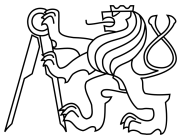
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Multicamera 3D Image Reconstruction in Dynamic Scenes
Student:	Bc. Jan Blažíček
Supervisor:	Ing. Libor Přeučil, CSc.
Study Programme:	Informatics
Study Branch:	System Programming
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2016/17

Instructions

1. Conduct a state-of-the-art study in the field of 3D scene reconstruction using multiple camera views. Consider methods applicable to a set of stationary cameras scanning near flat surfaces under forward motion with emphasis on lowering the overall computational complexity using statistical and graph theory and embedded system-based preprocessing.

2. Design, develop and implement an algorithm for the recovery of intensity images and scene depth. Apply the method to 3D reconstruction of vehicle undercarriage moving over a set of high frame-rate and high horizontal resolution stationary cameras.

3. Prepare a demonstrator of the abovementioned method allowing performance verification of the algorithm design. Conduct quantitative evaluation of precision, highest speed and overall robustness of the implemented approach with regards to this as of yet open problem.

References

Will be provided by the supervisor.

L.S.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague January 11, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Master's thesis

Multicamera 3D Image Reconstruction in Dynamic Scenes

Bc. Jan Blažíček

Supervisor: Ing. Libor Přeučil, CSc.

1st July 2016

Acknowledgements

I would like to thank my family for their undying patience and support, my supervisor for his insights regarding my research, my employers for their flexibility, my dog for just being the happiest little creature on the face of the Earth and my friends for making sure I remember to switch off once in a while. Special thanks goes to my friends: Vít for his tireless attempts at making me drink, Hanka for making me get to work every morning and to Tereza for making me stop each evening.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In V Praze on 1st July 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Blažíček, Jan. *Multicamera 3D Image Reconstruction in Dynamic Scenes.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Práce se zabývá problematikou rekonstrukce hloubkové informace v obraze získaném ze dvou vzájemně synchronizovaných a kalibrovaných kamer uložených ve stejné horizontální rovině. Speciální důraz je kladen na rozšiřitelnost aplikace pro možnost využití více kamer, efektivní předzpracování instance úlohy, optimalizaci pro běžně dostupné počítačové systémy a paralelizovatelnost tak, aby se vytvořil základ pro systém využitelný v reálně časových úlohách.

Klíčová slova disparita, hloubka, reálný čas, počítačové vidění

Abstract

This diploma thesis deals with the question of depth data reconstruction in a two mutually synchronised and calibrated camera setting mounted on the same horizontal plane. Special emphasis is placed on application extensibility in regards to using multiple cameras, effective preprocessing of the problem instances, optimalization for commonly available computer systems and parallelizability in order to create a basis for a system useful in real-time tasks.

Keywords disparity, depth, real-time, computer vision

Contents

List of Figures

List of Tables

Introduction

Stereopsis, the ability to perceive the world as a 3-dimensional structure, is basis for much of our success as species. A point could be made that it is one of the characteristic abilities of most known intelligent life on our planet. It is therefore only natural that we would try to equip our technological counterparts with a system equally as good, if not better. Even if we disregard the evolutionary advantage such an ability poses for robotics like the possibility of self driving cars incorporating depth perception in most of their navigational features, adaptive cruise control and various safety systems on existing vehicles, we can still imagine numerous mundane tasks difficult to carry out without it like precision manufacturing consistency measurements, which would all be, if not outright impossible, at least very much limited in their capabilities.

Computer stereo vision - the problem of obtaining depth information by comparison of multiple digital images of the same scene - is still very much an open issue. Numerous techniques attacking the problem from different standpoints exist and usually only solve one specific limited case. Stereo vision as a tool has found itself being gradually replaced by modern and usually very expensive technologies like Light Detection And Ranging (LIDAR) systems, various takes on Light Coding as used in Microsoft Kinect devices, Light Field cameras like Lytro etc., which undoubtedly have their place in the world of Computer Vision. For all their strengths however there is still a case to be made for classical stereoscopic vision, which can, under certain circumstances, provide us with more precise results and, depending on our requirements, for a fraction of the cost.

Numerous papers have been published on the issue of obtaining quality depth imagery from stereoscopic systems. In this work we would like to address the less commonly attacked issue of reducing the computational complexity of such systems on modern personal computers by porting existing embedded solutions and by making calculated sacrifices in result quality to a point where a base for a usable near real time solution is discovered.

Main contributions

The ultimate goal of this thesis is to explore the possibility of building a stereoscopic system, which could potentially solve the issue of scanning the undercarriages of moving cars for possible foreign objects posing security threats like bombs. We therefore want to obtain depth information for cases where the scanned object is relatively near to the camera set, the cameras are mounted on a single plane and they are oriented in the same way with a certain degree of overlap between their respective fields of view (FOV). Because there is a very small vertical window in which we can operate, we are mostly concerned with getting precise information in the middle 5% of the camera vertical resolution.

The main contributions of this work are the following:

- Guidelines on lens and stereo camera set calibrations for stereoscopic vision.
- An overview of existing global and local 3D from stereo algorithms with emphasis on local methods.
- In-depth description and implementation of a Census algorithm based local method.
- Exploration of possible computational complexity and state-space size reductions.
- Evaluation of the effects of various requirement relaxations on the precision and computational speed.
- Experimental evaluation of the proposed final algorithm on publicly available datasets and stereo imagery obtained with our own setup of stereo cameras.

Thesis structure

This thesis is organized as follows: Chapter 1 outlines the current state of development in the field of obtaining 3D information from stereoscopic digital imagery. Chapter 2 further discusses the theory behind our method of choice and reasons behind its selection. Chapter 3 discusses the hardware and software used in the development of our system and describes the implementation details of the selected method and proposes methods for parallelization and reductions in both computational speed and memory consumption. Chapter 4 evaluates the algorithm accuracy and performance on various datasets. Chapter 4.3 touches on the possible directions on future research and the conclusions of the work behind this thesis.

Chapter 1

State of the Art

This section describes the current state of development in the relevant fields touched by this thesis. In section 1.1 we will take a look at the currently used global stereo vision methods, their strenghts and weaknesses. In section 1.2 we overview the so called local stereo vision methods more relevant to our application. Section 1.3 briefly explores some of the currently used methods of obtaining depth information using a single camera or camera like systems.

1.1 Global stereo correspondence methods

Global stereo vision methods are currently the top choice in the field of precise 3D stereographic reconstruction. They generally do provide us with more precise results and greater overall robustness, but at the cost of very high computational complexity and memory requirements. Global methods are usually based on methods initially intended for solving diametrally different problems and very general algorithms. They are therefore not the most efficient choice, but at the same time they are well understood and allow for the application of similar optimization methods employed in graph theory applications and similar fields. Some of the most popular global methods include:

1.1.1 Simulated annealing

Simulated annealing, first described in [2], is a global optimum approximation method. It is mostly employed in situations where we don't require a precise result, but rather a local optimum in limited time frame. The method itself is a probabilistic one. Starting from an arbitrary location in the state space the algorithm jumps to different locations. As time passes the algorithm is progressively less likely to accept a worse solution. The ingenuity of the algorithm lies in that it does not necessarily only explore solutions that are better than the ones already found, which would inevitably lock the algorithm in a local optimum were it to start from a position not within reach of a global

one, but that it intentionally allows bad moves in order to explore greater part of the state space. Once a heuristically interesting part of such state space is identified or certain amount of time passes, it then lowers the temperature, which makes the algorithm a bit more conservative with its following choices.

Mathematically speaking, the algorithm uses the following acceptance probability function:

$$P(e, e', T) = \begin{cases} 1 & e' < e \\ e^{\frac{e-e'_i}{T}} & \text{else} \end{cases}, \tag{1.1}$$

where T is the current temperature and e and e' respectively are the energies (or rather the scores) of the current and the explored states. This means that we'll always accept a solution that is better than the current one and we may accept a worse solution with a nonzero probability given by its energy and the current temperature.

When it comes to stereoscopic depth estimation, the article [3] describes a simulated annealing based algorithm. The authors define an energy function:

$$U(p) = \sum_{n=1}^{5} \gamma_n U_n(p), \tag{1.2}$$

where p is a point in the image, γ_n are control parameters and U_n are five different terms, each one representing a different aspect of the cost. Without delving too deep into the math behind these terms here they represent different metrics for a shifting window around the current pixel position, which is an approach based on individual local methods described further in their respective sections. Term U_1 is the Sum of Absolute Differences (SAD), U_2 is a Census based metric, U_3 is the correspondence between edge images.

U_4 represents a smoothness constraint, which is worth explaining here. We can assume that the edges found in the images represent real edges on the scanned object. Outside of these special regions, we can expect the disparity to change gradually as edges are likely the only place where either one object ends and another begins, or the angle between two adjacent surfaces changes abruptly. Given three positions p, q and r in the disparity image we can write the smoothness constraint as:

$$U_4(p) = (D(p) - D(q))^2(1 - vL(p))) + ((D(p) - D(r))^2(1 - vL(r)), \tag{1.3}$$

wherever $(p_x = q_x) \wedge (q_y = p_y - 1) \wedge (p_x = r_x) \wedge (r_y = p_y + 1)$. $vL(p)$ and $vL(r)$ is defined as 1 when there is a vertical edge at a given position and 0 oterwise, D denotes the disparity. As you can see, this term only considers horizontal smoothness.

The last term U_5 is called the uniqueness constraint and it simply states that no two points in the left image can be projected onto the exact same point in the right image.

The simulated annealing then follows the algorithm 1.

Algorithm 1 Simulated annealing disparity calculation

```
1: function SimulatedAnnealing(I_left, I_right, Disparities)
2:     Assign initial temperature T
3:     for A limited number of iterations do
4:         for Each pixel in Disparities do
5:             Change the disparity value to another value within range.
6:             Calculate ΔU
7:             if (ΔU < 0) ∨ (e^(−ΔU/T) > ξ), 0 ≤ ξ ≤ 1 then
8:                 Accept the new value.
9:             end if
10:            Cool T by 0 < k < 1, so that T_{k+1} = k · T_k
11:        end for
12:    end for
13: end function
```

Further modifications to the algorithm can be found in [4] where the authors improve it by taking colour into account. Generally speaking, this simulated annealing algorithm and its modifications with differing energy functions produce precise results, because they take into account multiple different descriptors of a given image pair. The sheer volume of the state space combined with the amount of calculations required for every calculation of the state energies and the initially erratic behavior of the algorithm results in very long calculation times even for small images, rendering the method unusable for our purpose.

1.1.2 Graph Cut

Graph Cut algorithms are another example of a general algorithm applicable to the problem of stereo vision. More precisely we should talk about so called Max-Flow / Min-Cut optimization algorithms. These algorithms are based on the Max-Flow / Min-Cut theorem found in [5, Chapter 6.1], which says that the maximum flow in a network passing from the source to the sink is equal to the minimum capacity that can be removed to disrupt such network.

Applying the idea to the stereo correspondence problem [6], the resulting algorithm once again uses energy minimization to compute the resulting disparities. We're attempting to identify a disparity - a projection from pixel $p_1 \in \mathcal{I}_{1\text{left}} = [p_{1x}, p_{1y}]$ to a pixel $p_2 \in \mathcal{I}_{2\text{left}} = [p_{2x}, p_{2y}]$ in the right image, where $\mathcal{I}_{\text{left}}$ and $\mathcal{I}_{\text{right}}$ are the respective left and right input images. To do so,

we're trying to minimize the so called Potts energy:

$$E(f) = \sum_{p \in \mathcal{I}_{\text{left}}} D_p(f_p) + \begin{cases} \sum_{p,q \in \mathcal{N}} V_{p,q} & f_p \neq f_q \\ 0 & \text{else} \end{cases} \tag{1.4}$$

here $D_p(f_p)$ denotes the penalty for pixel p having disparity f_p, N is set of pairs of pixels p and q in the left image, which makes the second part of equation 1.4 a smoothing term, which penalizes two neighboring disparities for being too different.

Energy minimization involving equation 1.4 is an NP-complete problem, therefore approximative algorithms are used. Described in [7] the basic algorithm uses the so called $\alpha - \beta$ swap moves. α and β here denote disparities, usually called labels in papers discussing graph cuts even in the context of depth reconstruction to preserve consistency with the original intended purposes of the graph cut algorithm. The $\alpha - \beta$ swap algorithm 2 allows us to find a labeling $\hat{f}$ for a pair of labels α and β that minimizes the energy function E over all labelings within one $\alpha - \beta$ swap of f.

Algorithm 2 $\alpha - \beta$ swap move algorithm

```
1: function α − β SWAP(labeling f)
2:     success = 0
3:     for Each pair of labels {α, β} ∈ L do
4:         Find f̂ = arg min(E(f')) among f' in one α − β swap of f
5:         if E(f̂) < E(f) then
6:             f = f̂
7:             success = 1
8:         end if
9:     end for
10:    if success = 1 then
11:        GOTO 2
12:    end if
13: end function
```

The $\alpha - \beta$ swap algorithm requires an interaction potential V to be semi-metric on a space of labels $\mathcal{L}$, which means the following two conditions have to be satisfied:

$$V(\alpha, \beta) = V(\beta, \alpha) \geq 0, V(\alpha, \beta) = 0 \Leftrightarrow \alpha = \beta, \tag{1.5}$$

satisfied for any $\alpha, \beta \in \mathcal{L}$.

If we further satisfy the condition in eq. 1.6 - the triangle inequality, we can call a potential a metric. Given a metric potential, we can define an α-expansion algorithm3.

$$V(\alpha, \beta) = V(\alpha, \gamma) + V(\gamma, \beta), \tag{1.6}$$

satisfied for any $\alpha, \beta, \gamma \in \mathcal{L}$.

Algorithm 3 α-expansion algorithm

```
1: function α-EXPANSION(labeling f)
2:     success = 0
3:     for Each label α ∈ L do
4:         Find f̂ = arg min(E(f')) among f' in one α expansion of f
5:         if E(f̂) < E(f) then
6:             f = f̂
7:             success = 1
8:         end if
9:     end for
10:    if success = 1 then
11:        GOTO 2
12:    end if
13: end function
```

The α-expansion algorithm also finds a labeling $\hat{f}$ that minimizes the energy function E over all labelings within one α expansion of f. That means that the algorithm segments all α and non-α disparities with graph cuts, while iterating through all the possible α labels until it converges.

1.2 Local stereo correspondence methods

Local stereo correspondence algorithms can be looked on as different metrics solving the same problem in a similar fashion but using different operations. They all have their downsides and benefits. Generally, they use the notion of a stationary window around a point (pixel) in one of the image pair which we somehow compare to a shifting window of the same dimensions in the other image. We then select a disparity based on a difference metric the algorithm defines. The lower the difference, the more likely is the current center position of the left window to correspond to the center of the right window.

There are numerous considerations when selecting the appropriate size, shape and other properties of the windows. These are dependent on the metric used. The above generalized approach doesn't tell the whole story, the algorithms are usually much more involved to solve problems like occlusions, disparity uniqueness and the tendency to gradually assign smaller disparities the closer the stationary window is to the end edge of the image. These techniques are either highly specific to the one particular implementation of the chosen algorithm and therefore unimportant for the purpose of this work or they can be very well generalized to all of the below methods, we will discuss those in a subsection of their own.

We will only concern ourselves with algorithms intended for dense disparity map computation - these are generally simpler and faster than those intended for singled out point features and can be scaled more readily, allowing us to maintain a good amount of precision while still keeping the computational cost down.

1.2.1 Sum of Squared Differences (SSD)

The Sum of Squared Differences algorithm is a general purpose algorithm, based on a statistical method known as Mean squared error in other fields than computer vision like signal processing. It is, at its core, a squared l_2-norm. It is one of the simplest and most effective similarity measures in template matching. A popular choice for practically all tasks requiring correlation such as video encoding and related motion estimation in the h.264 standard [8], the basis for the SSD algorithm is the following equation:

$$\sum_{i,j \in W} (\mathcal{I}_1(i,j) - \mathcal{I}_2(x+i, y+j))^2, \tag{1.7}$$

here the window W is any shape, the center of said window in the first image $\mathcal{I}_1$ is the pixel we're currently in search of in the second image $\mathcal{I}_2$. The window is usually square or rectangular with odd dimensions in both the X and Y directions in order to simplify its placement. The x and y parameters denote the currently considered disparity.

In the vast majority of cases when working with stationary cameras our input images have already been corrected for lens distortion and stereoscopically rectified, therefore we only consider displacements in the X direction. The algorithm systematically works through the first input image sliding the correspondence window across its lines. The exact motion is usually dependent on the implementation and the format of data storage, but without loss of generality we can assume the algorithm first moves in the X direction, then moves to a line below in the Y direction and then starts in the X direction from the beginning of the line.

We can now limit ourselves to a single pixel $[x_1, y_1]$ in the reference (first) image and its surrounding window W_1. The algorithm then searches through the second image for a matching pixel. This is easy assuming the difference between the two frames isn't great and assuming no specularities, visibility occlusions and matching camera settings. We slide the shifting window W_2 across the line $y_2 = y_1$ in the second image and compute the SSD value according to equation 1.7 for each pixel. We then find the smallest difference across the entire line y_2 in the second image and read the corresponding x_2 displacement as seen on image 1.1. The disparity x can then be computed as $x = x_2 - x_1$.

The SSD algorithm brings along with it a major disadvantage though, which is the computational cost. As with most window based local algorithms,

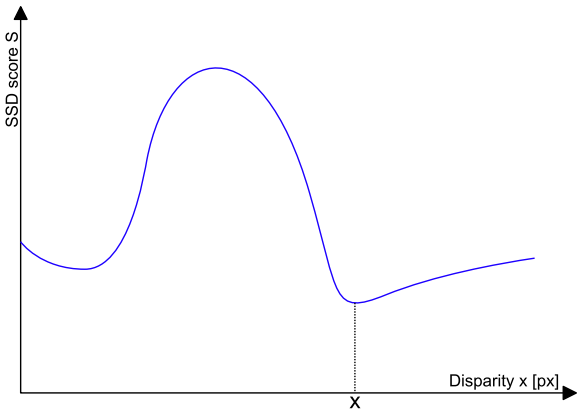


Figure 1.1: Disparity from SSD score minimum

we have to go through $X \cdot Y$ pixels in the first image, for each of those pixels we have to move through X pixels in the second image, which brings us to X^2Y. Given a window W size of $m \times n$ pixels, we then have to compute the difference between mn pixels for each of them and square each one. For the sake of simplicity, let's assume each of the addition, difference and square operations takes a unit time, we then arrive at an upper complexity limit of $O(X^2Y \cdot mn)$. Of course there are numerous applicable optimizations, but there are also multiple necessary steps required to obtain a robust result. Both will be discussed further.

Last but not least important consideration when working with SSD is its tendency to overemphasize the importance of image noise and outliers. See image 1.2 for details.

1.2.2 Zero-mean Sum of Squared Differences (ZSSD)

This SSD algorithm mutation enables us to partially suppress one of the other major flaws of the above-described approach. The SSD algorithm in its simplicity doesn't concern itself with the absolute intensity changes between the two frames. This however is a fairly common issue because of directional light reflection, lens vignetting etc.

The Zero-mean version of the SSD algorithm deals with this by subtracting the local area mean from the elements before calculating the squared differences. We can write the score as:

$$\sum_{i,j \in W} (\mathcal{I}_1(i,j) - \overline{\mathcal{I}_1}(i,j) - \mathcal{I}_2(x+i, y+j) + \overline{\mathcal{I}_2}(x+i, y+j))^2, \tag{1.8}$$

where $\overline{\mathcal{I}_1}$ and $\overline{\mathcal{I}_2}$ denotes the respective means over the window areas.

This approach naturally increases the computational complexity of the algorithm. Thankfully, we can easily and effectively precalculate the means

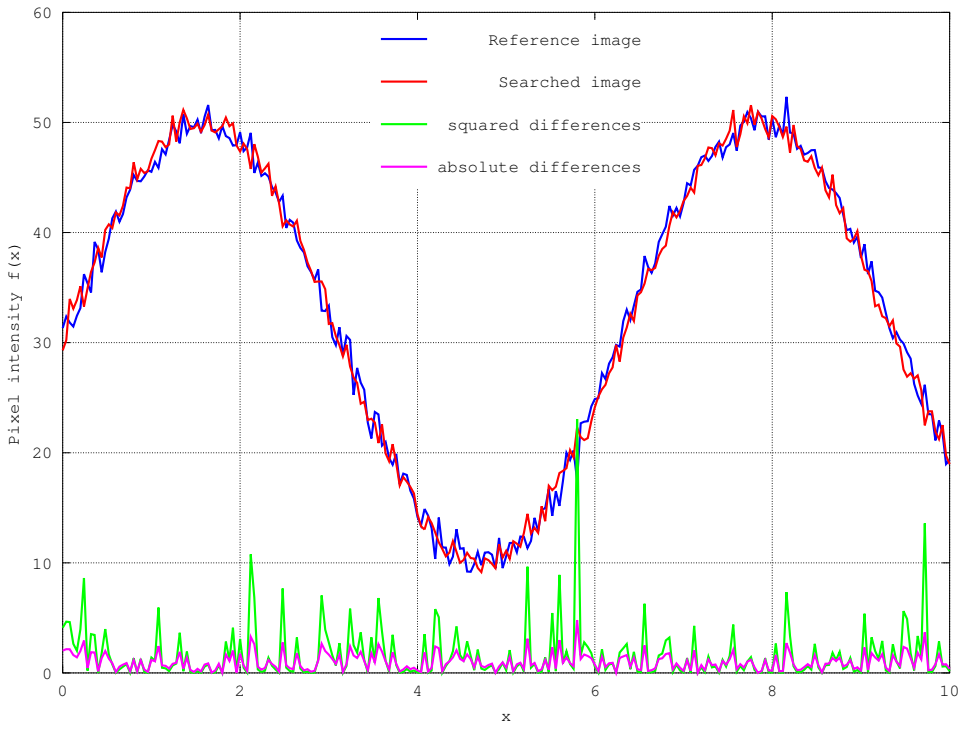


Figure 1.2: Noise enhancement tendency SD vs. AD. There is a clear difference between squared differences and absolute differences.

using parallelized algorithms running on a GPU, therefore the overall time complexity doesn't have to change drastically.

