this project HSA systems have provided a Zedboard Development Board [1]. This board is used for this project since it contains a Zynq Z-7020 SoC from Xilinx, which HSA Systems uses for multiple products. The Zynq SoC contains a ARM® Processing system and 7 series programmable logic (FPGA). This thesis project will focus on the programmable logic of Zynq SoC.

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 [19]

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. 3) and the blue lines in figure 6.1 shows where this movement occurs.

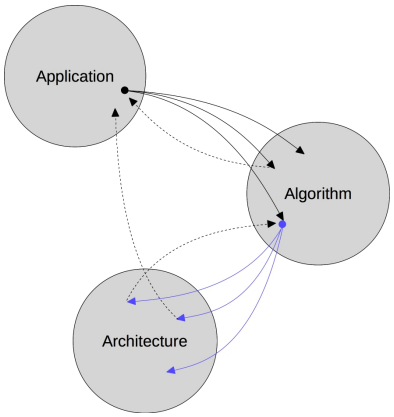
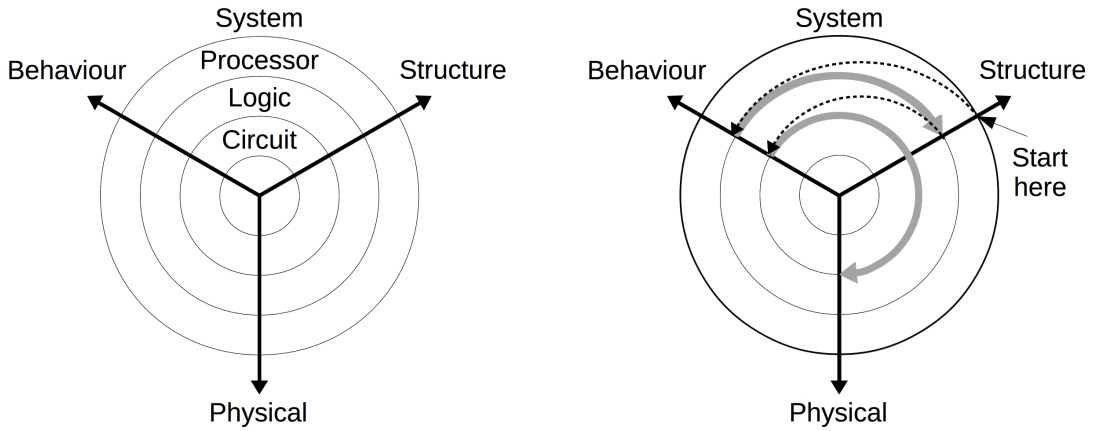


Figure 6.1: A^3 model with the movement from algorithm domain to architecture domain highlight

In [**gajski**] the Gajski-Kuhn Y-chart is described and it is illustrated on figure 6.2 (p. 40). This chart is structured design method which can help structure the design processes for creating a 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 a system. Going from lowest abstraction level to the higher abstraction level this domain starts being described by transfer functions and at higher levels it will be described by algorithms.
The structural domain describes systems using subsystems and their interconnection. Going from the lowest abstraction level to the higher abstraction levels this domain is first



(a) Illustration of the standard Gajski-Kuhn Y-chart [4]

(b) Illustration of the Gajski-Kuhn Y-chart showing the FPGA methodology [4]

Figure 6.2: Illustrations of Gajski-Kuhn Y-chart

described with transistors and at higher levels is will be described using subsystems, FSM, and Data-paths.
The physical domain describes how the system is physically put together and is at the lowest levels described using transistor layout and at higher levels it will be described using chips, boards etc.

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 implementation 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.
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 physical domain. Since it starts at a higher abstraction level the design process is simpler and it is easier to optimize the system but the cost function for the system is only available in the end of the design process.

Another methodology is the FPGA based methodology shown on figure 6.2b which is a combination of bottom-up and top-down.

Chapter 7

Architecture design

This chapter will explore the architecture domain in the a^3 model described in section 1.6 (p. 3) and it is the domain marked on figure **??** (p. ??). The design process in this chapter

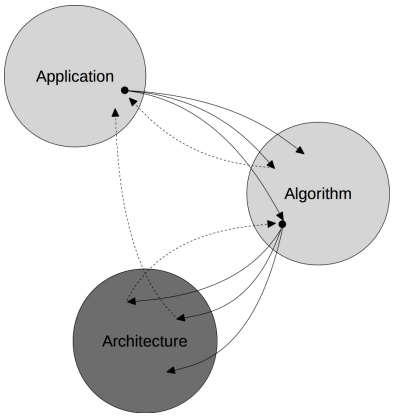


Figure 7.1: A^3 model with the architecture domain marked

Chapter **??** (p. ??) have explored the structural domain at the highest abstraction level and the next highest abstraction level in the bahivoural domain have be explored in chapter **??** (p. ??) where a algorithm were described and chosen. This chapter will start by exploring the next highest level in the structural domain of the Gajski-Kuhn Y-chart.
figure **??** (p. ??) shows the guided image filter
To simplify this chapter only a gray scale version of the guided image filter will be looked at. Figure **??** (p. ??) shows the guided image filter and the blocks it consists of.

7.1 Parallelism Analysis

The inherent parallelism of the system has been analyzed to find out which improvements can be made. Figure xx shows a diagram of the whole system and on the following pages, data flow graphs (DFG) for each subsystem can be found.

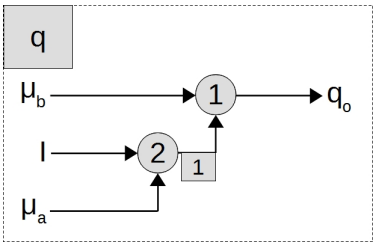


Figure 7.2: Synchronous data flow graph of the q block in the guided image filter

To find the parallelism in the system precedence graphs (PG) have to be created. Since the subsystems are simple the precedence graphs can be created looking at the system from end to start and for each operator find out which signal is needed and have to be calculated before. 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 7.2. In this example the topology matrix is then:

$$\underline{\underline{\Gamma}} = \begin{matrix} \\ \text{arcs} \end{matrix} \begin{matrix} \text{nodes} \\ \begin{bmatrix} -1 & 1 \end{bmatrix} \end{matrix} \tag{7.1}$$

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 $\underline{\underline{\Gamma}} \cdot \underline{q} = \underline{0}$. Then resulting $\underline{q}$ is:

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

This $\underline{q}$ vector expresses how many times each node have to be executed within one sample 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

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 (p. 42) some precedence graphs are found and theses graphs shows how many FUs can run in parallel. We need to know how much hardware is needed for each FU. To calculated some simple system have been generate in Vivado 2016.2 and then synthesized. Then utilization of the hardware can be found. Appendix A (p. 59) describes in detail the procedure and table **??** shows the result.

	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 7.1: Number of logic elements used in average for each FU