1.2.3 Sum of Absolute Differences (SAD)

Given the high computational cost and the memory requirements of the SSD algorithm a simplification was devised in the form of the Sum of Absolute Differences algorithm. The algorithm works in the same way as SSD, image $\mathcal{I}_1$ is used as a reference and the paired $\mathcal{I}_2$ as the searched counterpart. The metric for SAD is the following:

$$\sum_{i,j \in W} \left| \mathcal{I}_1(i,j) - \mathcal{I}_2(x+i, y+j) \right|. \tag{1.9}$$

SAD is, mathematically speaking, one of the simplest possible metrics useful for image registration and stereo vision. In comparison to SSD the computation time speedups aren't usually enormous. The SSD algorithm is historically the preferred method as it not only leads to simpler differentiation and is more mathematically tractable, but it is also a maximum likelihood estimator for Gaussian distrubuted data in statistics. While SSD generally does

outperform SAD as far as robustness is concerned, the differences are minimal and we can easily justify implementing SAD with a couple of optimizations in its stead.

SAD is the clear winner when it comes to simplicity of implementation. Multiple hardware accelerations such as the Intel SSE2 (Intel's take on a Single Instruction Multiple Data supplementary instruction set) PSADBW command which calculates SAD are commonly available.

Of course, it doesn't solve any of the problems associated with SSD, the algorithm is still sensitive to absolute intensity differences, it doesn't deal with perspective changes, the memory requirements are still quite limiting etc.

1.2.4 Zero-mean Sum of Absolute Differences (ZSAD)

Zero-mean Sum of Absolute Differences attempts to solve the same issues as the ZSSD algorithm described in 1.2.2. It does this once again by subtracting means over the entire window from the individual pixels before calculating the difference. The results and the computational speed are unimpressively similar to ZSSD. The metric used is the following:

$$\sum_{i,j \in W} \left| \mathcal{I}_1(i,j) - \overline{\mathcal{I}_1}(i,j) - \mathcal{I}_2(x+i, y+j) + \overline{\mathcal{I}_2}(x+i, y+j) \right|. \tag{1.10}$$

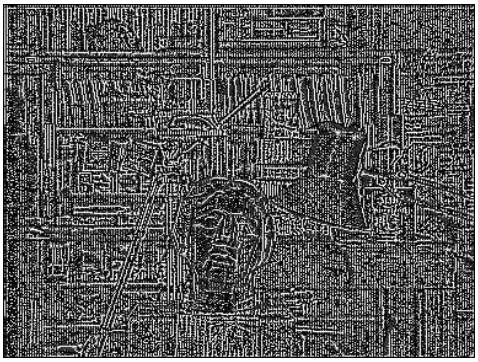
1.2.5 Rank

Rank method deviates from the previous ones in that it is entirely independent of absolute intensity differences between the two compared regions. The method stands on a so called Rank transform, a statistical method usage of which in a Computer Vision stereo correlation setting is described in [9]. It is considered to be a non-parametric transform. Non-parametric local transforms depend on relative ordering of the pixels in a window instead of their actual intensities. They are less sensitive to outlier and noise caused errors than the previous methods, less sensitive to camera gain and bias effects. On the other hand the rank transform does result in certain undeniable pixel information loss and it introduces problems where two completely different image regions with similar relative pixel intensities can result in the same or very similar rank transformed regions.

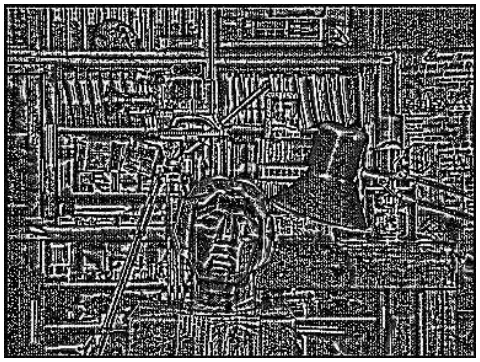
The transform itself is defined in the following way. Given a pixel $c = [x_0, y_0]$ located in the center of a square window W of dimensions $m \times m$, where m is odd and the pixel intensity $f(x, y)$ we can write:

$$R(x_0, y_0) = \sum_{i,j \in W} \begin{cases} 1, & f(x_0 + i, y_0 + j) < f(x_0, y_0) \\ 0, & else \end{cases}. \tag{1.11}$$

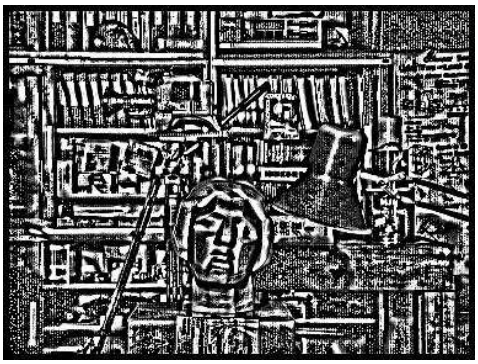
This defines the rank value for each pixel in the image. The correlation itself can then be performed using any of the above methods. The original authors used the L_1-correlation, which means the SAD algorithm.



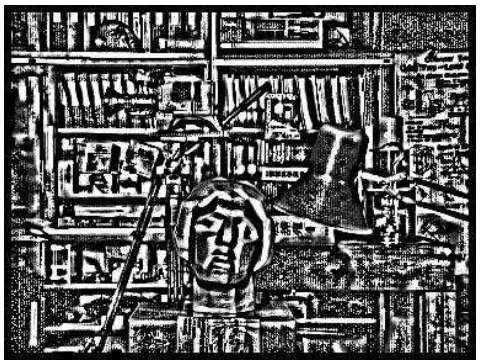
(a) Rank transform - size 3



(b) Rank transform - size 6



(c) Rank transform - size 11



(d) Rank transform - size 13

Figure 1.3: Rank transform for different square window dimensions

One of the problems of the Rank transform is that it is invariant to rotations. We can see in a study [10] an example of multiple completely different patterns that result in the same rank transform. This isn't a problem directly, because we expect the objects to be rotated in the same way between the frames, but it does increase the chance of encountering two different objects, or differently rotated objects that look the same from the standpoint of correlation. Figure 1.3 shows rank transforms for different window sizes. An example of a rank transform can be found on fig. 1.4b.

1.2.6 Complete Rank Transform (CRT)

Complete Rank Transform was introduced in [11] as an answer to some of the Rank transform problems. It solves the ambiguous rotation problems described in 1.2.5 and generally has more information per pixel than the Rank transform does, which results in lower correlation error.

The CRT does in essence the same thing as the Rank transform itself, but for the entire window instead of for the center pixel. An example can be found on figure 1.4c. Given a pixel $p = [x, y]$, anywhere within a window

W, its intensity $f(p)$ and a function lcount(W, p) which returns the number of pixels within the window W with intensity strictly lower than p, we can write:

$$\text{CRT}(p) = \text{lcount}(W, p), p \in W. \tag{1.12}$$

This way every pixel within a given window gets a new value assigned to it. Because we have to do this for every pixel in the image, the size of the image has to increase. If we use a window W with m^2 elements on an image of size $a \cdot b$, we will have to store the new information in a container of size abm^2. That said, we can still use any of the above correlation methods to measure the window similarity if we modify its behavior to use the transformed windows instead of the original images.

1.2.7 Census

This brings us to our method of choice, the Census transform. Once again, this method is entirely independent of absolute intensity differences between the compared windows. The transform is described in [9]. It is closer to the Complete Rank Transform in its implementation than to the Rank transform, in that it defines a new value for every pixel in the window instead of just the center one. It is however simpler than the CRT thanks to the fact that it doesn't concern itself with the actual number of pixels below a threshold intensity, but rather a comparison of any given pixel to the center one.

Given a pixel $p = [x, y]$ anywhere within the window W, a center pixel $c = [x_0, y_0]$ and the pixel intensity $f(p)$, we can write:

$$\text{Census}(p) = \begin{cases} 1, & f(p) < f(c) \\ 0, & else \end{cases}. \tag{1.13}$$

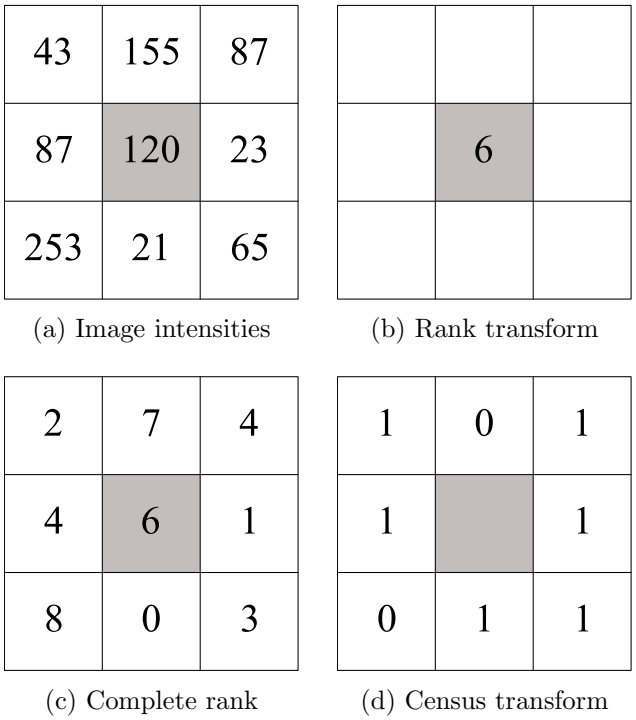
The center pixel can either be left out or set to 0 according to the definition.

One of the reasons we chose this transform is the compact data representation that lends itself to simple parallelization. Census works best with large window sizes and it is rather fast thanks to relying solely on intensity comparisons. Depending on our window size, we can either store the values in integer datatypes or make use of specialized containers like bitsets.

Census however brings along with it a need for yet another correlation metric as it would be ineffective to use SSD or SAD and their modifications. If we store the values inside a single window as a vector of bits, we can make use of the Hamming distance. For two bit strings A and B, this is defined as:

$$H(A, B) = \sum_{i=0}^{n} (A_i - B_i), \tag{1.14}$$

where A_i and B_i are the individual bits of the respective strings. With this exception the correlation works in much the same way as SSD and SAD. The algorithm in its entirety will be discussed further in the chapters 2 and 3.



(a) Image intensities (b) Rank transform

(c) Complete rank (d) Census transform

Figure 1.4: Non-parametric transforms comparison

1.2.8 Normalized Cross Correlation (NCC)

Normalized Cross Correlation, or better the Zero-Mean Normalized Cross Correlation (ZNCC) [12] is a technique normally used for other correlation tasks than stereo. It uses a much more involved metric than the previous local transforms:

$$\frac{\sum_{i,j \in W} (I_1(i,j) - \overline{I_1})(I_2(x+i, y+j) - \overline{I_2})}{\sqrt{\sum_{i,j \in W} (I_1(i,j) - \overline{I_1})^2 \sum_{i,j \in W} (I_2(x+i, y+j) - \overline{I_2})^2}}, \tag{1.15}$$

which does result in large computational complexity. Cross-correlation is a metric which measures the similarity of two signals shifted relative to each other. While it is relatively slow, it is designed to handle both constant gain and constant offset differences between the areas matched thanks to the subtraction of the means of the two areas from each intensity value within the window and to the division by the respective variances. Once again as with the relative measures like Census above, this does introduce some mismatched disparities for similar areas.

Normalized Cross Correlation at first sight lends itself to implementation using a Discrete Fourier Transform (DFT), but the more common option is

direct calculation, especially when it comes to stereo, as the DFT is only faster in some rather special cases unusual for our use. As in previous cases, it is not invariant with respect to imaging scale, rotation and distortion.

Various takes on NCC simplification have been proposed like the Sequential Similarity Detection algorithm (SSDA) [13] which allow for significant speedups, but usually don't guarantee finding the global optimal solution, which makes them hardly useful for complex texture scenes.

The correlation algorithm itself works in much the same way as for all of the above cases, a shifting window is slid across the searched image in order to identify, in this case, a maximum of the metric 1.15. NCC, with all of its shortcomings and despite its age, remains one of the most general metrics to date. It is universal in that it doesn't require any special parameters and no preprocessing of the image data, but it still gives quality results. It is however better suited for correlation of large images, e.g. full frames, instead of small window cutouts.

1.2.9 Mutual Information (MI)

Mutual Information [14] is a dense stereo correspondence similarity metric striving to solve the issue of differing lighting conditions for stereo with a wide baseline and cameras with different spectral responses like infrared and visible light cameras. The metric is a statistical one, relying on the entropy of the image probabilistic densities. All in all it is a very interesting metric with uses which still haven't been all fully explored (e.g. correlation between a pair of positive and negative samples).

Mutual Information relies on entropy and on the joint entropy of a pair of random variables, in this case image pixels taken from both images in the pair. Entropy can be used as a measure of randomness for numerous Computer Vision tasks such as lens focusing, where the image has a lower entropy when it is out of focus while the focused edges increase it. It is defined as

$$H(I) = -\sum_{k=1}^{n} p_k log(\frac{p_k}{w_k}), \tag{1.16}$$

where p_k denotes the bin probabilities of a grayscale image I histogram $h(I)$ and $w_k = 1$ is the width of a histogram bin. Mutual information on the other hand is a measure of mutual dependence between two variables. In layman terms it gives us an idea about the amount of information we can summarize about an image by looking at the other image. It is defined as

$$\mathrm{MI}(I_1, I_2) = E_{I_1,I_2} \cdot log\left(\frac{P(I_1, I_2)}{P(I_1)P(I_2)}\right) = H(I_1) + H(I_2) + H(I_1, I_2), \tag{1.17}$$

where $P(I_1, I_2)$ is the joint probability distribution, $H(I_1, I_2)$ is the joint entropy calculated using a joint histogram containing both I_1 and I_2 histograms in its two channels.

The Mutual information is bounded by the limits $0 < MI(I_1, I_2) < \min(\mathrm{MI}(I_1, I_1), \mathrm{MI}(I_2, I_2))$. It is 0 when the two images are completely independent. It is invariant to bijective projections, which results in it being useful for correlation between different sources of data. The correlation algorithm itself is a simple scanline search as in all other local stereo correspondence methods. A curvature of the similarity curve, defined as

$$\mathrm{Conf} = 2S_{\max} - S_{\mathrm{left}} - S_{\mathrm{right}}, \tag{1.18}$$

where $S_{\max}$ is the MI score of the currently searched pixel and S_{left}, S_{right} are the left and right pixel scores respectively, is used as a confidence metric.

1.3 Single camera alternatives

Without straying too far from the above methods, we'll take a brief look at the current advances in the field of depth estimation using a single camera or camera-like systems. There are numerous more or less successful methods being developed using very interesting ideas which may prove to be useful for systems like ours. They don't always require specialised hardware, but most of them rely on some sort of additional source of data to obtain the depth information. While the methods aren't really comparable to multi-camera systems, they are definitely noteworthy, as they may well represent the real future of computer vision in systems like self driving cars. Their scalability is a bit more problematic at this point in time as increasing the precision usually requires enormous financial investments, but there are affordable low precision / high speed systems available even now.

A very interesting point is that single camera systems do not naturally suffer from occlusion problems as much as their stereo counterparts given their tiny baselines. While there are cases where one part of the sensor may see a part of the image which is occluded from another part, they are rather infrequent and are considered a special case rather than the norm - they usually don't need to be addressed. Even in cases where there are noteworthy occlusions, the way in which they affect the end result is most often negligible as the algorithms don't rely on comparisons between multiple different views, hence the occlusion doesn't result in the algorithm failing completely.

1.3.1 Depth from Focus / Defocus

The focus / defocus is likely the most basic single camera depth estimation algorithm there is. It relies entirely on the physical properties of the camera lens, namely the depth of field. The article [15] offers an interesting comparison of this method to conventional stereo depth estimation methods. There are two basic modifications of the algorithm.

Depth From Focus (DFF)

Depth From Focus is a method that requires access to the focusing system on the camera. As the lens focus changes, objects in different depths from the camera come in and out of the focus at different times. We search for the precise lens system state under which the observed surface is in focus. Focused objects can be identified by a number of algorithms, the most often used measure of focus is the image entropy calculated over a limited window.

The lens can be modeled using a thin lens equation approximation:

$$\frac{1}{f} = \frac{1}{v} + \frac{1}{u}, \tag{1.19}$$

where f is the focal length, v is the distance between the lens plane and the image plane and u is the distance between the object in focus and the lens plane. Using this equation, for known lens parameters, we can easily calculate the depth of the scene. The algorithm usually starts by focusing at infinity and moving the focused plane towards the camera. The algorithm naturally requires a number of samples at different focal lengths and therefore may take a certain amount of time to complete - rendering it impractitcal for our purposes.

Depth From Defocus (DFD)

Depth From Defocus is an algorithm that is similar to the above, but which restricts the physical settings of the camera lens to a single value. Albeit a less precise method of obtaining depth information from stationary images, this method is beneficial in some respects in that it only requires a single sample taken with a single known camera setting. An illustration of the setup can be seen on image 1.5. The lens equation remains the same (1.19) as in the previous case.

The lens at distance v' focuses the object A at distance u in front of the lens sharply on the imaging plane (sensor). The green object B at distance u' from the lens plane is out of focus and projects to an area given by the blur circle with a radius r:

$$r = \frac{D\,|uf - v'u + fv'|}{2fu}, \tag{1.20}$$

where f is the lens focal length and D is the lens aperture. If we only take the two opposite points laying on the horizontal line intersecting both the blur circle and the sensor middle into account, we can denote them X_L and X_R. The distance between the points is $|X_R - X_L| = 2r$. For an in focus object $r = 0$. The problem then becomes very similar to the regular local stereo methods, only with a tiny baseline with a maximum size of a sensor diameter. Based on the above equations and measurements of lens properties we can calculate the depth easily.

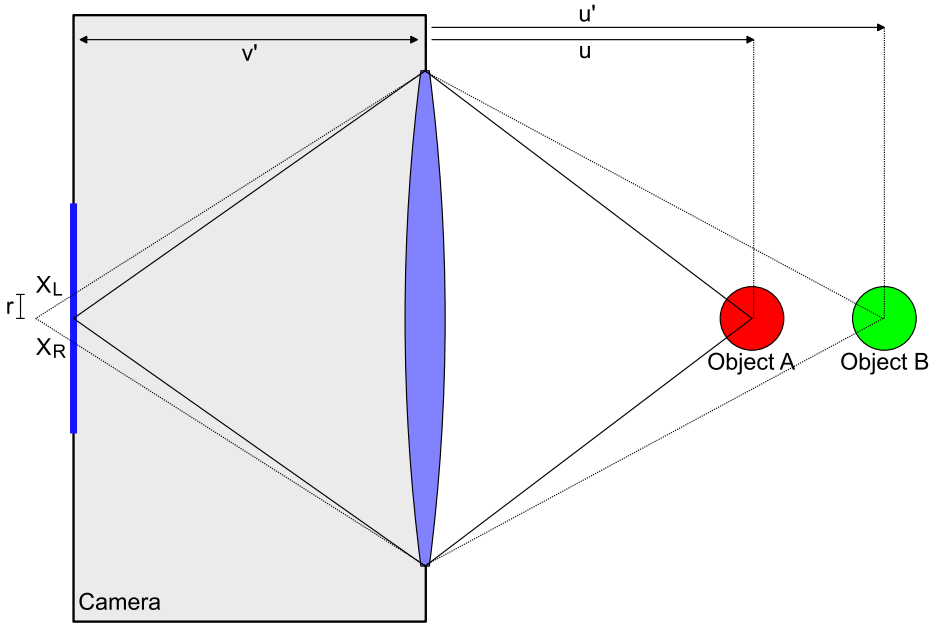


Figure 1.5: Schematic of a defocus usage in disparity calculation. The out of focus object B projects either in front of or behind the sensor plane, which results in a blur of the point over a circle with a radius r.

1.3.2 Light Field

Light Field is an emerging type of camera, or rather camera like systems. They are also known under the name Plenoptic cameras. They allow us to capture the Light Field of a scene - the amount and the direction of light flowing through space. While regular cameras only give us the information about the intensity of the light, Light Field cameras vectorize this information - usually by using an array of microscopic lenses instead of a single large one. While this design generally results in a noticeable reduction in angular resolution due to technological limitations, it allows us to refocus the image after the shot has been already taken. Not only that, the composition may also be altered in post processing slightly, resulting in an effect similar to being able to capture an entirely new shot of the same scene under a slightly different angle.

Formally, the way Plenoptic Cameras - the concept of which has been around for decades, but which have only recently started to enter mass production - do this is extremely complicated and completely out of the scope of this work. The details can be found in [16]. That said we can imagine the micro lens array to be an array of microscopic cameras positioned just above the main camera sensor. Each camera takes a photo of the entire scene projected onto the main camera lens from its coordinates. Naturally, these are

not cameras but lenses. Due to their precise positioning, we can tell exactly what point on the main camera lens the light ray hitting the pixel on the main camera sensor came from. What this means is that the camera takes a number of interleaved shots of the same scene from slightly different positions instead of a single photo. Additionally, the information about the rays allows us to compute what the image would look like if the in focus plane was shifted.

Naturally, what this allows for is the application of the Depth From Focus / Defocus methods on a single shot. In the article [17], the authors propose a method largely based on Defocus and correspondence algorithms, exploiting the advantages of both. The defocus is said to perform better on areas with repeating textures (as can be expected, correspondence based methods often fail under these conditions as there are multiple areas corresponding to each other and it's hard to decide which area is the corresponds to the one we're searching for), while the correspondence methods thrive on object edges and features.

While the entire algorithm is more involved and actually uses a much higher percentage of the views a Light Field camera provides, the authors offer a simplified explanation based on moving a regular camera along a single horizontal baseline - equivalent to shearing the Light Field image along the X axis. The number of views this shearing generates is then processed by a simple DFD and a correspondence algorithm. The shearing changes both the depth of the focus in the scene and the angle of view. An optimal shearing angle α is identified as one that has the highest defocus measure, or the lowest correspondence measure (or both). From this angle, the depth is easily computed.

As the real implementation of the algorithm uses multiple shearing directions instead of a single baseline, an information propagation is necessary alongside the confidence measures to ensure non-ambiguity. To do this a Markov Random Field based method is used to propagate the depth estimation from both correspondence and defocus to a global level, at which a global energy minimization step takes place ensuring smoothness using the second derivative, depth estimation flatness using Laplacian constraint and the confidence weights of the correspondence and Defocus measures.

Using the combination of both, where defocus provides robust information in noisy and repetitive areas and correspondence provides sharp edges to the depth map a consistently good depth estimation can be obtained. The downfall of the method is the sheer cost of a usable industrial level Light Field camera. While consumer level cameras like Lytro can be used for proof of concept solutions, these are entirely unusable in real time scenarios due to their restrictive software environment and closed hardware nature.