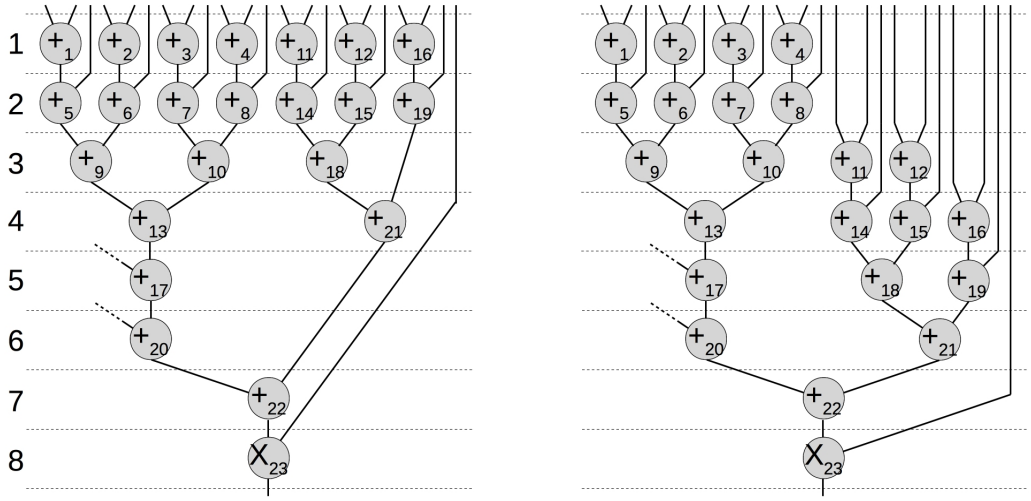
To create the FSM specification we need to introduce time into the precedence graphs to establish some states. To achieve this scheduling is used. There exists different methods for scheduling and some of these are:

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

7.2.1 RC scheduling

For both methods the first step is 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 schudules, e.i. $S_{ALAP}(op) - S_{ASAP}(op)$.
Figure 7.3 (p. 44) shows an example of an ASAP and an ALAP schedule. As seen on the figure node 1-10, 13, 17, 20 and 22-23 are part of the critical path and therefore mobility is 0 for each of the nodes. Nodes 11-12,14-15,18 and 21 all have a mobility of 2 and nodes 16 and 19 have a mobility of 3.

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 2 ($M(+_2) = 0$), 3. node 3 ($M(+_3) = 0$), 4. node



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

Figure 7.3: Illustration of ASAP and ALAP schedules. The illustration shows a part of the mean filter shown on figure **??** (p. ??)

4 ($M(+_4) = 0$), 5. node 11 ($M(+_{11}) = 2$), 6. node 12 ($M(+_{12}) = 2$) and 7. node 16 ($M(+_{16}) = 3$).

Lets say the system have 3 ALU's available then node 1-3 can be scheduled to state 1 since they are higher on the ready list and there is no free ALU for the remaining nodes. 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 4 ($M(+_4) = 0$), 2. node 5 ($M(+_5) = 0$), 3. node 6 ($M(+_6) = 0$), 4. node 7 ($M(+_7) = 0$), 5. node 11 ($M(+_{11}) = 2$), 6. node 12 ($M(+_{12}) = 2$) and 7. node 16 ($M(+_{16}) = 3$).

Then nodes 4-6 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 7 ($M(+_7) = 0$), 2. node 8 ($M(+_8) = 0$), 3. node 9 ($M(+_9) = 0$), 4. node 11 ($M(+_{11}) = 2$), 5. node 12 ($M(+_{12}) = 2$) and 6. node 16 ($M(+_{16}) = 3$).

Nodes 7-9 are scheduled into state 3 and the list is updated again: 1. node 10 ($M(+_10) = 0$), 2. node 11 ($M(+_{11}) = 2$), 3. node 12 ($M(+_{12}) = 2$) and 4. node 16 ($M(+_{16}) = 3$).

This is repeated until all nodes have been scheduled and the result is seen in figure 7.4 (p. 45). This schedule uses 2 states more than the ASAP and ALAP but it reduces the cost since it uses 1 less ALU than the ALAP schedule and 5 less than the ASAP schedule.
This information is described in [4].