1.3.3 Light Coding

Last but not least we'll take a look at Light Coding. This single camera method has been popularized by the Microsoft Kinect line of devices, which uses the concept to control a digital gaming system connected to a TV by feeding it information about the user motion and position in space. The system is not completely irrelevant to scientific research as it is a partially open platform and Kinect has been widely regarded as a go-to Light Coding equipped camera for hobbyists and researchers on budget.

That said, Light Coding is a method that has been proposed years before the first Kinect device. Devices using Light Coding to obtain depth estimation are called Structured Light 3D Scanners. Many variants of Structured Light scanning are possible, but in it's most basic form these devices work by projecting a known infrared light pattern on the scanned surface [18] and scanning the projection from different viewpoints, attempting to identify and match the distorted projections to the original pattern. By comparing these projections to the source image we can compute the shape, distance and other parameters of the observed surface.

The Structured Light methods can be classified into three subcategories: Time Multiplexing, Direct Coding and Spatial Neighborhood. **Time multiplexing** methods are usually the most robust of the three, but require certain amount of stationarity in the view as they rely on projecting a set of subsequent patterns, each taking a brief moment to process, to better gauge the scene. The **Direct Coding** methods project a single dense pattern onto the observed surfaces along with a reference pattern (to measure ambient light intensity) and code each point in the scene with a unique intensity / colour. In theory this allows for a high spatial resolution, but at the same time it requires an enormous spectrum of light intensities to cover the area and is quite susceptible to ambient lighting noise and variations. The third method, **Spatial Neighborhood**, also covers the area with patterns, but the code of each pattern in the scene also depends on the codes of the surrounding neighbors, thus enabling a lower resolution dynamic scene depth mapping. It is prone to decoding errors and therefore usually requires a more robust decoding solution than the other methods.

The depth estimation itself resembles abovementioned local correlation methods where one of the cameras is replaced by a light source. The projected patterns are usually formed in a way that satisfies a multitude of constraints. One of the more important ones is uniqueness with respect to a sliding window. That means that if we form a sparse pattern array using a number of symbols (let's say letters in the alphabet), we want to ensure that there are no repeating blocks of the same size as our sliding window. We then go through the captured image looking for a given window around its expected position and treat the detected displacement as disparity.

Light Coding can present a fast, reliable alternative to Stereo vision, albeit

with many of the same problems including occlusions etc. It cannot be used in all situations as the light source can be a rather limiting factor. We cannot hope to detect a pattern projected on a surface that is too far, the ambient light can confuse the camera, there are problems associated with the choice of coding color (or in the case of monochromatic light source the code) uniqueness etc. That said, of the three abovementioned single camera depth estimation methods it is the cheapest, most promising and most widely used solution, which could be easily adopted for our purpose as well.

Chapter 2

Proposed method

In this chapter we will take an in-depth look at the theory behind the selected methods and the reasons behind their modifications. The Census algorithm, already introduced in section 1.2.7, will be further discussed in section 2.1. In section 2.2 we will discuss the reasoning behind the selected correlation method. The section 2.3.2 discusses methods of left-right consistency verification. 2.3 talks about the confidence calculation and its significance for the algorithm. Section 2.4 covers our edge based modifications to the algorithm and dense disparity information retrieval and section 2.5 covers the depth calculation from disparity data itself. Finally, section 2.6 covers the camera calibration and stero rectification preprocessing steps required by the algorithm design and the various caveats encountered.

2.1 Census transform

The Census transform itself has been discussed previously. We will cover the algorithm in its entirety here except for particular implementation details which have their own chapter [REF].

We start off by obtaining image data. Without loss of generality we will expect a digital color sample in a RGB format with readily accessible pixel intensity values fro all three channels. We will call this image $\mathcal{I}$. The image has dimensions $h \times w$, $h, w \in \mathbb{N}$. The algorithm can be generalized to all three color channels, either by applying it to all three channels separately and then selecting the disparity based on multiple confidences or by generalizing the algorithm to use 3D matrix windows instead of 2D areas. That said, the Census transformation and other parts of the algorithm are complex enough without using specialised hardware even for a grayscale case, which, in any case, gives precise enough results.

To begin, we therefore have to convert the image $\mathcal{I}$ to grayscale, which we will denote simply I. There are numerous ways to do this. If we have a specific usage in mind an analysis of the most important colour channels

should be in order, but the two safest approaches are equal weight channel addition or better yet, because of the nonlinear color representation in the sRGB space, the luminance-preserving colorimetric conversion. According to [19] we can use the following equation for RGB to device independent grayscale conversion:

$$I = 0.2126R + 0.7152G + 0.0722B, \tag{2.1}$$

where R, G, B are the individual color channels of $\mathcal{I}$. We apply the equation per-pixel to the entire image.

After we have obtained said grayscale image, we can proceed with the transform. A Census transformed image can be perceived as a 2D matrix of bit strings. Each string represents the neighborhood of the matrix element it is stored in. We will call this neighborhood a window and denote it W. A window can theoretically be of any shape and size as long as it contains the center pixel and has positive integer dimensions. We can even define a ring region plus the center pixel, but there seems to be no particular benefit to doing this. In order to keep the algorithm as simple and due to storage and speed optimizations, we will define the window W to be a rectangular region of size $h_W \times w_W$, where $h_W < h$ and $w_W < w$ are both odd positive integers.

The transform itself, defined by equation 1.13, is a simple one. It takes every pixel p in the original grayscale image I at position $[p_x, p_y]$ and compares its intensity $f(p)$ to every other pixel $q = [q_x, q_y] \neq p$ and its intensity $f(q)$. Based on the result of the comparison, if $f(q) < f(p)$ it sets the bit A_i of the bit string A to 1, otherwise to 0. The string A is a flattening of the region defined by the window. For example, if we had the following 3×3 region, where b_i are the results of $f(q_i)$ comparisons with $f(p)$, we would flatten it in the following way:

$$W = \begin{bmatrix} b_q0 & b_q1 & b_q2 \\ b_q3 & b_p & b_q4 \\ b_q5 & b_q6 & b_q7 \end{bmatrix} = [q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7]. \tag{2.2}$$

b_p in itself is not interesting as its comparison to itself will always yield 0. We may want to include it in the resulting string however for optimization purposes where memory may not be as much of a concern as speed is.

What's truly interesting about Census is that while it indeed does contain less information about each particular pixel, it in fact allows us to store more information about the region as a whole. More specifically, it allows us to use bigger windows in the same amount of memory. A single 8 bit unsigned integer allows us to store intensity value of a single pixel. Meanwhile the same amount of memory can be used to store an entire 3×3 region information after the transformation. If we allow for 32 bit integers, we get to 5×5 and for 64 we can store 8×8. Realistically though, we will be using areas even larger than that, as that is where Census really shines.

2.2 Correlation and Costs aggregation

Census correlation is fairly straightforward. Given two grayscale Census transformed images I_C1 and I_C2 of the same dimensions we systematically move through the inidividual elements in each row of I_C1 from beginning to the end and calculate the Hamming distance to every element in the same line of I_C2. Supposing a window size of $n \times n$, we can write the algorithm as 4. The array Disparities contains the calculated disparities. The array HDistances contains the Hamming Distances which can later be used to calculate the confidence for each disparity value.

Algorithm 4 Census correlation

1: **function** CensusCorrelation(I_c1, I_c2, Disparities, HDistances)
2: **for** All rows y in I_c1 **do**
3: **for** All columns x_1 in I_c1 **do**
4: $CHD = \infty$
5: $d = 0$
6: **for** All columns x_2 in I_c2 **do**
7: $HD = \sum_{i=0}^{n^2} (I_c1(x_1, y) - I_c2(x_2, y))$
8: **if** $(CHD > HD)$ **then**
9: $CHD = HD$
10: $d = x_2 - x_1$
11: **end if**
12: **end for**
13: HDistances$(x_1, y) = CHD$
14: Disparities$(x_1, y) = d$
15: **end for**
16: **end for**
17: **end function**

This is a very basic and a fairly imprecise approach to Census correlation. The rest of the sections here will be dealing with various improvements and better disparity selection methods. As a side note you may notice that the Hamming Distance calculation here expects the Census transform to include the center pixel. This is intentional, the reasons will be explained in the chapter 3. For the time being suffice it to say that this addition to the calculation is in no way affecting the result.

The term Costs Aggregation is here just for completeness as it is mentioned within most papers written on the subject. With Census, matching without Costs Aggregation has no practical use at all. What the term means is the summation of the Hamming Distances between single bits and using that as a matching cost. For other algorithms, there may be reasons to use single pixel information as correlating two intensities directly without regard for

their surroundings may be of some limited use, at least on a local level. Here we would not only be correlating single bits instead of integers, but the value of those bits would be closely related to some other value within the same window.

One of the optimizations of the above algorithm saving us a lot of computation time is the quite safe assumption, that between two camera views, there is not any point in the image that could, no matter how close or far, move in a different direction than all the other points. Of course, the actual magnitude of the motion differs, hence the parallax effect described on figure 2.1. On this image we can clearly see that the projections of the two objects in different distances from the camera seem to move different distances when the camera position changes. This is the very thing that we're attempting to measure by correlation. While it does present numerous issues like occlusions where the objects may overlap on one image and not on the other one, it can be seen that all the objects move invariably in the same direction.

What this means is that we know for a fact that the object seen on the Camera 1 view cannot be projected further right on Camera 2 sensor. Thanks to this, we can effectively cut our correlation computation time in half and improve accuracy by skipping the search through the regions right of the current position denoted in the algorithm 4 second for loop as x_1.

Another consideration, which however doesn't lend itself for mentioning in relation to the general algorithm implementation is the possibility of limiting the search space when we know the precise disparity limits allowed. When we for example know that we're going to be measuring depths in the range of 2 to 3 meters, we can limit the search space only to those disparities that correspond to these depths and not only speed up the computation significantly, but also likely improve our results by limiting the number of possible errorneous miscorrelations.

2.3 Confidence

Confidence is a measure of how sure we are about a given disparity. At this point the only information readily available to us is the calculated Hamming Distance. The lower the Hamming Distance is, the closer the given searched area found at the disparity is to our source window. It then seems natural to start by defining the confidence as an inverse to the Hamming Distance. For two areas that are very similar, this value is going to be close to 1.

Relying on the Hamming Distance is not only our only choice right now, but it is also the most important factor overall. The abovementioned algorithm 4 implements only a simple Winner-Takes-All approach to correlation. If we see an area that has a lower Hamming Distance than the one we have identified as a candidate previously, we naturally change our decision and update the

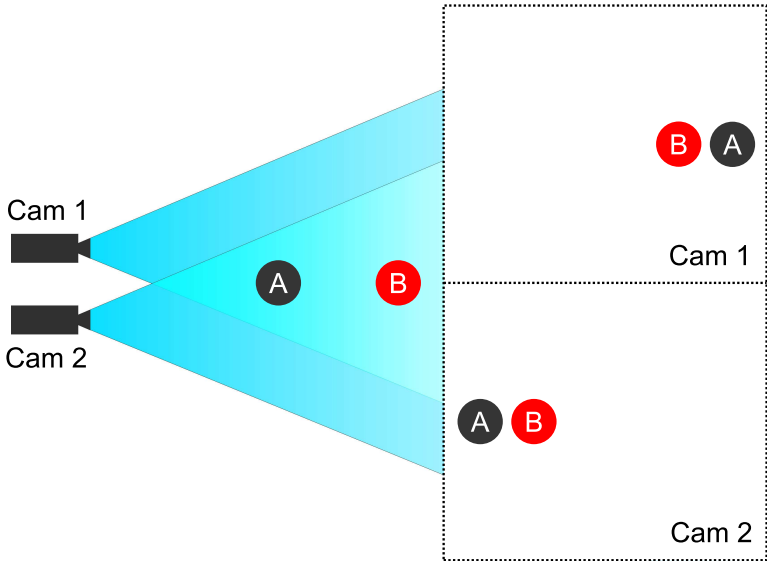


Figure 2.1: Parallax - Birdseye schematic on the left and camera views on the right

values accordingly. Another approach, a more beneficial one, is to keep track of a few possible disparities at a time.

2.3.1 Hamming Distance as confidence

If we take a look at any two images taken from a similar viewpoint and pick a random point in one of them, there is very likely a number of areas in the second one that look similar to it. Since Census removes the absolute intensity information, there are even more of them. Flat homogenous areas and repeating patterns are especially problematic. Because of all this, it is a good practice not to rely on the single best solution we have identified as it may only be a local optimum. To remedy this, we keep track of every solution which satisfies the condition $|\mathrm{HD}_{\mathrm{Minimum}} - \mathrm{HD}_{\mathrm{Candidate}}| \leq 1$, where $\mathrm{HD}_{\mathrm{Best}}$ is the best Hamming Distance we have identified up until this point and $\mathrm{HD}_{\mathrm{Candidate}}$ is the one we're considering. This means that deviations of 1 are acceptable and considered as a solution. If we on the other hand find a solution that is significantly better than the current one as far as Hamming Distance goes, we forget about the previous ones and start fresh with that one as the current best.

There are multiple steps to selecting the most likely disparity. To begin with we want to make sure areas with continuous disparity regions are rewarded. To do that, we want to go through all the disparities for all pixels, if there's a pixel within the 8-connected neighborhood of the current one with the same disparity as the one we're currently looking at, we will raise the con-

fidence of the given disparity by 0.5 for each one. This way we will cover the larger continuous areas. While some outliers may get caught in the process at the edges, the benefits greatly outweigh the losses.

2.3.2 Left-Right consistency checks

Left-Right consistency checking is a method that allows us to verify and enhance our results. In an ideal situation, since we searched the right image for a match to a part of a left image, we should also have the best match for a part of the right image in the left. This is not always the case. Due to noise, differences in perspective etc. it is very much possible that there is a better match for the part of the right image we just identified as a solution to our so called Left-Right (LR) correlation problem.

Another problem is that relying solely on disparities from a single sided run results in a gradient tendency towards one side of the image. When we start searching the right image from the left side to decide on the leftmost disparities in the left image, we tend to get larger outliers as whatever the current x_2 value is, the x_1 is close to zero. Therefore if we don't identify the correct x_2, the equation for disparity $d = x_2 - x_1$ in algorithm 4 gets close to $d = x_2$. At the same time the opposite is true - for large x_1, the equation may become $d = 0$ as $x_2 \approx x_1$.

To remedy this, we have to start working with the confidence again. First off, we have to run the algorithm in it's entirety for both the left and the right image. After we finish all of the previous steps, we should have two separate 2D arrays with vectors of disparities and their associated confidences. These were already refined as to their continuity within their 8 connected neighborhoods each.

The Left-Right consistency checking lies in comparing the values stored in both the disparity arrays. By going through all the elements of the left disparity array and looking at the elements of the right one at the location this disparity links to, we can search through the disparity vector stored here and reward such disparities, that link back to our starting position. This process is then repeated looking from the right array into the left.

This way, after were done, we should have more than enough candidates for our selection process with high enough confidences to get through the confidence thresholding that follows. The strongest candidates are going to be those that have their corresponding counterparts in both arrays and those that lie within a strongly connected patch of disparities that link to the same area.

LR consistency checking, when ran on regular implementations of various correlation algorithms, also helps with occlusion identification. Occlusions are areas that can be seen on one image and are hidden on the other. For illustrations of two common causes, see figures 2.2 and 2.3. Correlation on such areas will naturally fail completely. Thanks to LR consistency checking,

we can identify such occluded areas and then process them accordingly, as there will be no matching disparities in the other correlation direction.

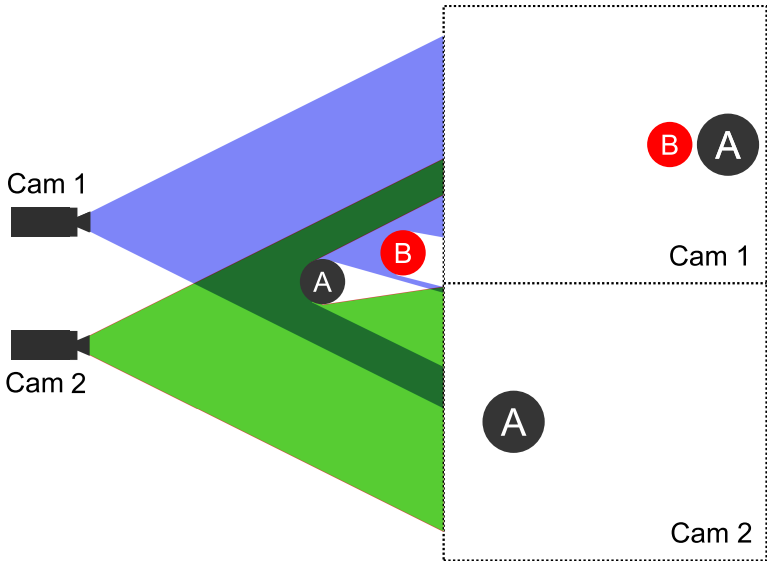


Figure 2.2: Illustration of an occlusion problem - both objects are visible to the first camera, but the object A appears in front of object B for the camera 2 and occludes the view.

2.4 Edge based approach

Census transformation and correlation, while being highly parallelizable and quite simple in its approach is still a computationally complex algorithm. We have to transform $2 \cdot width \cdot height$ pixels, each of them into n bits and then perform numerous comparisons on said bits. To remedy this, we devised an approach based on a system a lot of low powered embedded stereo systems use. The article [20] deals with hardware design considerations when working with edge based stereo, but the ideas are clearly applicable to software based implementations as well.

Restricting the correspondence algorithm only to the edges in the image greatly accelerates the computation by a significant reduction of the search space. Instead of transforming and searching through the entire image, we now only have to go through a much smaller percentage of pixels that actually contain some sort of significant information and are thus easily identifiable by the Census correlation. Naturally, this approach will result in a small error as the different camera positions can and do result in different edges, but overall this doesn't pose as much of a problem when looking at an object from a certain distance as the most significant edges are usually captured reliably.

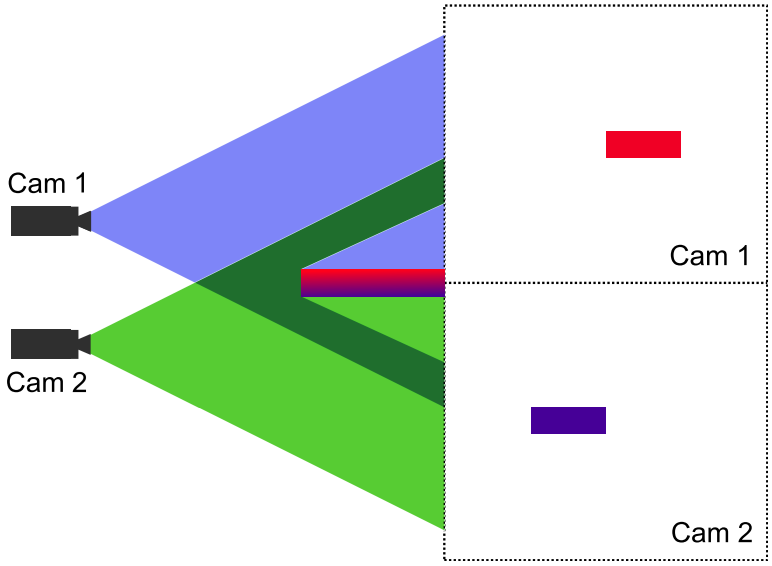


Figure 2.3: Illustration of an occlusion problem - another type of occlusion. The cameras see different sides of the same narrow object and due to different side colours cannot recognize it is the same object.

It would of course be a mistake to depend solely on edges as there may well be a number of smoothly transitioning curved surfaces that the system would perceive as flat. We have to instead take advantage of any easily available information that we have at our disposal. To do that, we shouldn't rely solely on strong object edges, but on any easily recognizable pattern or texture. Textures can also be very easily detected on both images of the stereo pair, so they're an even better target for correlation than object edges, which can change their appearance drastically from the standpoint of Census.

2.4.1 Finding the edges

There are numerous ways of finding edges, the most popular such algorithm is the Canny edge detector described in [21]. The problem with this detector lies in it recognizing the difference between so called strong edges and weak edges and filtering the weak ones out. For our problem we need to dig deeper and use only the Sobel operator that Canny detector stands on, the description of which can be found in [22].

Sobel operator gives us the magnitudes of the image gradient. Not only does it give us more edges and interesting features to work with than Canny detector, but it is also much simpler in its implementation as it can be written

as an addition of two convolutions with the following kernels:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{2.3}$$

The best thing about the Sobel operator is that it is not only much simpler than Canny, but that the kernels S_x and S_y are separable. That means they can be written as a multiplication product of two one dimensional kernels each. Specifically:

$$S_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, S_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}. \tag{2.4}$$

What this means is that the algorithm can be parallelized extremely effectively on modern GPUs if needed as described in [23].

First the image is ran through an edge preserving median filter in order to remove noise. The Sobel operator is then ran on all 3 color image channels treated as separate grayscale images in order to obtain as much information as possible. The channels then converted to absolute values, both x and y directional derivatives are added and they are separately thresholded in order to remove insignificant noise. The result can be seen on figure 2.4. It is then converted to a binary image in order to create an edge map and finally dilated with a rectangular structuring element in order to grow the search space beyond the points closest to the edges, to obtain reliable depth information near discontinuities and to close up regions, which will later help with segmentation.

The algorithm so far be summarized as follows:

1. Undistort both the left and right color images $\mathcal{I}_1$ and $\mathcal{I}_2$
2. Find the edge maps E_1 and E_2 for both $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively
3. Convert $\mathcal{I}_1$ and $\mathcal{I}_2$ color images to grayscale I_1 and I_2
4. Census transform the I_1 and I_2 windows W_1 and W_2 where center pixels are white in E_1 and E_2
5. Perform Hamming Distance based LR / RL correlation to identify candidate disparities
6. Perform confidence refinement by
   a) Performing the 8 connected neighborhood test
   b) Performing the LR and RL consistency checks
7. Select the most likely disparities based on a Winner Takes All algorithm

(a) Source image

(b) BGR edge map

(c) Thresholded binary edges

(d) Dilated edge map

Figure 2.4: Edge identification.

8. Threshold the confidences and forget a percentage of the most uncertain disparities

2.4.2 Segmentation

This gives us a good amount of quality data around the edges. To obtain dense disparity data in the central areas we have to interpolate. To make that easier, we want to split the image into independent sections with discontinuities at the edges but otherwise assumed continuous. Since we already have the edge maps, we will use those as a basis for a segmentation algorithm.