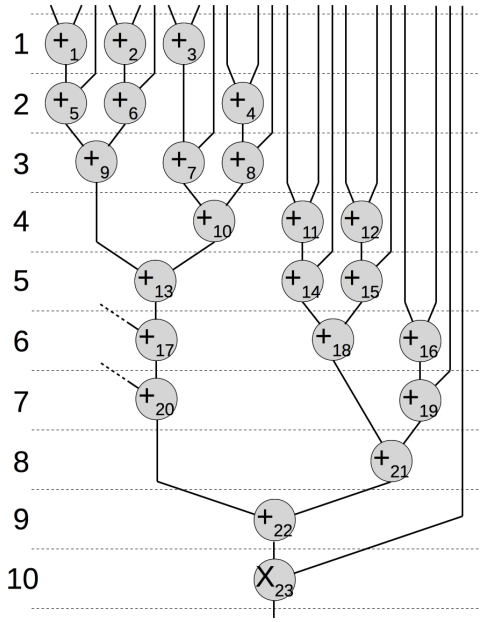


Figure 7.4: RC schedule

7.2.2 TC schedule

The RC schedule improves the cost of the implementation while the TC intends to improve the 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. operation 11 in 7.3 (p. 44) have 1/3 probability for being scheduled in each of state 1-3 if maximum number of states is kept at 8. Figure 7.5 (p. 46) shows the probabilities for each operation in figure 7.3 (p. 44). Fortsæt med at skrive senere

7.3 Finite State Machine with Data Path design

7.3.1 FSMD example

This section will contain an example of a FSMD design example but only of a small part of the system to show have to design a FSMD but also to show that it is a really extensive process. For this example the *b* block (see figure 7.18 (p. 51)) from the guided image filter (see figure 7.12 (p. 50)) and it is shown here on figure 7.6a (p. 46).

```
-- test
...
begin
  process (clk) -- Sequential process
```

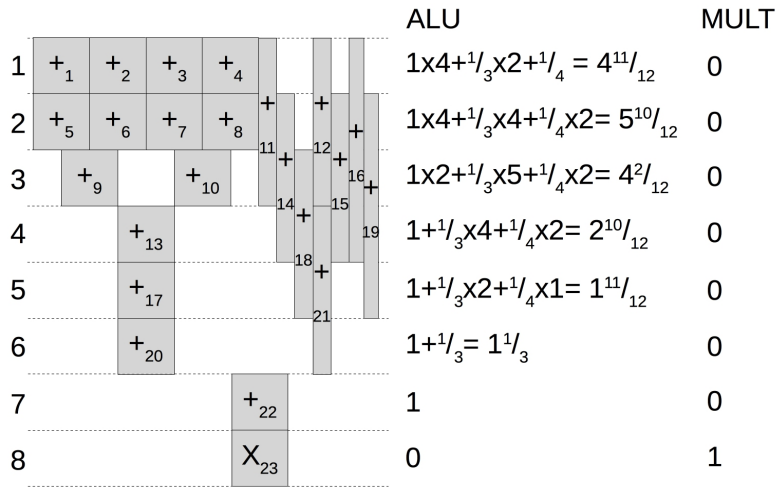
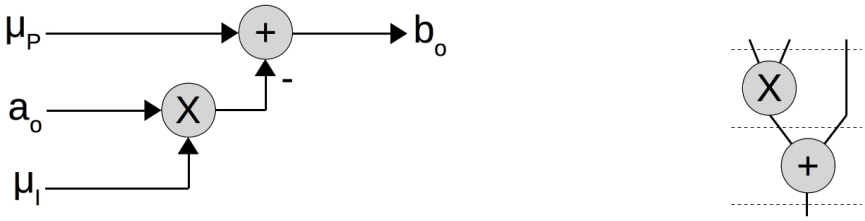


Figure 7.5: Probabilities for each operation in figure 7.3 (p. 44)



(a) *b* block

(b) Synchronous State Diagram of the *b* block

Figure 7.6: *b* block FSMD stuff

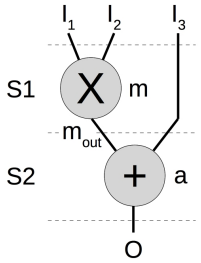


Figure 7.7: Synchronous State Diagram with every connection, functional units, in- and outputs named

```
begin
  if rising_edge(clk) then
    state <= next_state;
  end if;
end process;
```

	S_1	S_2
I_1	√	
I_2	√	
I_3		√
m_{out}		√
O		√

Table 7.2: Life-time analysis

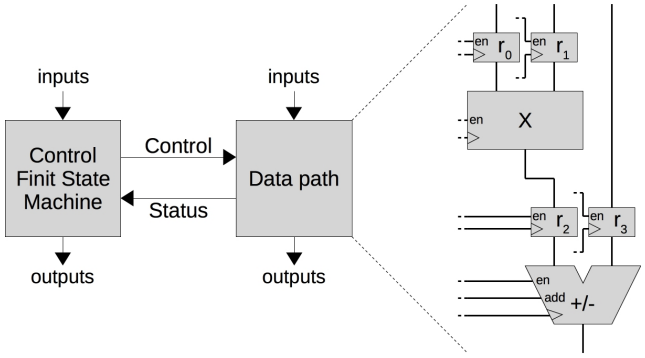


Figure 7.8: Finite State Machine with Data path and the design of the data path is shown.

```
process (state, reset, cars, short, long) -- Combinational process
begin
  if reset = '1' then
    next_state <= S1;
  else
    case state is
      when S1 =>
        m_out = I_1 * I_2;
      when S2 =>
        r_2 <= m_out;
        O <= r_2-I_3;
    end case;
  end if;
end process;
```

noter til mig selv

Controller <–signaler–> Data path
Controller:
Data path: indeholder register, FU, connections, network osv.

skal jeg lave et eksempel med en lille del af mit system? så kan man se hvor voldsomt det kan blive og se processen. Derefter kan jeg vise hvad jeg kommer frem til
noter fra møde med peter:
- lifetime analysis
- synchronous diagram