To perform the segmentation, the edge maps are searched for contours and those that are too small are skipped to filter out details and noise which would interfere with the distance transform. The contours are then drawn into a separate image, distance transformed and thresholded to obtain seeds in center areas. The areas in the middle can be assumed continuous, the edge map therefore gives a good approximation of segmentation borders. The thresholding splits the distance transform into a number of separate areas in cases the edges were not properly detected.

The thresholded distance transformed seeds are then used as a basis for watershed segmentation. Watershed transformation, described in [24] as an image segmentation algorithm, grows the individual segments from the seeds using the image gradients as borders to stop the flow. We can imagine it working as if a source of dripping water was placed in each seed position and the image gradient was a landscape. The low magnitude areas would flood first up until the high magnitude gradient areas are reached.

The watershed transformation produces an image of markers, in which each area is assigned a different colour and can therefore be uniquely identified. For purposes of further processing the areas are again searched for individual contours, which can be easily scaled and iterated over in the next step.

The individual steps and the result of watershed transformation can be seen on figure 2.5. It is generally better to oversegment the image and then merge the adjacent regions for which we have a multitude of tools at our disposal. We can measure the differences in their mean intensities, we can transform the image into some lighting invariant colorspace as in [25] or some of the more common choices such as Normalized RGB, Logarithmic HSV and measure the region similarities between those. We can measure the region variances, mutual information, histograms and entropies. We can also use the information about their dimensions, e.g. when there is a segment with a large bounding box but a small area, it is more likely to be a part of a larger area near it such as text on a homogenous background.

2.4.3 Interpolation

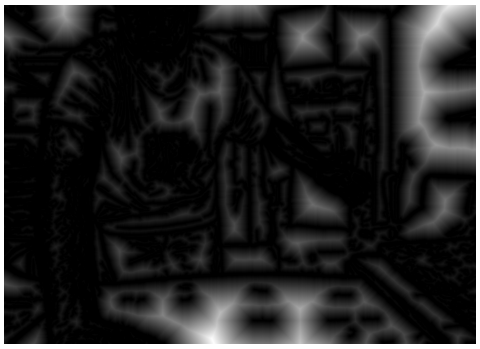
The interpolation is a last step we need to perform in order to obtain a dense disparity map. It is a process of inpainting non-measured data computed from its surroundings. A functional equivalent would be curve fitting. In order to interpolate between measured data points, we need to make two assumptions:

1. A segment represents a continuous surface with a zero net Gaussian curvature, that can either face us at an orthogonal angle or it can be rotated in any dimension. A zero net Gaussian curvature is a curvature of any surface that can be obtained by bending a plane as derived by Gauss in [26, p. 102]. Such a surface has a characteristic equation:

$$\left(\frac{\partial^2 z}{\partial x \partial y}\right)^2 - \left(\frac{\partial^2 z}{\partial x^2}\right)\left(\frac{\partial^2 z}{\partial y^2}\right) = 0, \tag{2.5}$$

   where z is a function of x and y. What this means is that in every point of a surface developable from a plane there is at least one direction of motion with a zero derivative and that all points in this direction form a line over the entire length of a surface, hence such a surface can be imagined as a set of loosely tied lines. Given such an assumption, or even

(a) Inverted edge map



(b) Distance transformed edge map

(c) Result of watershed segmentation

Figure 2.5: Watershed segmentation steps.

better assuming completely flat surfaces (which is not far from reality when we take a look at the typical design of a vehicle undercarriage), we are able to interpolate the region disparity.

2. Two adjacent segments can have a disparity discontinuity in between them. This means that the two segments might - but don't necessarily have to - represent two completely different objects at different depths and therefore the measurements from one of them should not affect the other one.

While it is true that the above two conditions may not be entirely satisfied in all cases - like a parabolic surface, they provide a good approximation for our needs and can be leveraged to form a simple and easily parallelizable inverse distance weighting interpolation algorithm.

The algorithm starts by obtaining the segmented contours from the above step. The contours are drawn into separate binary maps which are then eroded very slightly to obtain similar shaped regions which are guaranteed to lie within the boundaries of the segment. These shapes are searched for a second set of contours which is used for the actual interpolation. These new contours are iterated over point by point and the surrounding pixels of each vertex is searched for candidate measurements. Multiple measurements near each vertex (if they exist) are taken into account by computing the mean disparity over them, so that a possible outlier measurement doesn't affect our result too much.

For each point within a segment which doesn't have a measurement assigned to it, we can write the equation for inverse distance weighting proposed in [27]:

$$d(x_0, y_0) = \frac{\sum\limits_{v \in C} \frac{f(v)}{(v-(x_0,y_0))^2}}{\sum\limits_{v \in C} \frac{1}{(v-(x_0,y_0))^2}}, \tag{2.6}$$

where the coordinates $[x_0, y_0]$ represent a given point, v is a vertex lying on a contour C and $f(v)$ is the mean disparity value assigned to it. This interpolation method provides smooth regions with discontinuities at the edges. To get rid of the remaining disparity outliers, we can perform an optional edge preserving median blur step at this point which is however omitted in the application.

2.5 Triangulation

Triangulation is the last step of stereo vision where we obtain actual depth information from the calculated disparities. The book [28] covers the entire process in great detail for every setup imaginable. In cases where we have a

set of two cameras with two identical calibrated lenses with a camera focal length f, mounted on a baseline of length b, with coplanar image planes and parallel optical axes, we find ourselves a situation captured in detail on 2.6.

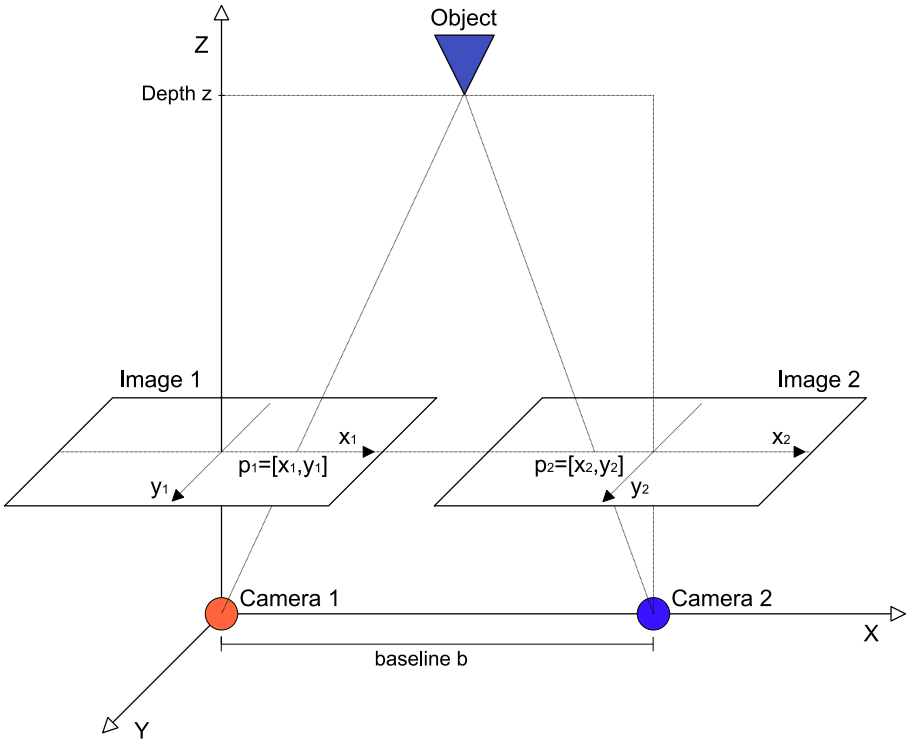


Figure 2.6: Depth from disparity schematic

There are a number of relations when viewing a scene from two different positions, arguably the simplest and most computationally efficient case is the one we use. The epipolar geometry of our setup, derived from a pinhole camera model at [29], results in disparities only in the x direction. This greatly limits the size of our state space search and results in linear relations between the disparities and the actual spatial positions.

As we can summarize from the image, the stereo pair is rectified and $y_1 = y_2$, therefore the disparity is calculated as $d = x_1 - x_2$ the spatial point which we're triangulating $p = [x_p, y_p, z_p]$ is at position given by $x_p = \frac{bx_1}{d}$, $y_p = \frac{by_1}{d}$, $z_p = f\frac{b}{d}$.

2.6 Calibration

There are two steps omitted from the previous description of the algorithm which can however be separated from the rest of the system and perceived as unavoidable preprocessing. These are the lens calibration and stereo rec-

tification. Both these steps have to be performed once and result in a set of transformation matrices that are applied to each subsequently captured frame and which make our correlation work easier.

2.6.1 Lens calibration

In order for our epipolar assumptions to hold true, we require our system of cameras to view the scene through exactly the same lenses with the same parameters. Any lens, no matter how well manufactured, introduces some degree of distortion. What this means is that the scene viewed through the camera lens can and indeed does look differently when inspected through different part of the image sensor. It is easier to discuss the various types of distortions and how each of them affects our correlation problem separately and later summarize how we can deal with the most problematic ones. The list below is by no means comprehensive, lens design and the associated problems form an entire field of study and are out of the scope of this work. A more comprehensive review of all the different problems can be found in [30]

To begin with we'll take a look at **vignetting** described in [31]. Vignetting is a term used to describe changes in brightness or saturation along the edges of the image. It has many causes. It can be caused by mechanical obstructions in the lens design or by additional filters. If we use a lens with smaller diameter than our sensor size, we will observe a total occlusion of a circular part around the image corners as can be seen on figure 2.7.

Another cause of vignetting is the design of the camera CMOS sensor itself. As the light hits the sensor at different angles in different parts of the sensor, more photons excite the pixels in the center which are hit at angles close to orthogonal as opposed to the pixels at the edges, which are hit at oblique angles. Most camera manufacturers spend significant resources measuring and fighting this problem in internal processing of the raw camera data upon conversion to more common image formats, but imperfections can be observed, especially on low end cameras and when working with RAW formats.

As far as vignetting is concerned, we don't have much to worry about, Census by its nature isn't much affected by absolute pixel intensity. We do face the problem of it affecting the transformation itself when the change is abrupt, as then one side of the transformation window has lower relative intensity than the other, in which case we can either use a different lens or multiply the affected image pixels by a radial gradient gain to remedy the problem at least partially. If we face a FOV restriction, which is very much possible with the M12 mount fisheye lenses intended for the end application, we will have to either crop the useful image portion (due to computational efficiency a much more likely scenario) or define a circular mask to filter out the black pixels which contain no information for our use. Vignetting can also be, in some cases, reduced by using a smaller aperture.

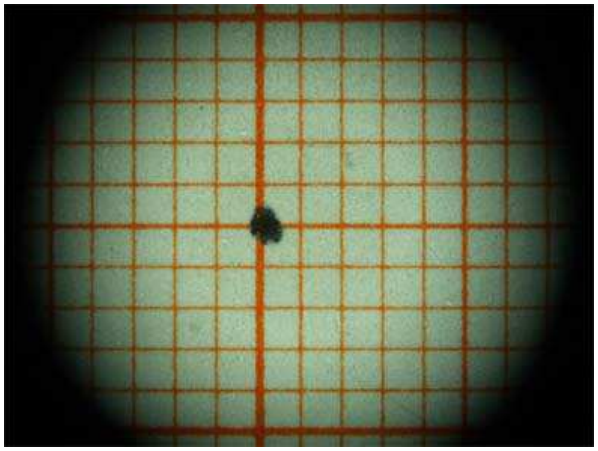
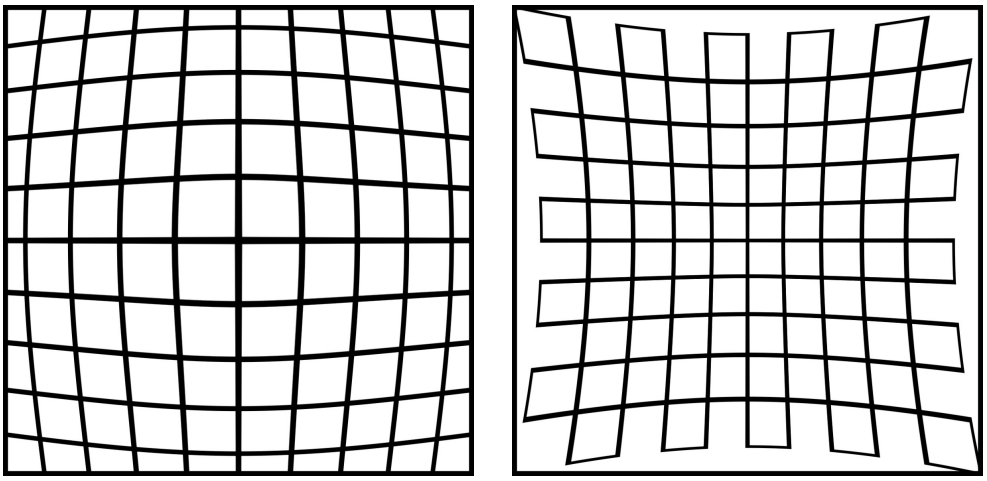


Figure 2.7: Example of vignetting and field-of-view restriction on a microscope lens [1]

Another often faced problem is **chromatic aberration** (CA). CA occurs because the lens refracts different light wavelengths at different angles (in much the same fashion as a prism does). It is observed as a gradual shift between the individual color channels that is more pronounced at the edges of the image, resulting in a blurry composite image. The shift is usually only a couple of pixels wide, but presents a challenge nonetheless as an object can be located in the center of focus for one of the cameras and at the edge for the other, in which case the correlation may fail even when using a grayscale input image as the grayscale image is just a weighted addition of the different colour channels.

Some very interesting research went into reducing the effects of Chromatic Aberration, namely [32, 33]. The processing is however fairly computationally intensive and out of the scope of this work. Some research [34] suggests, that chromatic aberrations can interestingly cause both an increase and a decrease in stereo correlation matches. The issue can be partially reduced by using a larger matching window, which is already our intention with Census, therefore we won't concern ourselves with the issue much other than considering better quality lenses for the end use.

One problem which we do have to concern ourselves is the overall softness in the image introduced by **lens resolution**. Most C and CS mount lenses are built to produce quality image for VGA to 720p resolutions usually used for closed circuit surveillance. The costs of a high end CS mount lens with the required parameters sits in the same range as the cameras themselves. Lens resolution is usually specified in terms of lines per mm which it can produce in the center and at the edge of a frame. In an ideal case, this number would match or exceed the number of lines in the corresponding part of the imaging sensor itself. The number of lines however is often matched in the center,



(a) Barrel distortion (b) Pincushion distortion

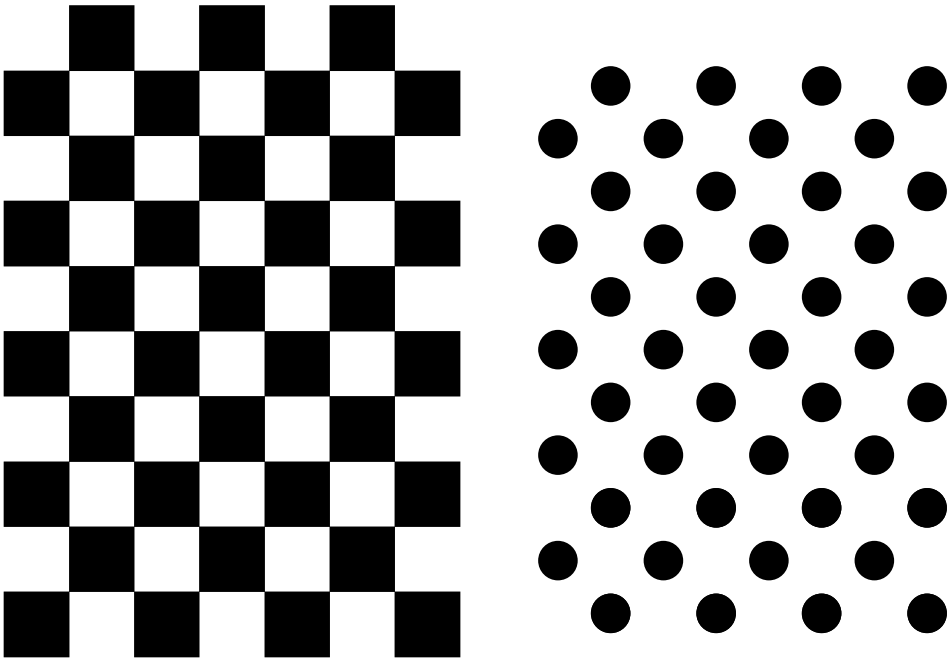
Figure 2.8: Overview of lens distortions

where the lens surface is close to flat and the optical path of the light is the simplest, but not at the image edges as it quickly degrades with lens curvature.

More details on the issue can be found in [30]. From our standpoint, we are especially interested in the softness around the object edges and in textures, which we need to detect and which we use for the correlation as described above. Experiments have however shown that any attempts at sharpening introduced more problems with chromatic aberrations, the amount of noise in the image and have not increased our matching rate at all. We try our best not to introduce anymore softness around the edges by using edge preserving blurring methods like the median filter in edge identification steps. Since we're intending on using only a very limited part of the image sensor in the vertical middle, where the situation is less pronounced than at the top and the bottom, we don't concern ourselves with the issue outside of selecting a good quality lens and considering a larger imaging sensor camera.

This brings us to the last part, which is lens **distortion** in the general sense of the word. Even if we have a sharp image without chromatic aberrations or vignetting, every lens produces a certain amount of curvature in parts of the image. There are different types of distortion, generally there's the Barrel distortion and the Pincushion distortion captured on figure 2.8. A combination of both is possible and is called a Handlebar distortion, from the shape of a Handlebar moustache which curves down and then back up.

Both of these distortions are fairly straightforward to correct. A lens calibration board of known properties like the dimensions and number of corners such as the ones on figure 2.9 are used for measuring. The standard checkerboard pattern proved itself to be unusable in a number of situations as most lenses introduced too much softness around the corners on small imaging



(a) Checkerboard grid (b) Asymmetrical circular grid

Figure 2.9: Lens calibration boards

sensors and even when thresholding was applied, the corners had a tendency to get disconnected. The asymmetrical grid shown on figure 2.9b proved to be a more robust candidate as entire circles need to be detected, which is easier as their areas are large and a number of algorithms including Hough transform [35] can be used to detect them and compute their centers.

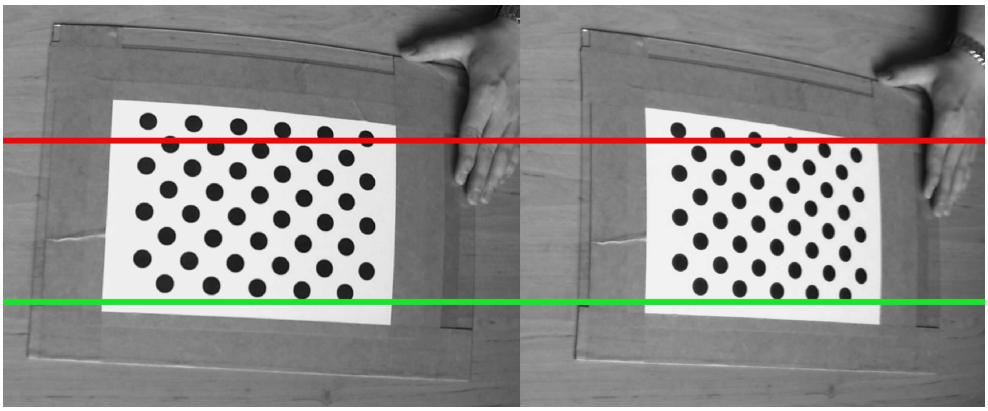


Figure 2.10: Example of lens distortion effects on the epipolar lines

The effects of lens distortion on the epipolar lines in a noncalibrated and

nonrectified stereo pair are captured on 2.10. The composite is a cutout of the upper right quarters of two images used for calibration purposes. The green line is close to the image vertical center, therefore the lens distortion effects are not as pronounced here, but the red line clearly shows a distinct curvature in the view. This problem would in fact be corrected by stereo rectification, which is however a very uncertain and computationally intensive process even when the images are undistorted and should be corrected beforehand.

A method proposed in [36] is used for camera calibration purposes. The method identifies the camera intrinsic and extrinsic parameters. The relation between a 3D spatial point **M** and the image projection **m** in a pinhole camera model can be written as:

$$s\mathbf{m} = \mathbf{A}\,[\mathbf{R}\ \mathbf{t}]\,\mathbf{M}, \tag{2.7}$$

where s is a scale factor, $[\mathbf{R}\ \mathbf{t}]$ are the camera extrinsic parameters containing information about the rotation and the translation relating the world coordinates to the camera, **A** is the camera intrinsic matrix given by:

$$\mathbf{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.8}$$

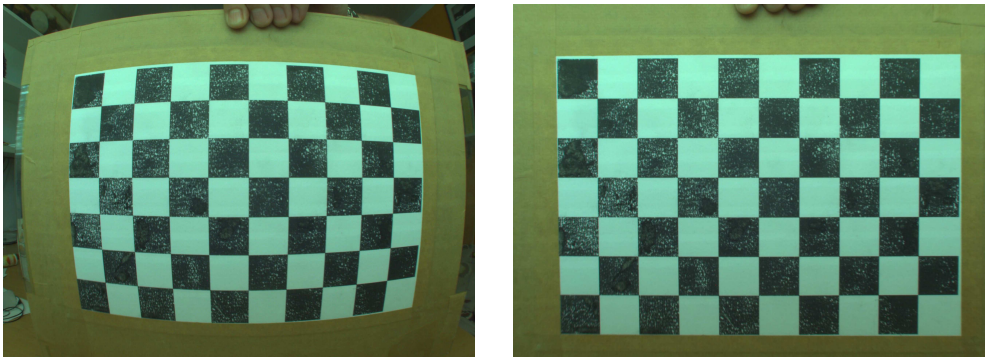
where $[u_0, v_0]$ are the principial point coordinates, α and β are the scale fators in the image axes u and v and γ is a parameter describing the skew between the two image axes.

Using the above equations, we can express the relations between the objects in view and the image itself and correct size, skew and rotation problems for a pinhole camera model. Removing the distortions for a real lens is a more involved process building on a set of nonlinear equations based on the above and a statistical maximum likelihood criterion method and is out of the scope of this work. The reader is kindly referred to the freely available paper by Zhengyou Zhang referenced above. The effects of the calibration algorithm on an image of a flat checkerboard pattern can be seen on 2.11.

Note that removing the distortion on a fisheye lens is an even more involved process as lenses with FOV above 90° introduce more complex distortions than Barrel and Pincushion.

2.6.2 Stereo rectification

Stereo Rectification [37] is a process of stereo camera calibration, in many ways similar to the previous step - the lens calibration. In this case we're attempting to project the captured frames in such a way that we obtain a pair of images where every point in image A can be found on the same y coordinate in image B. The process doesn't and really cannot say anything about similarity in the x direction, which is why we have to perform a search for disparities in the first place.



(a) Camera output without calibration (b) Camera output with calibration

Figure 2.11: Lens calibration effects

While the lens calibration step should theoretically remove the need for stereo rectification, there are many issues which it doesn't account for. Modern cameras often have very slight differences in sensor placement on the board between individual units, the camera lenses might not be set up in exactly the same way and the lens threading may introduce very slight shifts between the images as well. All of these problems, combined with the fact that setting up the cameras to be precisely parallel to each other often proves to be very tricky as well, result in a need for software remapping of one or both images in order for the epipolar lines to be perfectly horizontal.

The process of Stereo Rectification relies on similar techniques as the above. Using an iterative algorithm we're trying to find a set of projections that would minimize the reprojection error between a set of the corners of a calibration pattern captured by two different calibrated cameras. The implementation itself relies on internal OpenCV functions and was not modified, our only modification to the algorithm lies in moving from a calibration checkerboard pattern to an asymmetrical circle grid that proved itself more efficient a tool for lens calibration as well. The reasoning behind this is the same as above, instead of having to deal with a large print of a checkerboard as the corners would be very hard to identify on a small one for all practical distances, we try to find the centers of sets of rather large and easy to identify circles, which is much more robust a method.

Chapter 3

Implementation

In this chapter we will concern ourselves with the details of the implementation of various parts of the system. We will also discuss the used hardware, software and the reasons for their selection. The solution proposed is more of a proof of concept rather than a working deployable system. Some optimizations were implemented in order to get insight into the scalability of such a system, other possible optimization directions will be discussed here based on our previous experience with similar problems, but were not yet implemented.

3.1 Equipment and software

The equipment used for testing the algorithm consisted of a **set of Elphel 353L cameras** equipped with a 1/2.5" MT9P001 sensor. The cameras come with a CS mount and a C mount ring adaptor. They were equipped with a **pair of Kowa C mount LM4NCL lenses** which allowed us to capture an area with a Field of View of approximately 79° each. Both lenses have been measured thoroughly using a 4×5 assymetric circular grid in order to remove lens distortions and then calibrated as a stereo pair to obtain sets of images with matching lines.

The Elphel cameras were selected for their previous performance in a similar project. They are unique in their open hardware design offering a reprogrammable FPGA, access to all internal image processing capabilities, running a low impact distribution of Linux. They offer RS232 Fast Ethernet connection - which is sadly a limiting factor when it comes to obtaining high framerate data, but enough for our application. The cameras are synchronised using an RJ11 telephone trigger cable in a Master - Slave configuration as seen on 3.1.

The cameras running in full 5 MPx resolution of 2592 x 1944 offer a limiting framerate of 10.6 frames per second. The cameras can however be reprogrammed to produce a multitude of resolutions using binning and Region of Interest combinations. A 2592 x 96 px, a near linescan mode, is assumed for the final application. At this resolution the cameras are able to produce

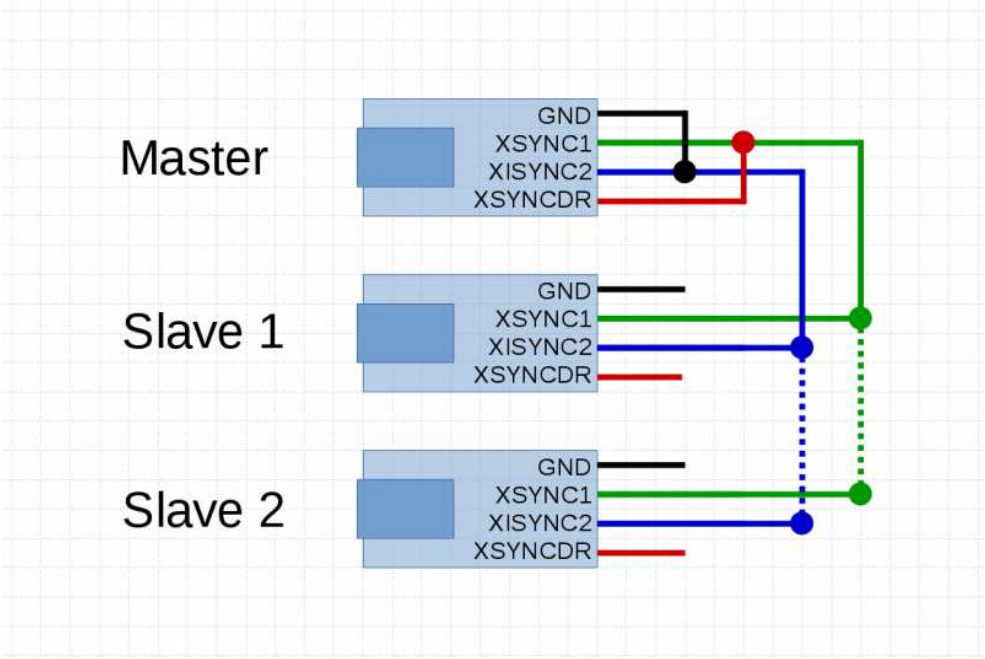


Figure 3.1: Trigger wiring schematic - courtesy of Elphel Inc.

approximately 300 FPS. The cameras, after multitude of tests to decide the best inter-axial baseline distance, were mounted 10 cm apart on an aluminium U profile.

The cameras are connected to a gigabit ethernet router, and the data is streamed into a **Macbook Air 1,8 GHz Intel Core i5 13"** laptop equipped with 8 GB 1600 MHz DDR3 memory and Intel HD Graphics 4000 1536 MB graphics, running OS X 10.11.4 (15E65) using a hardwired connection.

The software used consists of Octave, an open-source Matlab replacement for quick testing and **OpenCV 3.0**, released during the course of development of the applications related to this thesis, for end use implementations. OpenCV internally uses FFMPEG for image data streaming. A GStreamer based implementation was considered but dropped after previous experience with problematic high framerate and high resolution processing.

All the end use applications were written in **C++**, initially built with GCC 5.0, a GNU distribution unusual for OS X, obtained through Macports for its **OpenMP** parallelization support and better debugging tools compatibility as compared to Clang. Due to a number of problems with linking OpenCV 3.0 to GCC and with debugging in an OS X setting, an approach using **Clang/llvm 3.9**, which is as of yet not distributed with OS X developer tools and had to be built from source has been devised in the end. This version was selected due to its support for OpenMP. **CMake** was used to control the compilation process due to its close interconnectedness with OpenCV. **Libcurl** functionality was

used in a limited scope to set up the cameras for streaming and query their parameters over HTTP.

3.2 Implementation details

The applications were implemented using OpenCV 3.0, an open source computer vision library written in C and C++. OpenCV is a mature and well tested library that offers a multitude of implemented algorithms useful for our application. It was chosen over others for its speed, documentation and thought out design, which allows for a rather quick development once everything is set up. While some problems were encountered during the course of the development, they were mostly caused by minor bugs that were introduced with a major version release and did very little to hamper the overall progress.

No GUI functionality was implemented outside of displaying the results using OpenCV provided methods to preserve modularity of the applications. After previous experience and with considerations regarding the close interconnectedness with OpenCV, Qt would be the likely framework of choice when it comes to GUI creation and would likely replace some of the other libraries used like libcurl as well.

The software comes as a proof of concept bundle of applications. First application called **live_view** sets up the cameras on the specified IP addresses according to supplied parameters, sets up the synchronization and displays a live feed with a corrected lens distortion where camera calibration data is provided. It then queries the cameras for new frames over HTTP using a buffered image server on the cameras periodically. This is by no means the fastest way of obtaining data from Elphel cameras, but it is the most reliable and straightforward one. A system where the cameras mounted a ramdisk on the target system using NFS was previously successfuly employed, which allowed us to transfer a 2592 x 96 px MJPEG stream over Fast Ethernet with framerates of over 300 FPS for limited periods of time until the circular buffer on the cameras overflowed. A similar solution is intended for our end use if it is required. The application then allows the user to save selected frames by pressing the spacebar when either of the live feed windows is active.

Two other noteworthy applications heavily based on the samples provided by the OpenCV library are the **lens_calibration** and **stereo_calibration**. As their names suggest, they allow the user to calibrate the cameras using sets of images obtained using the **live_view** application. The **lens_calibration** expects unrectified and uncalibrated data as an input and produces an XML output file containing the intrinsic and extrinsic camera parameters, the reprojection errors, the distortion coefficients and optionally a list of detected calibration board corners and their positions in the image. The **stereo_calibration** application expects an input of calibrated unrectified samples where both cam-

eras have an unoccluded view of the calibration pattern and produces a YAML encoded set of projection matrices used to stereo rectify the data from the respective cameras when successful.

3.3 Correspondence algorithm

This brings us to the core **block_census** application which searches the pairs of calibrated and stereo rectified samples for disparities and which will be discussed in-depth. The application is intended as a proof of concept and not a mature solution. It allows the user to input a pair of samples in an OpenCV supported format (likely JPEG) along with the intended window size and a scale factor and outputs the resulting measured correlation result and an interpolated result both with an applied colormap and raw data in a bitmap. The triangulation part of the algorithm is trivial and is not implemented at this point as the disparities themselves are more telling and the triangulation would greatly benefit from additional measurements in the end setting, which is infeasible for purposes of development.

The application begins by loading the two samples passed to it as arguments. The first sample is expected to be the right one and the second is the left one. Unless a -nocorrection argument is used, the calibration matrices used for stereo rectification are loaded from the path "./calibraton/intrinsics.yml" and "./calibration/extrinsics.yml" respectively. A previous application of lens undistortion on the samples is assumed (and is optionally done using the live_view application in our case). The samples are then remapped in order to obtain a stereoscopically rectified pair.

If the previous initialization was succesfull, the correspondence algorithm begins. It can be separated into four distinct parts as far as the data flow and processing is concerned. Before running them we have to process and prepare the inputs. To continue, we need an edge images - which are obtained from the full sized colour input images using a Sobel operator, converted to absolute scale and thresholded to obtain a binary representation and then downsampled when required to match the correlation input size (correlation is performed on downsampled frames). We also need grayscale versions of both the left and right input images, downsampled as well. Arguably we could have saved some computation time by obtaining the edge maps from downscaled samples right away, but the difference is rather negligible and the extra information comes in handy during Sobel operator application.

All these steps are highly parallelizable on CUDA if need occurs. As mentioned before, Sobel is a 2 dimensional convolution operator with separable kernels, which is applied to each image channel separately. This is an ideal solution for CUDA and we can observe speedups in the range of hundreds as opposed to CPUs. Merging the 3 channels into a grayscale image is a simple addition operation on completely independent data points, the same applies

to absolute rescaling, thresholding and downsampling. If we can feed enough data to the GPU, we can expect enormous speedups on the preprocessing section. As it stands the steps are partially parallelized using OpenMP and internal OpenCV routines.

3.3.1 Census Transform

The first part of the correspondence algorithm itself is captured on figure 3.2. The entire section can be split into a left and right part for the respective input images. The inputs are the downsampled left and right edge images and downsampled grayscale rectified input images. The first two parts of the section dilate the edges slightly in order to obtain certain amount of information beyond the bare minimum. We need to clearly separate the different segments later on and to do that we need data measurements in parts of the image which do not hold any significant information (smooth surfaces) which aren't captured on the edge maps. The correlation window is larger than most edges anyway and we can therefore expect good results when applied to points near to them. For faster processing the images are then converted into left and right Boolean value vector masks as the OpenCV binary image representation is excessively bulky and unoptimized for our purpose.

At this point we can iterate through the image rows and transform the individual windows around each pixel into a Census defined bitset. For ease of use the windows are implemented in a Struct holding their x and y coordinates and the midpoint intensity for reference. For our purposes we use a bitset size of 512 bits, which allows use to use windows of sizes up to 22×23. Our application uses square correlation windows as there was no real benefit observed when using general rectangles, but it was tested using rectangular and single dimensional windows as well. Only pixels which have the corresponding value in the Boolean mask set to true are transformed for both the left and the right image. The transformation is parallelized row wise according to the input image and the previous edge mask preparation sections are parallelized for the left and right part separately.

Once again this part of the image is highly parallelizable as there are no cases where the algorithm has to write into the same memory cell and there are no race conditions or similar problems. There is arguably some degree of thread divergence when the input images are masked by the edges, but this shouldn't pose much of a problem for our end application as the data we expect is taken from the horizontal middle part of the scene and should contain a number of near equidistantly spaced edges to use the GPU effectively. If problems occur (or rather if we don't observe and slow downs doing so) with this approach, we can transform the entire image and then filter our the results we don't need. Some Census implementations on CUDA include [38], which implements the entire dense correlation algorithm with promising results of 22.2 FPS for 800×600 images and a Census window of 5×5 implemented

as a sparse convolution kernel which skips each other row and every other column, with results similar to a 10×10 kernel. The authors of [39] comment on various aspects of the Census transform itself on a CUDA platform using a modern Fermi based GPU. They have observed that a while a cached global memory proved faster for small window sizes, shared memory produced more predictable and faster results for larger windows (the likes of which we use).

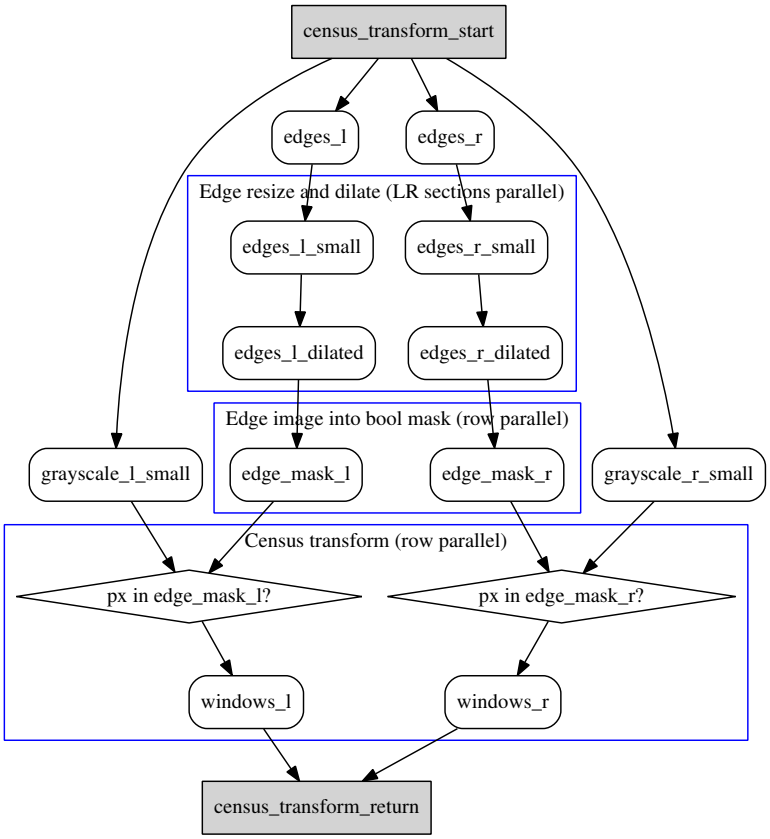


Figure 3.2: Census transform data flow

3.3.2 Census correspondence

The fig. 3.3 captures the basic idea of the correspondece computation section. This part starts with the Census transformed windows obtained in the previous section. The way the algorithm works has been explained multiple times in the previous chapters. We'll take a look at the implementation itself here.

We start off by defining a Disparity class. This class holds the integer disparity value and the disparity score (confidence). It implements setters and getters for these values, comparison operators for sorting purposes and a method for calculating the distance between two disparity values. We could

easily implement the Disparity class inline using regular STL data types, but this minimalistic implementation makes the code a little more readable without heavily affecting the computation time. We then prepare a 3D matrix able to hold multiple disparity values for each pixel in the image. Arguably this could be done in a more compact fashion, but it would heavily affect the code readability and likely the computation time. This container is only temporary for purposes of the computation itself anyway.

The algorithm starts by finding the disparities for the LR part of the problem. We parallelize using the left image rows. Moving through each row in the outer loop and through each pixel in the inner loop, we first test each window container for the default value of its x coordinate (which is equal to testing the Boolean edge mask in the previous section). If we find the x has been set, we move on to perform the LR comparisons using a hamming_distance function. Every time the hamming distance is the same or has a distance of one from the one we currently store, we push back the associated Disparity to our 3D matrix. When we find a hamming distance that is significantly better, we clear the Disparities we have already stored and push this one in instead and update our hamming distance accordingly. The score is based on the number of pixels in the correlation window, specifically as: HD/window size.

The RL search is performed in exactly the same way. The edge cases where the correlation window falls out of the image coordinates are not searched as they are prone to errors anyway and aren't expected to hold any relevant data due to the shift between the cameras, at least in the x direction.

We then continue to fill the disparity and confidence arrays. These are just simple 2D arrays in an OpenCV mat image containers do to the mathematical operations OpenCV includes. Again - this could arguably be done faster if STL containers were used and the required math was implemented directly as is our intention for the end application. To fill the arrays we must perform the selection of the best Disparity values based on their scores.

We start off by performing the neighborhood tests for both L an R Disparity matrices, which rewards continuous areas of Disparities with the same values. We move through the matrices row by row (in parallel) and column by column, working through the left and right matrices at the same time. For each pixel we create a vector holding the disparities found in the neighboring 8 pixels. For each disparity value in the current pixel we then search and count the number of its occurrences in the neighborhood and reward the current one's confidence accordingly.

The LR and RL consistency checks are performed in the following step while still in the same loop. In this case we reward those disparities in the same pixels that link back to themselves in the opposite disparity matrix - they are contained in the Disparity vector stored at the corresponding position.

For both the left and the right matrices and each of their respective pixels, we select the best disparity according to its confidence and store both the confidence and the disparity in their respective OpenCV containers. Pixels

which have no associated Disparities are omitted and left black.

As far as GPU parallelization is concerned, this is likely the most problematic part of the algorithm as the two images are dependent on each other for the correlation itself and the individual pixels within a single image are dependent on each other for neighborhood tests and then on the opposite image again for the LR and RL consistency checks. It seems most plausible to shuffle the steps and first parallelize by rows for the LR and RL correspondence and consistency checks and then parallelize overlapping 2D image tiles for the neighborhood checks.

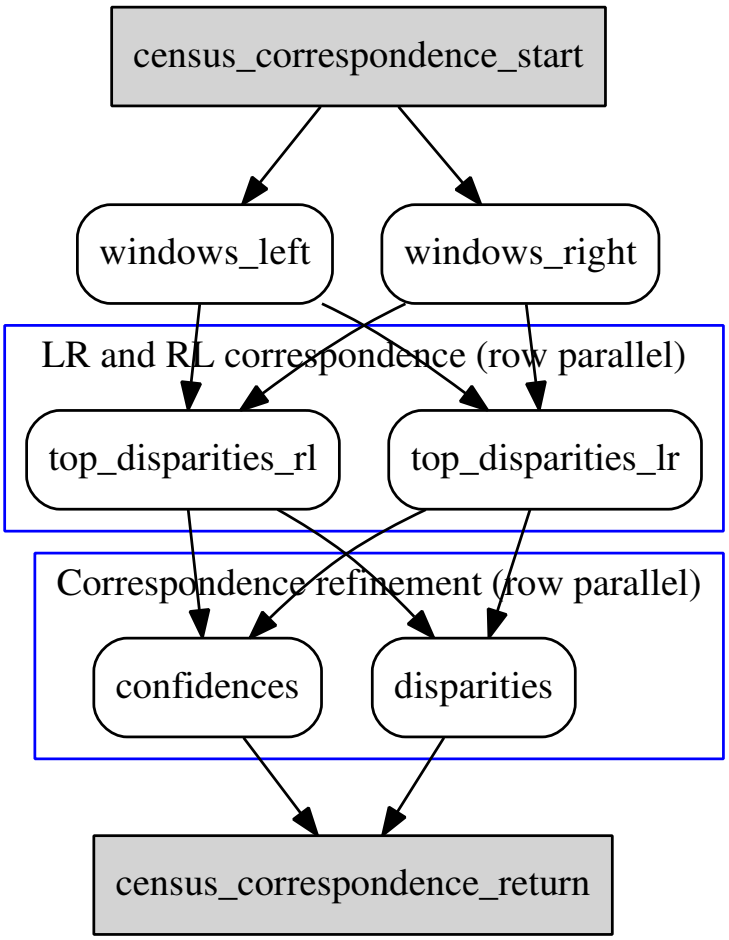


Figure 3.3: Census correspondence data flow

3.3.3 Watershed segmentation

Watershed segmentation data flow is captured on 3.4. The segmenation preprocessing has two major branches. Given an input of an edge image (in our

case we base the segmentation as well as the resulting disparity image on the left input) and the full resolution color image, we start off by transforming the edges into contours and filtering out the small area contours which may represent noise in the image and are usually not interesting for correlation purposes. We use the remaining contours to draw a filtered edge binary image which is then distance transformed and thresholded. This way we can expect to have some sort of a marker in every major segment area. Once again we perform a contour search on these and assign them each a different colour. The resulting image is used as seeds for the watershed algorithm itself.

In the other branch we take the input colour image and apply a median blur to it in order to remove noise and uninteresting features while preserving object edges. This approach proved to increase the watershed segmentation efficiency. Both the markers and the processed image are fed into an OpenCV implementation of a watershed segmentation algorithm, which provides us with an image stored in the markers input where the individual segments are each drawn with a different colour. These areas are separated into contours, which are returned as the result of the watershed segmentation.

As far as GPU segmentation is concerned, watershed is a very fast algorithm even on modern CPUs. The authors of [40] have observed speedups of up to 3 times when comparing modern midrange CUDA capable GPUs to CPUs. That said, there are a number of algorithms better suited for parallel processing such as the Really Quick Shift described in [41] which allows for processing of 256×256 images at $10 - 15$ FPS. Image segmentation in general is a fairly computationally intensive task and as such any relaxing condition and algorithmic alternative should be explored for the purposes of the final solution. Watershed however is a well defined standard solution in cases where we have reliable seed data and represents one of the faster solutions.