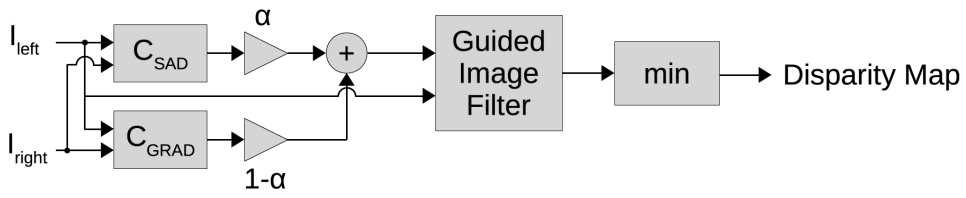


Figure 7.9: Data flow graph of the whole system

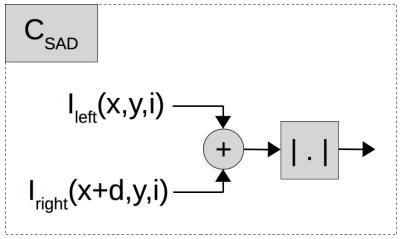


Figure 7.10: Data flow graph of the C_{SAD} block in figure 7.9

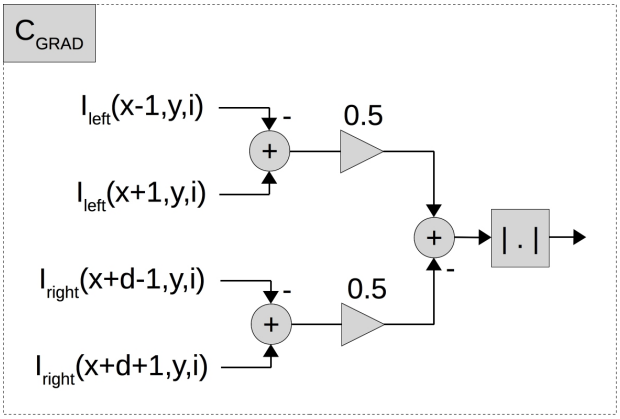


Figure 7.11: C_{GRAD} ikke sikker

noter til mig selv

7.4 VHDL + Simulation

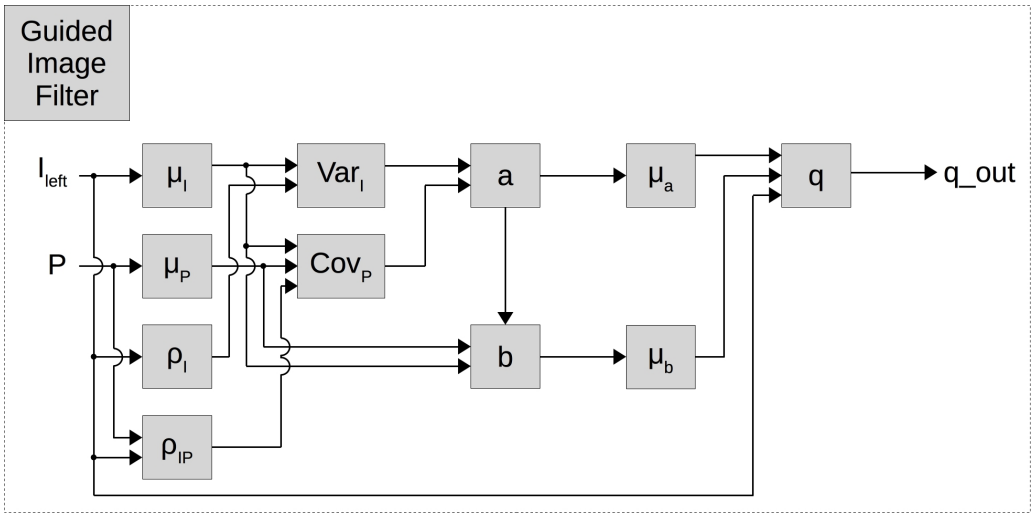


Figure 7.12: Guided image filter

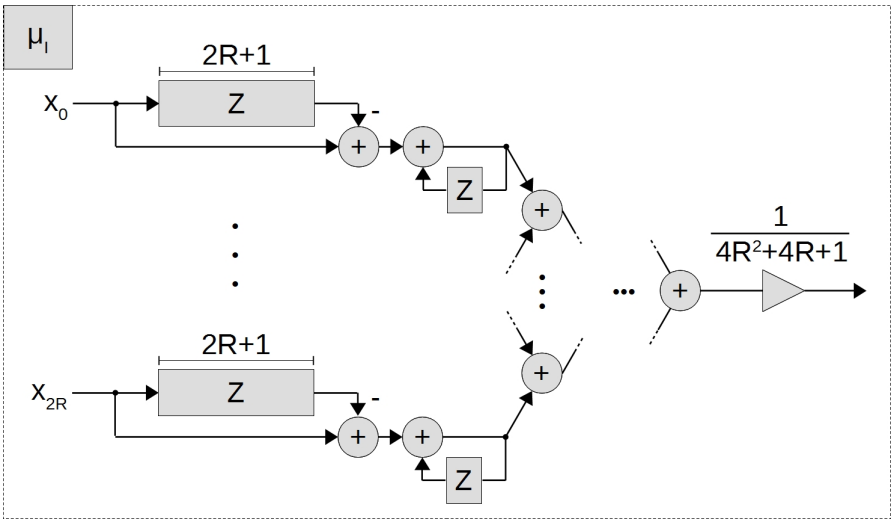


Figure 7.13: Mean filter

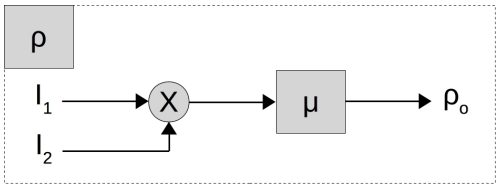


Figure 7.14: Correlation

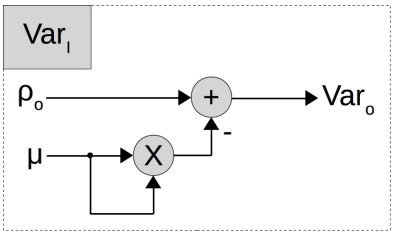


Figure 7.15: Variance

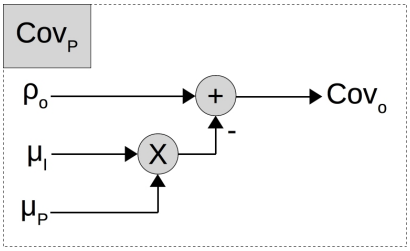


Figure 7.16: Covariance

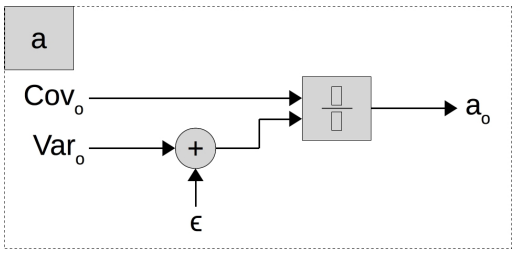


Figure 7.17: A box

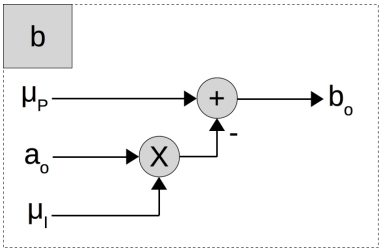


Figure 7.18: B box

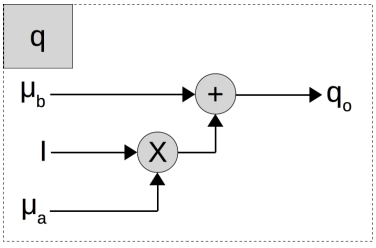


Figure 7.19: Q box

Chapter 8

Acceptance test

Chapter 9

Conclusion

This chapter will contain the conclusion

Bibliography

[1] 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.

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

[3] 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.

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

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

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

[7] 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.

[8] HSA systems. HSA systems. http://hsasystems.com/. 2016.

[9] 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.

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

[11] 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.