3.3.4 Interpolation

The interpolation is a necessary last step to obtain dense disparity information from a sparse disparity map. The algorithm data flow is captured on 3.5. Numerous approaches could be selected, but we need to focus on computational complexity first and foremost. Whatever direction we choose, we will end up calculating and setting the pixel intensities one by one as there is no other way to obtain a similarly smooth gradient across the surfaces. Relaxing this requirement and drawing entire small patches results in negligible speedups.

We start with a sparse disparity measurement obtained in subsec. 3.3.2 and the watershed segmentation contours. The contours are drawn into binary masks and eroded. Erosion is a fast morphological operation resulting in slightly smaller areas while preserving their general shapes. This way we can ensure we are calculating the gradients from the inner disparity measurements of a contour rather than the ones on the exact edge or even those falling into

neighboring segments, which would inevitably result in disparities bleeding over and affecting the neighbouring segments.

The binary masks are again searched for contours. Contours are basically vectors of edge points and their coordinates. There is likely to be a number of measured disparity points around each vertex of a given contour, but we are not actually guaranteed to find one, much less one that is representative of its surrounding area. In order to ensure the smooth transition from measured data into an interpolated area, we search the area surrounding each vertex for at least 3 measured points (or more), iteratively growing the searched area up to a certain point where we proclaim the area void of measurements and move on. After we obtain a satisfying amount of values, we compute their mean and use that as a representative disparity at a vertex. This way we traverse the entire contour.

The way interpolation works afterwards is a very simple, computationally complex, but an ideally parallelizable task. We obtain a bounding rectangle of the starting (non eroded) contours and move through each pixel in it. We perform two checks, the first one tests whether a given pixel falls within the contour using the binary mask, the second tests whether a given pixel already has a measured value - in which case it is skipped. If both conditions are satisfied, we have a viable interpolation candidate. We compute a weighted average over the vertices on our eroded contour based on their distance from a given pixel and set it as the pixel intensity - and thus obtain the disparity.

There are a number of possible optimalizations we could perform as the interpolation step is indeed computationally intensive, but the speedups mostly don't outweigh the quality losses. One possibility that would however heavily depend on the required end precision would be to skip certain rows and columns in order not to interpolate every single point using the admittedly excessive number of measured points along the contour and instead e.g. fill every other row and interpolate between them during a second pass.

Interpolation is an ideal candidate for GPU parallelization and would likely provide us with enormous speedups. While we cannot efficiently segment the masks into chunks where we could ensure minimal thread divergence in the contour finding and preprocessing stage, CUDA is an extremely fast tool when it comes to the interpolation (computation of midpoints) itself. Modern gaming GPUs (which are usually the preferred for computationally intensive CUDA tasks on a budget) are actually ideally equipped to do just that, dynamically compute the colours and shadings of various 3D surfaces, which can be leveraged nicely.

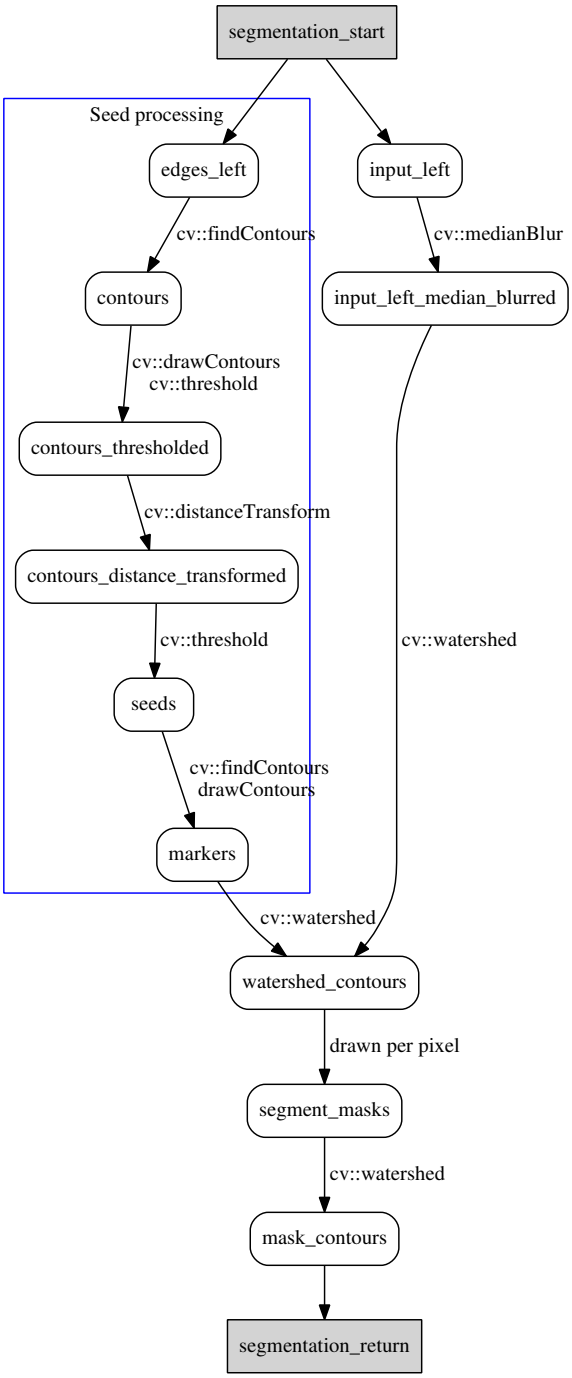


Figure 3.4: Watershed segmentation data flow

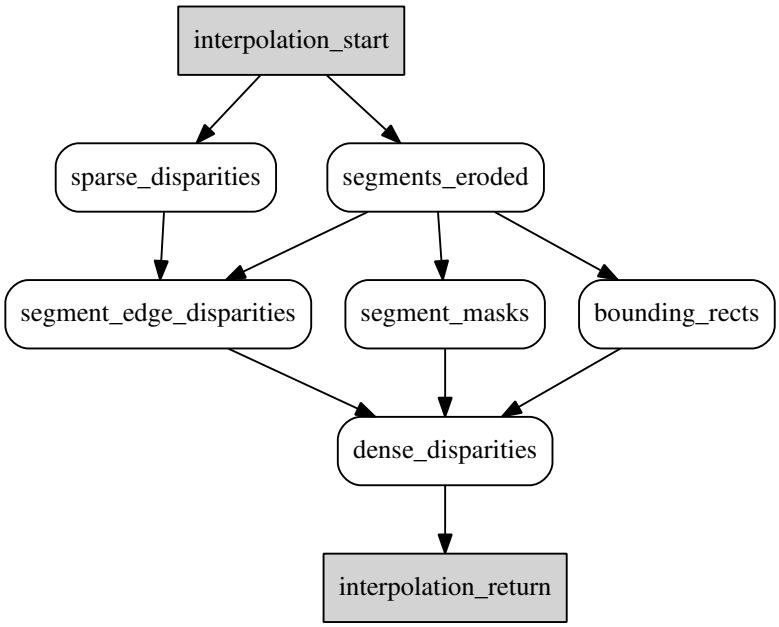


Figure 3.5: Disparity interpolation data flow

Chapter 4

Experiments

For experiments, we will use images of the Middlebury dataset and their corresponding ground truths available from the authors of [42], which are often used as a standard reference for stereo related research. A number of different tests were devised in order to gauge the algorithm robustness, speed, and precision when compared to other algorithms and to various modifications of the algorithm itself.

As mentioned previously the algorithm consists of three main sections, the transformation, the correspondence and the interpolation. These parts can be heavily modified without directly affecting each other. The interpolation itself poses a problem of being ill defined. There are numerous factors affecting the selected interpolation approach when dealing with sparse edge disparity information. Some interpolation methods can take ages to finish, while some are rather fast. There are numerous differences in the resulting smoothness, edge behavior, precision. All of the above makes the interpolation a candidate for separate measurements - we will therefore usually evaluate both parts independently.

The section 4.1 deals with the computation time for both parts of the algorithm and evaluates the overall speed of the algorithm as a whole for different inputs. The section 4.2 attempts to compare the algorithm to existing solutions and to modifications of the algorithm itself and shows how the input parameters change the obtained results. Finally, section 4.3 identifies the problematic parts of our approach, shows the circumstances under which the algorithm may possibly fail and discusses the implications these problems have for the intended end approach.

4.1 Algorithm speed

In this section we will measure the dependency of computation time, parallelized and sequential, for different inputs, different window sizes and different

(a) The left image

(b) The right image

Figure 4.1: Middlebury 2014 Stereo datasets Adirondack input

input resolutions. We will also analyze the computation time and compare the algorithm to some of the other available solutions.

There are a number of important considerations when implementing a stereo vision algorithm. A number of experiments could be conducted to verify the algorithm complexity. Some of the aspects that greatly affect the speed of the algorithm are the correspondence window size - which affects both the Census transformation and the correspondence algorithm, the input image dimensions - which affect all parts of the algorithm to a degree and also affect the algorithm parallelizability, repetitiveness of the measured scene - thanks to which the correspondence algorithm may not properly identify a number of false positives in the initial steps and which results in a larger state space for the consistency checks and last but not least the algorithm is greatly affected by the scene complexity itself as a complex scene results in more points being identified as object edges and therefore results in a larger state space for the correspondence search itself.

4.1.1 Correspondence window dimensions

A number of abovementioned algorithms were implemented during the course of development of the correspondence application. Namely Census, SAD, SSD and the Complete Rank with SAD correspondence metric. Each of these algorithms differs in its memory consumption, precision, robustness, parallelizability, inherent computational complexity etc. A brief overview of the last point is in order so we can see why Census was chosen in stead of the other alternatives. While computational complexity does not tell the entire story, it is a very important factor for the selection of an ideal candidate for a near real time system.

In order to gauge the complexity of the correlation itself, only the relevant parts of the algorithms were measured. Namely the transformation, correspondence and refinement. Window sizes ranging from 3×3 in odd increments

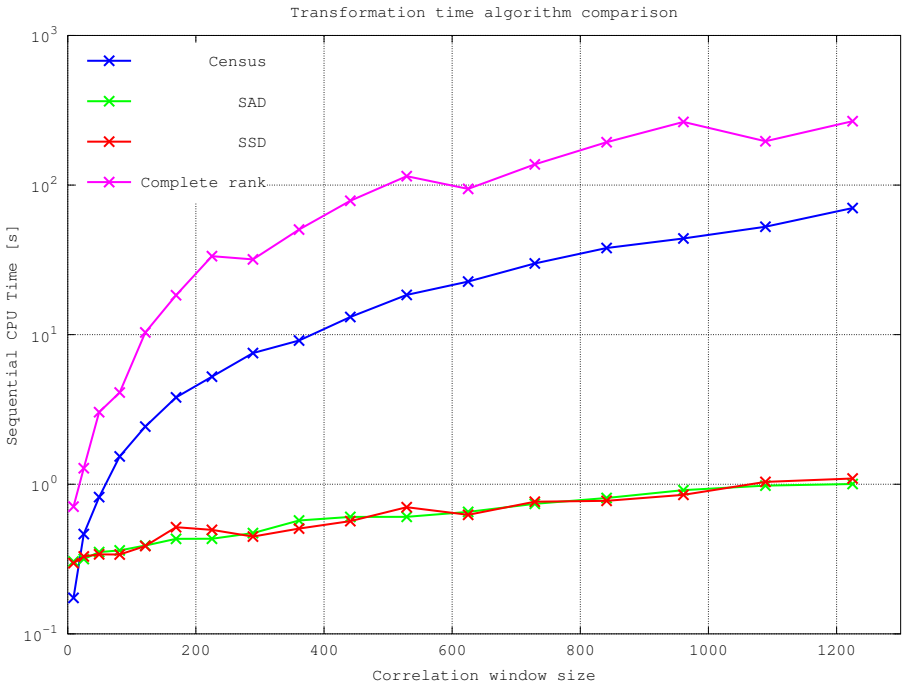
up to 35×35 were measured as the algorithms behave very differently for different window dimensions. While Census may shine for large windows due to its simplicity, the other metrics are usually used with smaller windows, therefore an overview is in order.

The computation time was measured for a sequential version of the application so we can make observations about their complexities. OpenCV parallel optimizations were turned off during the run. Each algorithm section was measured separately up to 10 times and the times were averaged. The image pair used for testing can be seen on 4.1. To really see how the correspondence algorithms themselves perform, a search over the entire image pair was performed (no edge based masking was used). The input image was resized to 25% of its original size, so the correlation was performed on resolution of 720×497.

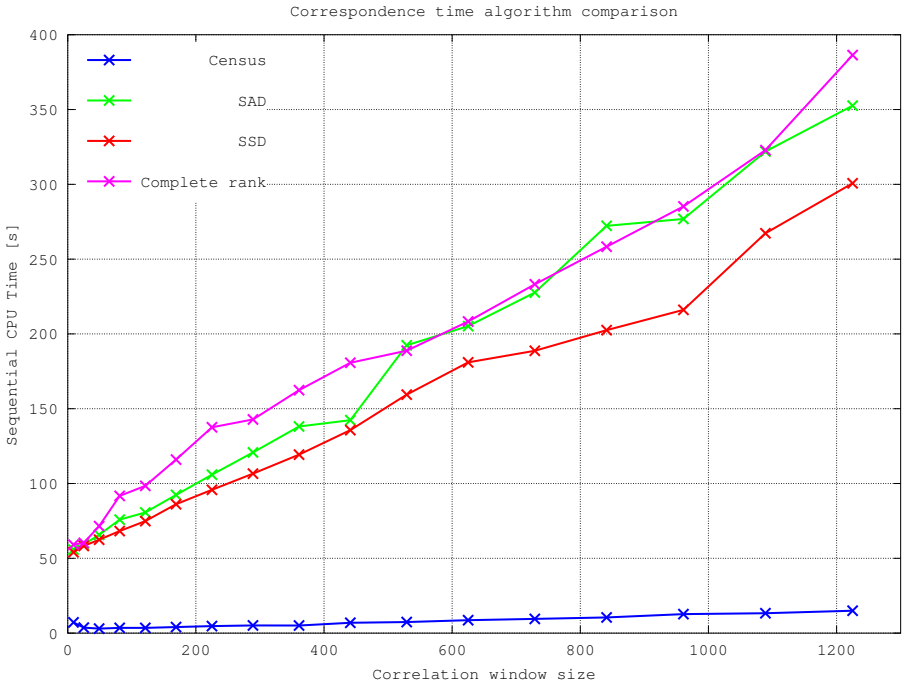
Fig. 4.3a shows the dependency of the transformation time on the rectangular window size. Due to the massive differences between the different transforms, the plot y axis is logarithmic. We can see that the section is clearly dominated by the Census and Rank transforms. The rank transform has an upper limit of $w_{\text{width}} w_{\text{height}}$ compare - write operations, where $w_{\text{width}} = w_{\text{height}} = w$ for each pixel in the window, which means the overall complexity is $O(w^4)$. For the Census, we only perform a single compare - write operation for each pixel in the window, which results in complexity limited by $O(w^2)$. The complexity of the entire image transform is then bound by $O(I_{\text{width}} I_{\text{height}} w^4)$ for the Complete Rank and $O(I_{\text{width}} I_{\text{height}} w^2)$ for Census. The remaining two, Sum of Absolute Differences and the Sum of Squared Differences perform no transforms at all, they just copy the data into a new array and the time dependency is therefore linear, even if the tendency isn't clearly visible on such small data batches, and can be written as $O(I_{\text{width}} I_{\text{height}} w)$.

The figure 4.2b shows the dependency of the correspondence computation time on the window dimensions for the same algorithms. We can see 3 distinct linear algorithm types. The SAD and the Complete Rank both use the L1-Norm for difference measurements. The SSD algorithm uses the L2-Norm, otherwise known as Least Squares. Surprisingly, even though the L1-Norm is a simpler metric than L2, we can see that the L1 based algorithms perform worse than their squared counterpart. The difference is however minor, dependent on the internal OpenCV processing and likely to disappear when the optimizations are turned on as most modern CPUs implement SIMD instructions specifically for this purpose on a hardware level. The third type we can see is the Census, which performs much faster than its counterparts. The Census algorithm sacrifices the number of similarity levels, of which there would normally be $2^{w_{\text{width}} w_{\text{height}}}$ if the window was interpreted as an integer, for simplicity and speed by using the hamming distance, which results in just $w_{\text{width}} w_{\text{height}}$ similarity levels, but at the same time allows the algorithm to take a much larger area into account.

All off the algorithms perform the Left-Right and the Right-Left corres-



(a) Transformation sequential running time algorithm comparison



(b) Correspondence sequential running time algorithm comparison

Figure 4.2: A sequential computation time dependence on a square correspondence window dimensions for different implemented algorithms.

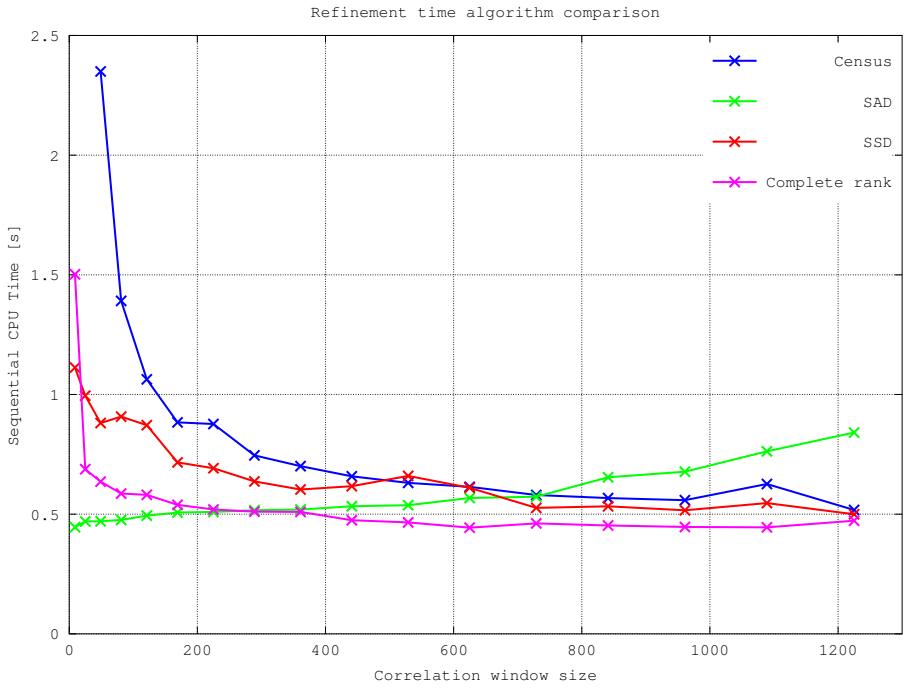
pondence steps in a very similar fashion. If we limit ourselves to a single row and LR correspondence, we start at the beginning of the row and compare the first pixel to every pixel in the opposite row. In the next step we compare the second pixel to each pixel in the opposite row, but skip the first one. This means that a single row correspondence requires, if no further optimizations are employed, $I^2_{\text{width}}/2$ difference operations to finish. If we extend the computation for all rows, we get $\frac{I^2_{\text{width}}}{2} \times I_{\text{height}}$. As the algorithm performs the search in both directions, the fraction disappears: $I^2_{\text{width}} \times I_{\text{height}}$. The difference operations themselves are different for each of the algorithms. Each of the various differences perform w^2 operations, but their actual complexity is platform dependent and constant. All in all we can estimate the complexity as $\Theta(I^2_{\text{width}} I_{\text{height}} w^2)$, although we can effectively remove the square by limiting the minimum distance from the cameras.

The complexity of the remaining step is almost impossible to estimate as the search space is heavily dependent on the previous steps and the analysis would bring very little information. As the figure 4.3a shows, the computation times are negligible in comparison to previous steps and all of them show roughly linear to linearly rational dependency. We can see that the larger the window, the more precise the results are and the smaller the refinement state space is. In essence we need to perform fewer consistency checks for larger windows as there are fewer occurrences of a pixel projecting into multiple other pixels with the same score. As the Census metric has the fewest levels, we can see it performs worse at this step than its counterparts. The difference however grows negligible for larger window sizes.

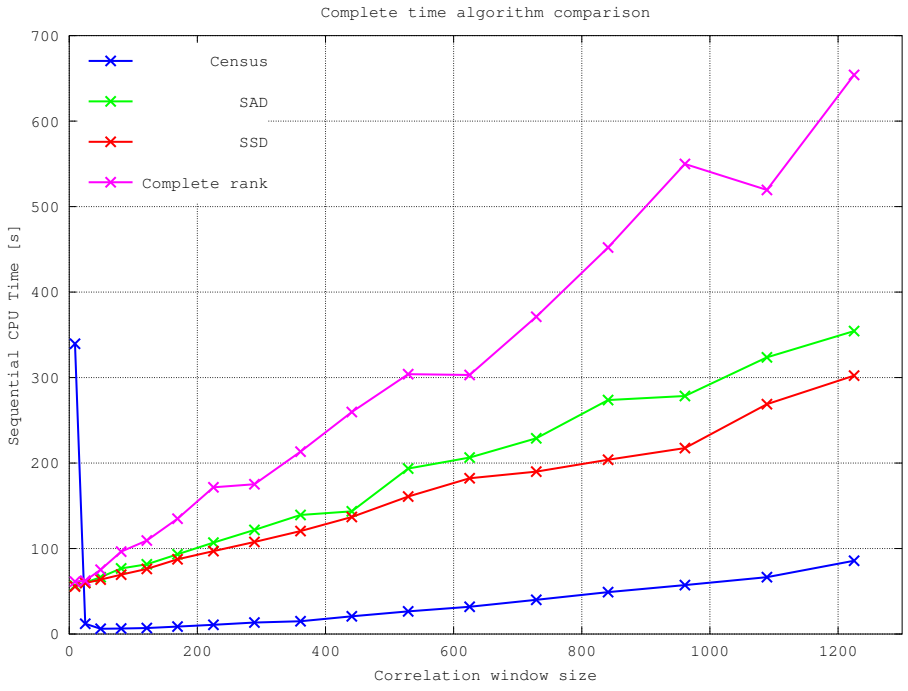
The sum computation time for the above steps is captured on 4.3b. In conclusion, we can see that Census clearly outperforms its counterparts in most situations, sometimes by a factor of 10. This potential is the main reason behind its selection for the purposes of near real time correspondence. As we will further discuss, the expected quality degradation in regards to fewer difference levels is not as prominent once we reach a certain window size.

4.1.2 Image size

The image dimensions play a crucial role in algorithm performance as well. Based on the above, we have a rough estimate of $O(I_{\text{width}} I_{\text{height}} w^2) + \Theta(I^2_{\text{width}} I_{\text{height}} w^2)$, which means the image width and the window size are the two most important factors for the computation time. The algorithm scales linearly with image height and the coefficient is dependent solely on the correspondence window size. The situation for image width is the same, only the dependence is squared and a little more interesting. To verify our assumption an experiment resizing the input image pair from figure 4.1 only in the x dimension was devised. We start with the full horizontal resolution of 2880px and iteratively downsample the image by 50% for each run while keeping the y dimension constant. Naturally, the algorithms will be gradually less and less successful searching for



(a) Refinement sequential running time algorithm comparison, note the logarithmic y scale



(b) Overall sequential running time algorithm comparison

Figure 4.3: A sequential computation time dependence on a square correspondence window dimensions for different implemented algorithms.

the disparities, but this mostly affects the refinement stage, which we won't concern ourselves with. As in the previous case, no masking was applied and the algorithm was ran 10 times for each section and the values were averaged. A 11×11 correspondence window was used due to memory restrictions. No optimizations were turned on.

The figure 4.4 shows the differences between the running times for dense algorithm implementations for changing image width. We can see that while the dependence is roughly quadratic in all cases, Census, if it can withstand the precision tests, is the clear winner as it performs at least an order faster compared to the competitors.

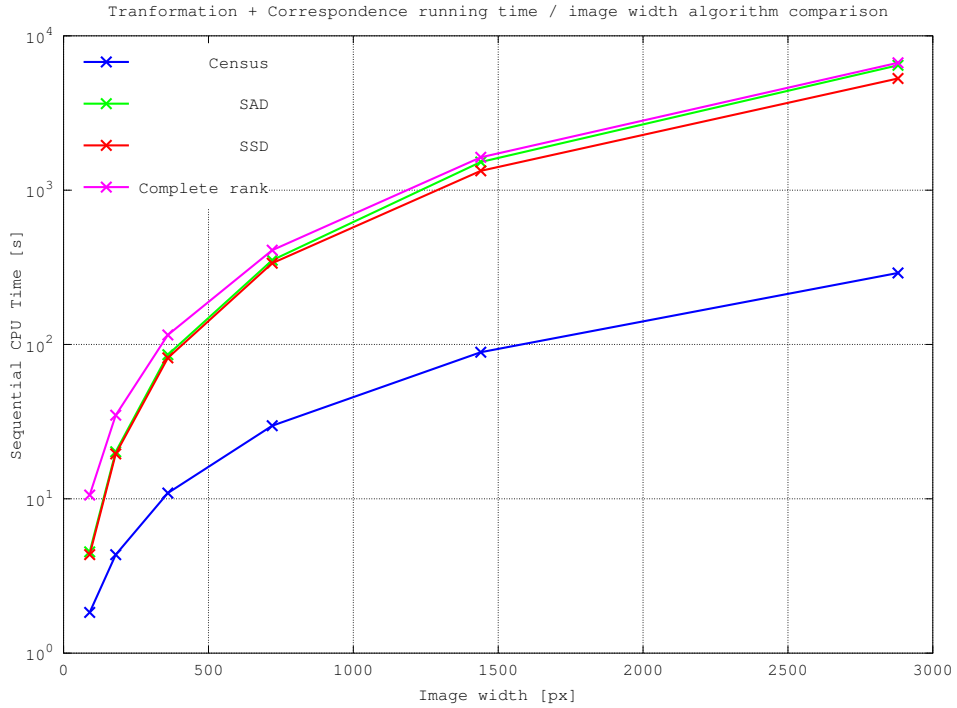


Figure 4.4: Transformation and correspondence running time sum dependence on image width for different algorithms, note the logarithmic y scale

4.1.3 Optimizations

Turning on the parallelization and the OpenCV optimizations has a profound effect on the algorithm performance. The presented implementation is by no means ideal. OpenCV creates a lot of overhead which could be avoided by writing the code from scratch. For example when it comes to the copying in the transformation parts of the SAD and SSD algorithms, we have to create whole new OpenCV Mat objects for each window. This is still incomparably faster than just referencing the parts of the original array as it solves the

(a) Transform parallelization speedups

Algorithm	Sequential wall time [s]	Parallel wall time [s]	Speedup
Census	2.431	0.849	2.862
SAD	0.389	0.241	1.613
SSD	0.386	0.182	2.115
Complete Rank	10.343	4.073	2.539

(b) Correspondence parallelization speedups

Algorithm	Sequential wall time [s]	Parallel wall time [s]	Speedup
Census	3.540	1.548	2.287
SAD	80.678	42.017	1.920
SSD	74.861	43.180	1.734
Complete Rank	98.439	48.237	2.040

Table 4.1: Sequential vs. parallelized algorithms computation wall times and the associated speedups for an 11×11 correspondence window.

issue of conflicting data reads. OpenCV also, by its internal architecture, prevents us from implementing the algorithms in a way that would require as many independent parallelized sections (see chapter 3 for implementation details), which does result in a bit of an overhead due to the thread creation. Given a rather underpowered machine like the Macbook Air mentioned at the beginning of this chapter and the heavy interconnectedness of OpenCV and the X Window System, we were not able to meaningfully measure the dependence of running time on the number of used threads on a dedicated server.

We will however compare the running time between parallel and sequential runs of the algorithms. We will use 4 threads running on a dual core machine with Hyper Threading wherever possible. The algorithms ran on a 25% resolution of the same image pair on figure 4.1, no masking was applied. A correspondence window of 11×11 was used, no calibration corrections were performed and the masking and segmentation stages were once again entirely omitted from the results as they are irrelevant for dense correspondence.

The figures 4.5a and 4.5b show the differences summarized in the tables 4.1a and 4.1b. The results seem fairly unimpressive, but the algorithms employed so far are all easily parallelizable and the machine they were running on is underpowered for the task. We can however see that Census is the most readily parallelizable of the 4 algorithms. The end application is intended to run on a multicore system equipped with a GPU for image processing acceleration. We cannot expect the machine to perform the task of correspondence calculation in real time, as the Elphel cameras provide hundreds of frames per second for the 2592×96 resolution. The previously employed application used a single camera to capture the vehicle undercarriage and performed a

near real time image stitching to generate a shot of the entire car underside at resolutions around 2592×10000. A large chunk of the original 96 px vertical resolution was lost in the process.

Based on the above we have two options, either compute the depth information on the fly, sample by sample and accept significant overhead caused by the overlapping images. This is a straightforward task which would likely result in better precision and actually help us generate more precise image stitching results using the depth information for corrections. We would however likely end up not using the GPU to its full potential as the images we would send to it are simply too narrow. We also have to concern ourselves with the information loss near the image edges as the correlation window requires a padding of size $(w_{height} - 1)/2$ in the y direction. While it is likely a nonsquare rectangular window will be leveraged not just to suppress this effect, but to speed up the computation as well, as the row data reads are faster due to the internal data representations. We also have to allow for post processing as there is no guarantee of depth continuity at the strip stitching lines.

The other approach is to at least partially stitch the resulting image, offload the data to GPU for processing and meanwhile keep busy stitching the rest of the images. Running the correspondence on larger chunks of data would likely result in lower precision as the strip stitch lines are never perfectly aligned, but it would also result in significantly higher speedups as we could leverage more of the processing power of the GPU at once. We would have the possibility of making sure the transitions at the stitching lines are continuous. Last but not least tere would also be less overhead due to the already overlapping rows.

In addition to the previous, a way of possible computation speedup was devised by means of segmentation of the input and later interpolation of the data from the values on the segment edges. This way of computation naturally works with the most prominent data segments in the image, the edges. By masking the highly uniform areas which we expect to be continuous and fairly flat, we can, in theory, not only achieve higher throughput, but also obtain more precise, even subpixel disparities. This is given by the fact that the algorithms tend to inevitably perform most of the unnecessary (meaning later filtered out) computations in the uniform areas, which result in low confidence values and produce many outliers. At the same time the edges are easy to identify and by reducing the state space to them we not only limit the number of operations, but also reduce the risk of disparity miscalculation significantly.

The algorithm performs Sobel based edge identification and later applies a rectangular dilation element on the edges. The dilation is performed after resizing the edge map to match the resized input image. For most images, the ideal state space reduction seems to be around 50%. We cannot perform the search solely on the edges as we are mostly interested in the areas closely surrounding them, from which we interpolate.

The number of segments and the state space reduction very much de-

pends on the input image. For the image used in the above tests, around half the areas can be considered edge surroundings. For simpler scenes like the vehicle undercarriage, we can hope to achieve results around 30-40%. The reduction in the state space is similar to reduction in the image width and achieves similar speedups as in the above section 4.1.2. We will therefore skip the transformation and correspondence metrics and take a look at the computational reductions in refinement and the complete running time and observe the segmentation and interpolation times as well. We will also skip the measurements of the rest of the algorithms, as the effects and the state space reductions affect them in exactly the same way and the segmentation and interpolation steps are implemented identically.

In order to present the results in a consise fashion, the same fig. 4.1 image pair was used, the images were resized to 25%, the correspondence window size was set to 25×25 and the computations were performed for multiple different dilation element sizes all the way up to dense correspondence. The figure 4.6 presents the results for refinement, segmentation, interpolation and the complete algorithm running time. We can see that the first three remain largely unaffected. For refinement, there are small changes in the state space, but the algorithm is rather fast and differences are mostly not noticeable for inputs the size of images. The interpolation time is affected in a very minor way, as the maximum limit of the pixels we're interpolating from remains the same, even though we will likely find enough values closer to the starting segment vertex. The segmentation itself is completely unaffected by the dilation and depends solely on the number of segments and image size.

What remains is the correspondence and transformation computational complexity, which has been discussed thoroughly above. All in all, it remains questionable whether the segmentation and interpolation steps actually help with achieving higher throughput. It is definitely a useful tool for large window sizes, as it can significantly reduce the computation times for the correspondence and transformation, but for most practical window sizes of which the 21×21 is a representative, the computational speed gains are immediately replaced by the losses over the segmentation and interpolation.

The segmentation times are unlikely to change much. While our implementation isn't particularly optimal (the OpenCV watershed algorithm itself takes about 10% of the total segmentation time), the employed pre and post-processing takes up a lot of resources. A faster GPU based segmentation alternatives could possibly be implemented with speedups of up to 3 times as mentioned in subsection 3.3.3, the segmentation remains computationally very intensive. The interpolation itself could see significant speed improvements on GPUs, especially for large image inputs.

4.2 Precision

This brings us to another important aspect of stereo correspondence, the precision. While a number of experiments could have been devised with our own set of cameras, the standard Middlebury datasets seem to be a clear choice given their provided ground truth, for which they are used in most researched published on the subject of stereo correspondence. We will show how the various settings affect the different algorithms, provide results and meaningful comparisons to the ground truth data.

4.2.1 The dense disparity

As was the case in the previous section, we will fist show how various settings affect the resulting disparity data on the dense, non-masked, runs to better gauge how the correspondence algorithms themselves behave. Multiple scenes will be used to show the different behaviors of the presented algorithms. The chair scene from the previous section on figure 4.1, the motorcycle on 4.7 and the trash bin on fig 4.8.

To begin with, we present a visual overview of the algorithm performance on figure 4.9. All the different algorithms were used to compute the dense disparity maps for the chair input pair with various square correspondence window dimensions. The last row presents the interpolated segmentation data based on the last column 21×21 outputs of each algorithm. The black areas on the first 4 rows represent parts of the image where the confidence of the correspondence wasn't high enough to pass the confidence masking and the disparity was removed from the result in order to not affect the data normalization required for presentation. The overall relative differences in the image intensity between the ground truth and the computed disparities is mostly caused by this normalization as well as the disparities had to fit into a 0-255 intensity range for purposes of presentation and any high intensity outlier that didn't get caught in the filtration steps can cause a shift in the entire image intensity.

Overall the SAD and the SSD algorithms seem to be the worst performers of the lineup. While the algorithms have computed fairly precise results in the prominent areas around the edges, there are large segments that aren't populated at all. This can be solved using interpolation as we can see on the last row, but then the algorithm computation times are simply too high to be of any use. Along the edges, in the occluded areas and, when the result is defined, in the low textured areas the algorithms show a number of outliers. Not just separate sparse points, but entire segments misclassified completely.

The Complete Rank based solution seems to perform rather well overall, the results are some of the most consistent with the ground truth images of the 4 even for smaller window sizes and the areas around the edges are densely

Algorithm	Adirondack [%]	Motorcycle [%]	Recycle [%]
Census	2.908	2.431	3.118
SAD	3.437	2.924	1.446
SSD	2.977	2.483	1.294
Complete Rank	1.473	1.484	1.461

Table 4.2: The relative errors for the dense disparity map computation using a 21×21 correspondence window for different algorithms.

populated as is the case with SAD and SSD. The occluded areas are mostly not defined, as are the low texture areas.

The Census algorithm results are the sparsest of the set, but they are well defined around the edges, mostly undefined in the occluded and low texture areas such as the chair back support and show significantly lower number of outliers than SAD and SSD. Given the remarkably low computation time in comparison to others

The figures 4.10 and 4.11 show various problems of all the algorithms. The Motorcycle scene is a very densely populated one. The occlusions cen be clerly identified along the rightmost edge of the bike. All of the algorithms perform rather well in this scene, but the interpolation step seems to introduce a certain amount of softness as it was not tuned for such highly textured scenes and does result in a bit of a detail loss. The Recycle scene is problematic for all of the algorithms beacuse of the surface uniformity, but Census seems to have most problems. Increasing the correspondence window size should put it on par with the rest of the algorithms and still keep the comptation time below the rest.

To better measure the algorithm performance, we will turn the disparity filtering based on confidence off for a while and compare the results to the ground truth. We will compute the pre-normalization L1-norm between the measured data and the ground truth image and present the results for the three different scenes for a 21×21 correspondence window as a percentage obtained by dividing the norm by the maximum possible difference representing the 100% error.

The table 4.2 summarizes the relative errors for different metrics and input scenes. We can see that while Census generally sits in the middle between the Complete Rank and the SAD / SSD metrics as far as quality goes - which is consistent with the observations on the visual comparison shots for the different scenes, it is more prone to errors in large low texture areas such as the Recycle scene. Here, the algorithm would require a larger correspondence window size to be useful. We can also see that the Complete Rank transform consistently outperforms the other metrics, which was expected. Strangely the SAD and the SSD algorithms tend to thrive on the scenes Census fails on.

Algorithm	$\sigma = 1$ [%]	$\sigma = 3$ [%]	$\sigma = 5$ [%]	$\sigma = 10$ [%]
Census	3.312	3.657	4.557	7.557
SAD	3.821	4.142	4.340	8.121
SSD	3.935	4.438	5.129	9.495
Complete Rank	2.924	3.234	4.232	6.385

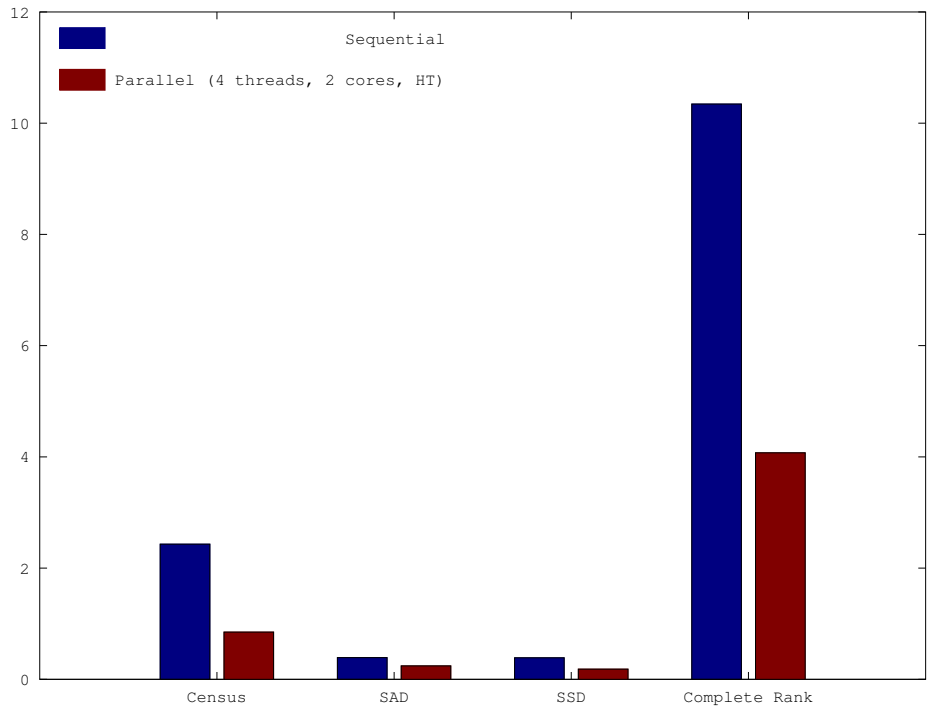
Table 4.3: The relative errors for the Adironrack scene based on the introduced random noise standard deviation.

4.3 Robustness

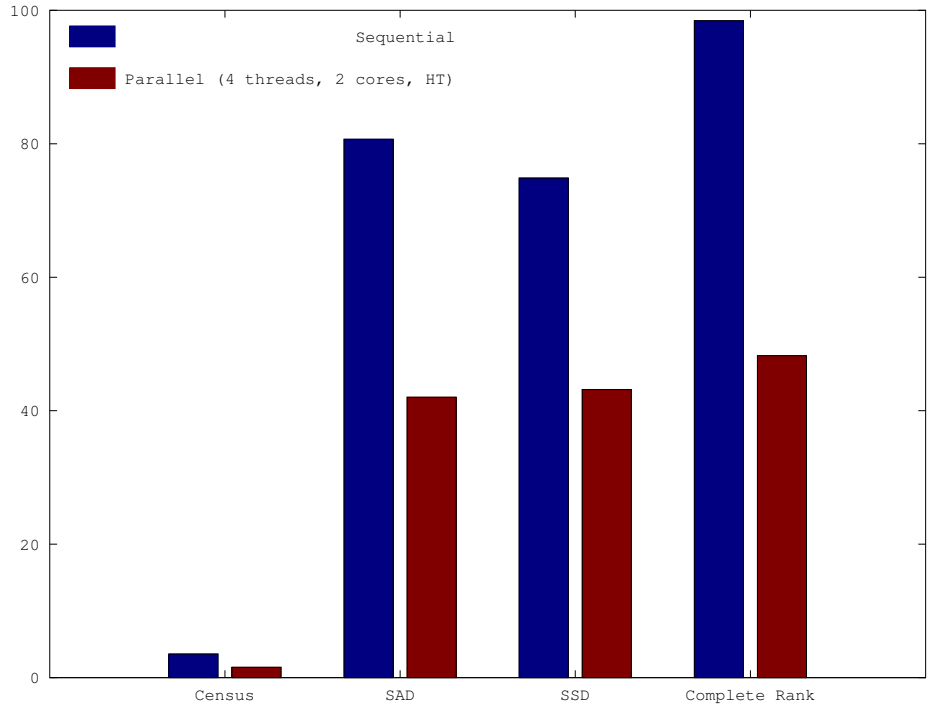
This brings us to the last section of the algorithm evaluation, which is the robustness. As was mentioned before, mostly in the chapter 1, all of the algorithms behave very differently under various imaging conditions. The previous section gave some insight into the algorithm behaviour as far as the scene itself goes, but we should take image quality into account as well. One of the most common issues when capturing high speed data is the lighting. As we set the camera exposure shorter and shorter, either we have to provide more light or we have to boost the camera gain parameters, which results in noise enhancement.

A test was devised to evaluate the relative error dependency on the standard deviation of image noise introduced into both samples. The above algorithms were ran with identical settings as in the test summarized in table 4.2, but noise was gradually introduced to the scene. Only the Adironrack (Chair) scene was used as it is the middle ground representative and contains both densely textured and low texture areas, which makes it an ideal candidate for the test.

The table 4.3 shows the relative errors for different values of the noise standard deviation. While errors seem to grow at an enormous rate given how small a visual change an addition of even the highest measured $\sigma = 10$ noise to the scene represents, the results arent surprising. The two better algorithms - Census and the Complete Rank are both non-parametric and more robust with regards to local perturbances in the scene as they rely on the local relative ordering of the pixels rather than their actual intensities. The Complete Rank transform was once again the most successful in this regard. As far as SAD and SSD is concerned, they are in the same range, with SSD displaying its tendency to enhance noise caused by the squaring of the difference.



(a) Transform wall running time algorithm parallelization comparison



(b) Correspondence wall running time algorithm parallelization comparison

Figure 4.5: A comparison of the wall times for different tasks for sequential and parallelized algorithms.