[12] 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 2 1 (2003), pp. 195–202.

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

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

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

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

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

[18] 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.

[19] 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 **??** (p. ??) 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. 59).
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

Extra Figures

noter til mig selv

This chapter will contain extra which might fill to much in the report. Looking at you precedence graphs. Maybe make it as fold out image as we did with large diagrams in earlier reports.

Appendix C

Middlebury data set

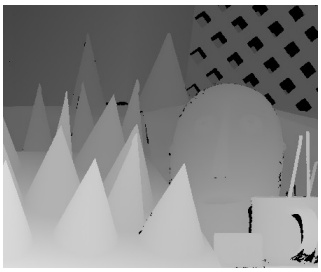
This appendix contains a brief description over the data sets from Middlebury. The computer vision department at Middlebury College have a large library of stereo pair with ground truth[14]. These data sets are used all around the world for evaluating stereo vision algorithms. This thesis uses a subset 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 C.1: Tsukuba - 384 $\times$ 288 [11]

(a) Left image

(b) Right image

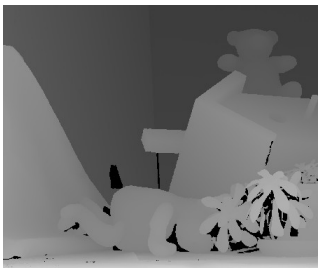


(c) Ground Truth

Figure C.2: Cones - 450 × 375 [12]

(a) Left image

(b) Right image

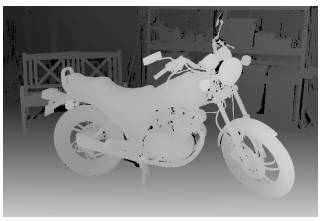


(c) Ground Truth

Figure C.3: Teddy - 450 × 375 [12]

(a) Left image

(b) Right image



(c) Ground Truth

Figure C.4: Motorcycle - 741 × 497 [13]