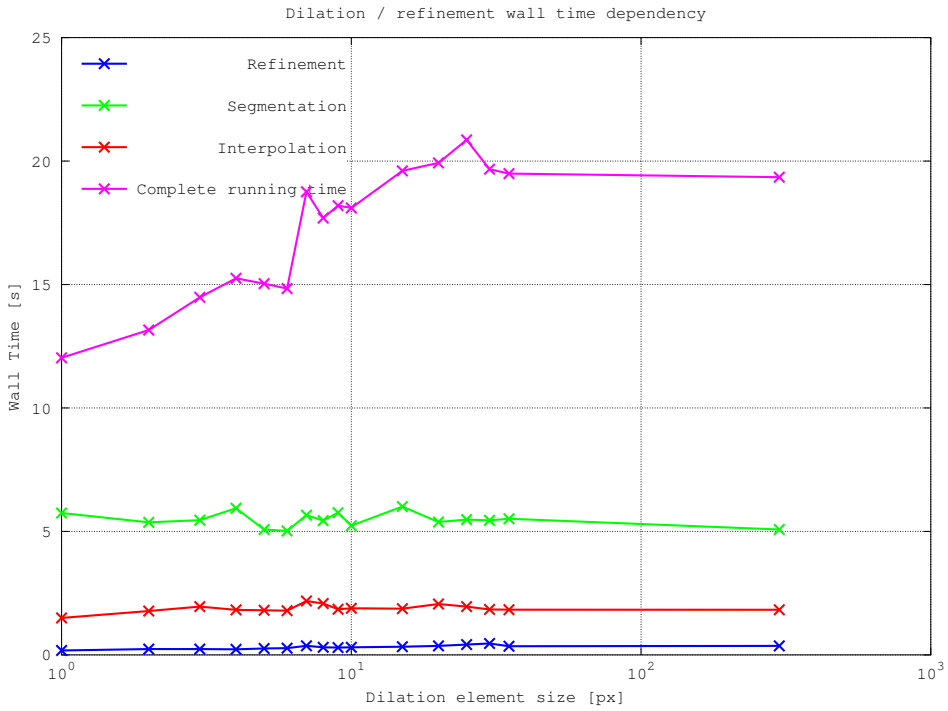


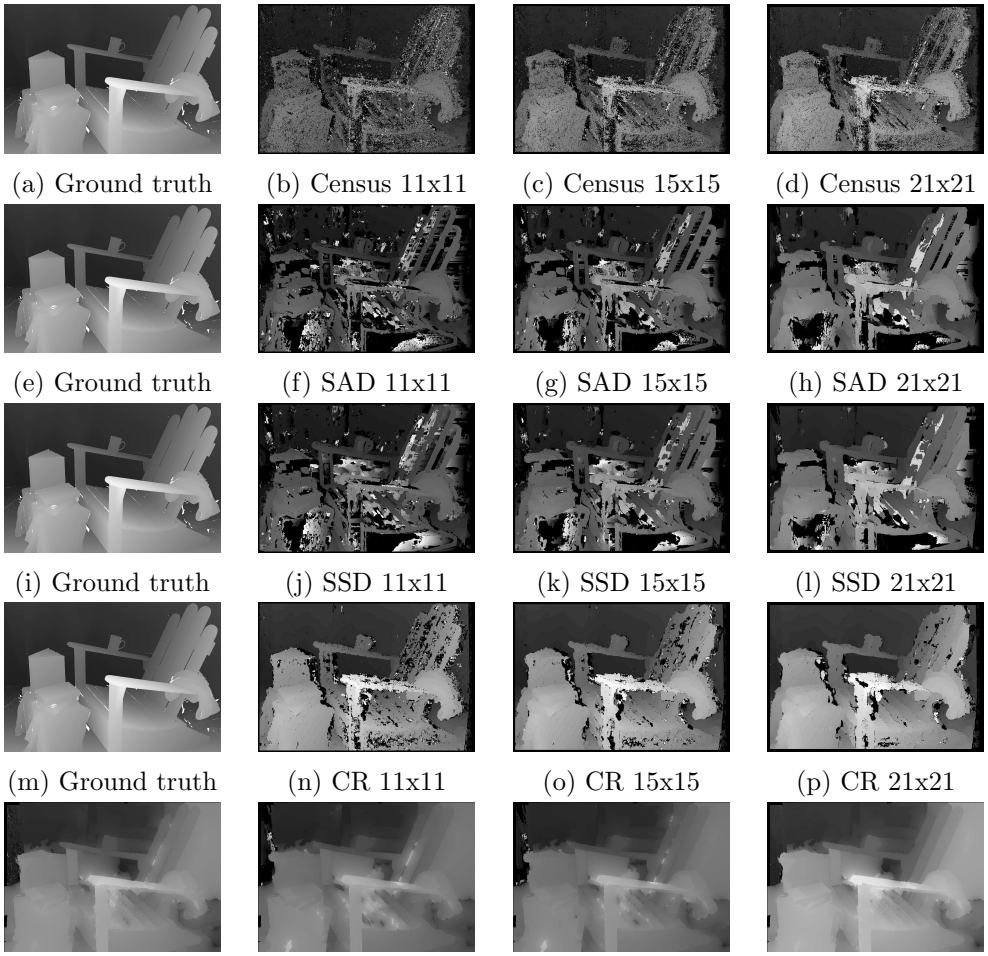
Figure 4.6: Dilation element size effects on the running time of various parts of the algorithm

(a) The left view (b) The right view

Figure 4.7: Middlebury 2014 Stereo datasets Motorcycle input

(a) The left view (b) The right view

Figure 4.8: Middlebury 2014 Stereo datasets Recycle input



(a) Ground truth (b) Census 11x11 (c) Census 15x15 (d) Census 21x21

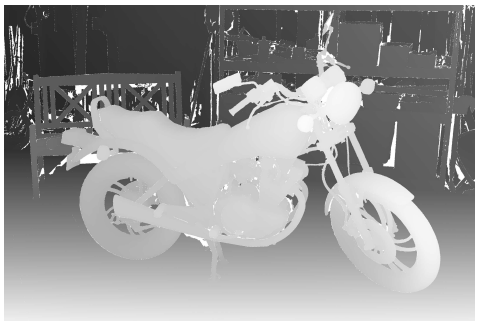
(e) Ground truth (f) SAD 11x11 (g) SAD 15x15 (h) SAD 21x21

(i) Ground truth (j) SSD 11x11 (k) SSD 15x15 (l) SSD 21x21

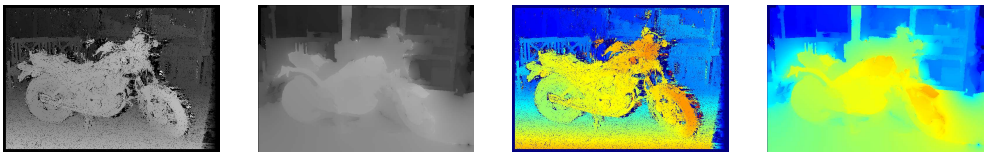
(m) Ground truth (n) CR 11x11 (o) CR 15x15 (p) CR 21x21

(q) Census 21x21 interpolated (r) SAD 21x21 interpolated (s) SSD 21x21 interpolated (t) Rank 21x21 interpolated

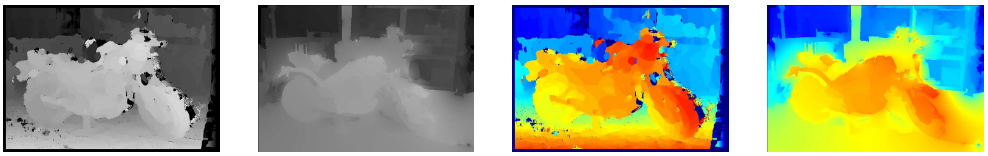
Figure 4.9: Correspondence algorithm precision performance overview for different window sizes



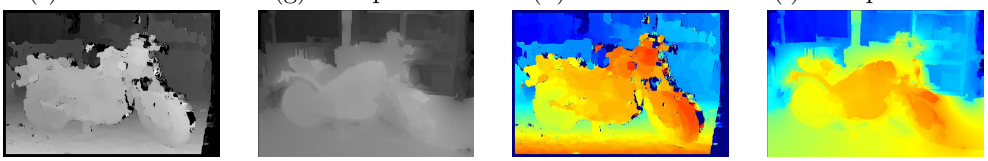
(a) Ground truth



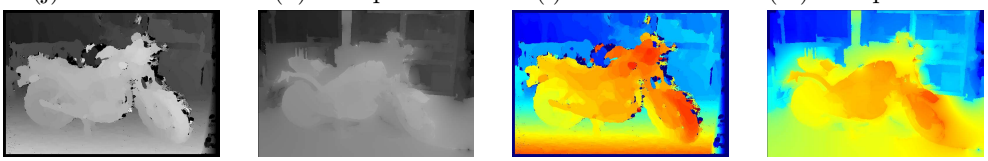
(b) Census dense (c) Interpolated (d) Dense color (e) Interpol. color



(f) SAD dense (g) Interpolated (h) Dense color (i) Interpol. color

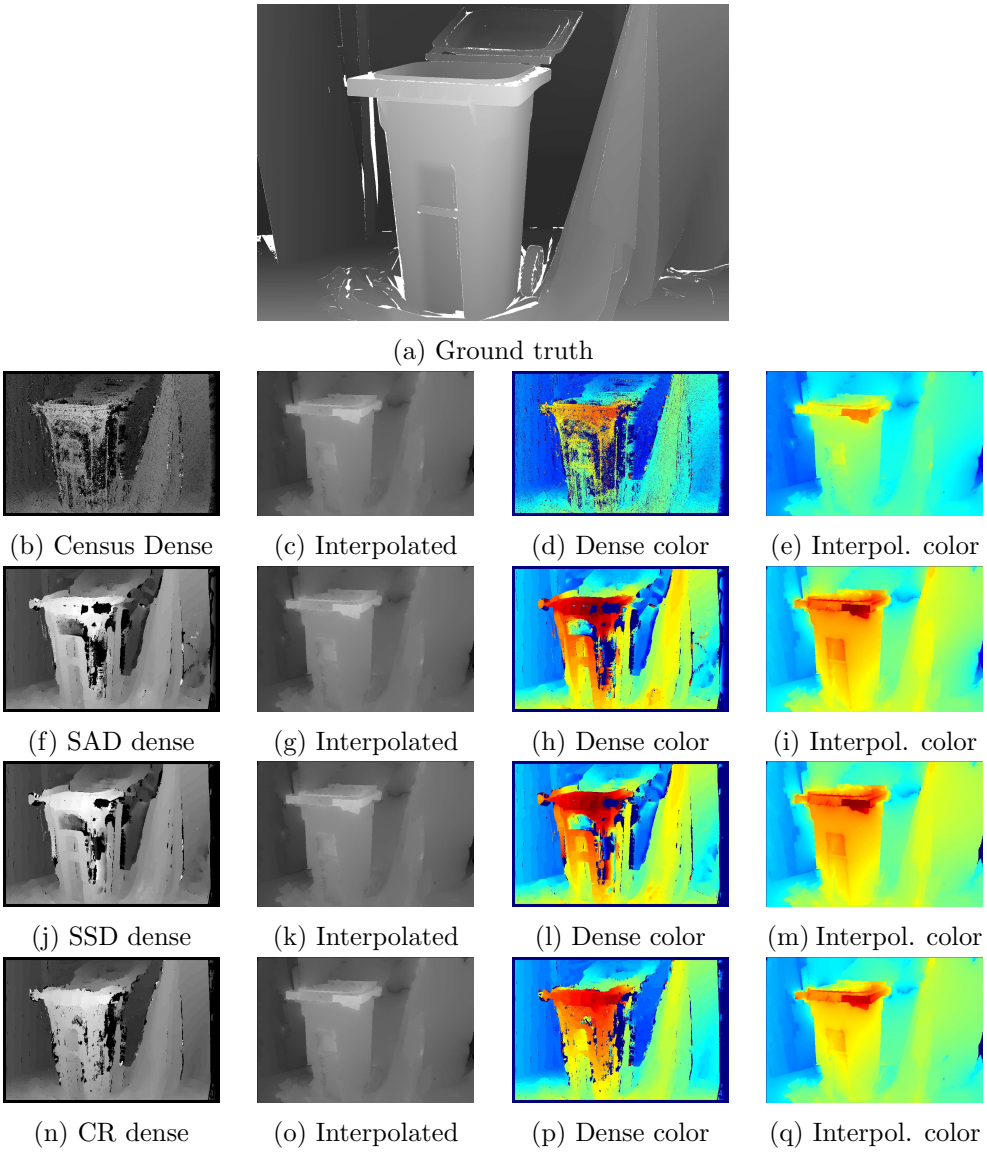


(j) SSD dense (k) Interpolated (l) Dense color (m) Interpol. color



(n) CR dense (o) Interpolated (p) Dense color (q) Interpol. color

Figure 4.10: Motorcycle scene correspondence algorithm precision performance overview for the window size of 21×21



(a) Ground truth

(b) Census Dense (c) Interpolated (d) Dense color (e) Interpol. color

(f) SAD dense (g) Interpolated (h) Dense color (i) Interpol. color

(j) SSD dense (k) Interpolated (l) Dense color (m) Interpol. color

(n) CR dense (o) Interpolated (p) Dense color (q) Interpol. color

Figure 4.11: Recycle scene correspondence algorithm precision performance overview for the window size of 21×21

Conclusion

In this thesis, multiple local stereo correspondence algorithms were implemented and evaluated with focus on the Census correspondence metric as a means of achieving precise high throughput dense depth information for a system with two cameras. The implementation was mostly inspired by state-of-the-art embedded systems employing edge based disparity computation masking and segmentation to quickly obtain the required results. The same approach was applied on the other implemented algorithms - The Sum of Absolute Differences, Sum of Squared Differences and Complete Rank transformation with SAD correspondence metric.

Census, being the simplest of the four computationally, allowed us to use much larger correspondence windows and therefore achieve relatively high precision in the same timeframe. SAD and SSD algorithms seem to be on par with each other. The SSD algorithm generally achieves slightly better correspondence computation times, even if the opposite should have been the case. This is likely caused by the internal OpenCV implementation. Complete Rank, being by far the slowest of the four due to its intensive transform stage built on top of the already slow SAD metric, proved itself to be completely infeasible computationally and doesn't seem to bring any major precision improvements either.

While the achieved computation times of any of the algorithms do not seem entirely tractable for employment in a near real time system as was the original intention at this point in development, interesting results were observed and multiple optimization possibilities were proposed. Census proved itself to be a robust metric, while achieving computation speeds orders of magnitude lower than the rest of the local based competitors and thrived where some of the others failed. Implementation of a made to measure Census based solution from scratch without OpenCV dependence and levaraging GPU processing power along with the internal camera FPGA capabilities for which Census is almost ideally suited could prove to be an interesting direction of research in the future. As an alternative the author suggests the equally interesting

Light Coding single camera approach, which might be better suited still for real time processing without requiring major hardware investments. Global methods such as Mutual Information and Graph Cuts algorithms were not implemented and remain a possibility, but the existing research suggests they are more suited for extremely precise slow computations.

Bibliography

[1] Manske, M.: Randabschattung Mikroskop Kamera. Wikipedia, the free encyclopedia, May 2007, [Online; accessed April 27, 2013]. Dostupné z: `https://commons.wikimedia.org/w/index.php?curid=23034764`

[2] S. Kirkpatrick, M. P. V., C. D. Gelatt: Optimization by Simulated Annealing. *Science*, ročník 220, č. 4598, 1983: s. 671–680, ISSN 00368075, 10959203. Dostupné z: `http://www.jstor.org/stable/1690046`

[3] Compañ Rosique, P.; Satorre Cuerda, R.; Rizo Aldeguer, R.; aj.: Disparity estimation in stereoscopic vision by simulated annealing. -, 2003.

[4] Rizo; Molina, R.: Improving Depth Estimation using Colour Information in Stereo Vision. In *VISUALIZATION, IMAGING, AND IMAGE PROCESSING*, Benidorm, Spain, Září 2005, s. 520–525.

[5] Papadimitriou, C. H.; Steiglitz, K.: *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[6] Kolmogorov, V.; Zabih, R.: Computing visual correspondence with occlusions using graph cuts. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, ročník 2, 2001, s. 508–515 vol.2, doi:10.1109/ICCV.2001.937668.

[7] Boykov, Y.; Veksler, O.; Zabih, R.: Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 23, č. 11, 2001: s. 1222–1239.

[8] GHANDI, M.; GHANBARI, M.: THE H. 264/AVC VIDEO CODING STANDARD FOR THE NEXT GENERATION MULTIMEDIA COMMUNICATION. 2004.

[9] Zabih, R.; Woodfill, J.: Non-parametric local transforms for computing visual correspondence. In *Computer Vision—ECCV'94*, Springer, 1994, s. 151–158.

[10] Banks, J.; Corke, P.: Quantitative evaluation of matching methods and validity measures for stereo vision. *The International Journal of Robotics Research*, ročník 20, č. 7, 2001: s. 512–532.

[11] Demetz, O.; Hafner, D.; Weickert, J.: The Complete Rank Transform: A Tool for Accurate and Morphologically Invariant Matching of Structures. In *BMVC*, 2013, s. –.

[12] Hannah, M. J.: Computer matching of areas in stereo images. Technická zpráva, DTIC Document, 1974.

[13] Barnea, D. I.; Silverman, H. F.: A Class of Algorithms for Fast Digital Image Registration. *IEEE Transactions on Computers*, ročník C-21, č. 2, Feb 1972: s. 179–186, ISSN 0018-9340, doi:10.1109/TC.1972.5008923.

[14] Egnal, G.: Mutual information as a stereo correspondence measure. *Technical Reports (CIS)*, 2000: str. 113.

[15] Schechner, Y. Y.; Kiryati, N.: Depth from defocus vs. stereo: How different really are they? *International Journal of Computer Vision*, ročník 39, č. 2, 2000: s. 141–162.

[16] Ng, R.; Levoy, M.; Brédif, M.; aj.: Light field photography with a hand-held plenoptic camera. *Computer Science Technical Report CSTR 2.11 1-11.*, 2005.

[17] Tao, M.; Hadap, S.; Malik, J.; aj.: Depth from combining defocus and correspondence using light-field cameras. In *Proceedings of the IEEE International Conference on Computer Vision*, 2013, s. 673–680.

[18] Albitar, C.; Graebling, P.; Doignon, C.: Robust Structured Light Coding for 3D Reconstruction. In *2007 IEEE 11th International Conference on Computer Vision*, Oct 2007, ISSN 1550-5499, s. 1–6, doi: 10.1109/ICCV.2007.4408982.

[19] Poynton, C. A.: Rehabilitation of gamma. In *Photonics West '98 Electronic Imaging*, International Society for Optics and Photonics, 1998, s. 232–249.

[20] Ttofis, C.; Theocharides, T.: Hardware design considerations for edge-accelerated stereo correspondence algorithms. *VLSI Design*, ročník 2012, 2012: str. 4.

[21] Canny, J.: A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník PAMI-8, č. 6, Nov 1986: s. 679–698, ISSN 0162-8828, doi:10.1109/TPAMI.1986.4767851.

[22] Shrivakshan, G.; Chandrasekar, C.: A comparison of various edge detection techniques used in image processing. -, 2012.

[23] Podlozhnyuk, V.: Image convolution with CUDA. 2007.

[24] Beucher, S.; aj.: The watershed transformation applied to image segmentation. *Scanning Microscopy International*, 1992.

[25] Finlayson, G. D.; Drew, M. S.; Lu, C.: Intrinsic images by entropy minimization. In *Computer Vision-ECCV 2004*, Springer, 2004, s. 582–595.

[26] Gauss, C.: *General Investigations of Curved Surfaces of 1827 And 1825; Tr. With Notes And a Bibliography by James Caddall Morehead And Adam Miller Hiltebeitel.* Scholarly Publishing Office, University of Michigan Library, 1902, ISBN 9781418163877. Dostupné z: `https://books.google.cz/books?id=_Ma-PAAACAAJ`

[27] Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, ACM, 1968, s. 517–524.

[28] Hartley, R.; Zisserman, A.: *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[29] Iocchi, L.: Stereo Vision: Triangulation. April 1998. Dostupné z: `http://www.dis.uniroma1.it/~iocchi/stereo/triang.html`

[30] Malacara-Hernández, D.; Malacara-Hernández, Z.: *Handbook of optical design.* CRC Press, 2013.

[31] Catrysse, P. B.; Liu, X.; El Gamal, A.: QE reduction due to pixel vignetting in CMOS image sensors. In *Electronic Imaging*, International Society for Optics and Photonics, 2000, s. 420–430.

[32] Chung, S.-W.; Kim, B.-K.; Song, W.-J.: Detecting and eliminating chromatic aberration in digital images. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov 2009, ISSN 1522-4880, s. 3905–3908, doi:10.1109/ICIP.2009.5413971.

[33] Sadeghipoor, Z.; Lu, Y. M.; Mendez, E.; aj.: Multiscale guided deblurring: Chromatic aberration correction in color and near-infrared imaging. In *Signal Processing Conference (EUSIPCO), 2015 23rd European*, Aug 2015, s. 2336–2340, doi:10.1109/EUSIPCO.2015.7362802.

[34] Haeusler, R.; Klette, R.: *Pattern Recognition: 32nd DAGM Symposium, Darmstadt, Germany, September 22-24, 2010. Proceedings*, kapitola Benchmarking Stereo Data (Not the Matching Algorithms). Berlin,

Heidelberg: Springer Berlin Heidelberg, 2010, ISBN 978-3-642-15986-2, s. 383–392, doi:10.1007/978-3-642-15986-2_39. Dostupné z: http://dx.doi.org/10.1007/978-3-642-15986-2_39

[35] Hough, P. V. C.: Machine Analysis Of Bubble Chamber Pictures. In *Proceedings, 2nd International Conference on High-Energy Accelerators and Instrumentation, HEACC 1959*, ročník C590914, 1959, s. 554–558. Dostupné z: http://inspirehep.net/record/919922/files/HEACC59_598-602.pdf

[36] Zhang, Z.: A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 22, č. 11, 2000: s. 1330–1334.

[37] Hartley, R. I.: Theory and practice of projective rectification. *International Journal of Computer Vision*, ročník 35, č. 2, 1999: s. 115–127.

[38] Weber, M.; Humenberger, M.; Kubinger, W.: A very fast census-based stereo matching implementation on a graphics processing unit. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, Sept 2009, s. 786–793, doi:10.1109/ICCVW.2009.5457622.

[39] Pantilie, C. D.; Nedevschi, S.: Optimizing the Census Transform on CUDA enabled GPUs. In *Intelligent Computer Communication and Processing (ICCP), 2012 IEEE International Conference on*, Aug 2012, s. 201–207, doi:10.1109/ICCP.2012.6356186.

[40] Vitor, G. B.; Körbes, A.; de Alencar Lotufo, R.; aj.: Analysis of a step-based watershed algorithm using CUDA. *Nature-Inspired Computing Design, Development, and Applications*, 2012: str. 321.

[41] Fulkerson, B.; Soatto, S.: Really quick shift: Image segmentation on a GPU. In *European Conference on Computer Vision*, Springer, 2010, s. 350–358.

[42] Scharstein, D.; Szeliski, R.: A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, ročník 47, č. 1, 2002: s. 7–42, ISSN 1573-1405, doi:10.1023/A:1014573219977. Dostupné z: http://dx.doi.org/10.1023/A:1014573219977

Appendix A

Glossary

GUI Graphical user interface

CV Computer Vision

FOV Field-Of-View

SAD Sum of Absolute Differences

SSD Sum of Squared Differences

SIMD Single Instruction Multiple Data

CRT Complete Rank Transform

NCC Normalized Cross Correlation

ZNCC Zero-Mean Normalized Cross Correlation

LR Left-Right

RL Right-Left

FPS Frames per Second

DFT Discrete Fourier Transform

MI Mutual Information

CA Chromatic Aberration

CMOS Complementary Metal–Oxide–Semiconductor

DFF Depth From Focus

STL Standart Template Library (C++)

Appendix B

Enclosed CD contents

- `applications` ..... Implemented applications
  - `correspondence` ..... The main correspondence application source
  - `readme.txt` ..... Readme on building and running the application
  - `support` ..... Supporting camera related applications
- `text` ..... Thesis source
  - `DP_Blazicek_Jan_2016.pdf` ..... Thesis in a PDF format
  - `DP_Blazicek_Jan_2016.tex` ..... Thesis LaTeX source