Universiteit Leiden

Opleiding Informatica

Finding correspondences in stereo image pairs using an adaptive window comparison algorithm

Name: P. Moerkerk
Date: 13/08/2014
supervisor: Dr. M.S. Lew
2nd reader: Dr. E.M. Bakker

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In context of the Bachelor's Project as part of the Bachelor Computer Science curriculum of the Leiden University this research in Computer Vision has been done. It has its focus on a correspondence finding method in stereo images using an adaptive window algorithm. The proposed algorithm B_n uses a square window support area and dynamically selects part of best matching $n\%$ of pixels for comparison aggregation. The performance of B_n is evaluated over a broad range of n and window sizes and related to SSD using a subset of the Middlebury stereo data set. The influence on feature point selection and performance in disparity discontinuity regions are also researched. The results show that B_n does (on average) not perform significantly better than SSD using the proposed feature point selection method. Another conclusion is that the performance strongly dependent is on type of scene. In disparity discontinuous regions B_n shows promising results with an possible average gain of 7% in correct prediction fraction. The conclusions are substantiated using an in-depth research of behaviour with attempted explanation using basic theory and several examples.

Contents

1 Introduction **7**
1.1 Related work 7

2 Theory **8**
2.1 Epipolar geometry and image rectification 8
2.2 Disparity and depth 9
2.3 Correspondence problem 9
2.3.1 Pixel matching metric 11
2.3.2 Reference sum-of-square distances algorithm (SSD) 11
2.3.3 Proposed partially counting (best of) window comparison function (B_n) 11
2.4 Problem areas 11
2.4.1 Plain areas and geometric distortion 11
2.4.2 Feature point detection 12
2.4.3 Regions of occlusion and disparity discontinuous regions 13

3 Implementation **16**
3.1 Framework 16
3.1.1 Input / output 16
3.2 Performing disparity search 16
3.2.1 Search region 16
3.2.2 Feature point selection 17
3.2.3 Window sizes and values of n for B_n 17
3.3 Data set 17
3.4 Evaluation methods 17
3.4.1 Input domains for algorithms 17
3.4.2 Missing ground truth data 19
3.4.3 Correct disparity classification fraction 19
3.4.4 Average deviation from true disparity 20
3.4.5 Comparing the algorithms over the whole data set 20

4 Results **21**
4.1 Evaluation of B_n without feature point detection 21
4.2 Evaluation of B_n (when using feature point detection) 21
4.2.1 Overall performance 21
4.3 Interpretation 22
4.3.1 Overall performance gain 22
4.3.2 In depth research of performance on single scenes 22
4.3.3 In depth: pro's and cons of B_n 24
4.4 In-depth analysis on disparity discontinuous regions 26
4.5 Comparison of optimal performance B_n on full range vs. disparity discontinuous area 27

5 Conclusion and Discussion **28**
5.1 Selection of disparity discontinuities 28
5.2 Optimizing B_n 28
5.3 Unevaluated areas of interest 28

6 Final word **29**

A Table of images in dataset **31**

B Code listing of C++ implementation of B_n algorithm **33**

C Code listing of C++ implementation of feature point selection function **34**

D Code listing of disparity discontinuity region mask creation **35**

E Results of B_n when feature point detection is included **36**
E.1 Correct classification fraction per sample pair 36
E.2 Average error in predicted disparity per sample pair 40

F Results of B_n when no feature point detection is done **45**

1 Introduction

With increasing number of applications these days the need to derive 3D models of real scenes still grows. Think about the development of autonomous driving vehicles like Google Self-Driving Car Project [16], autonomous flying drones like HEXO+ or the rise in popularity of 3D movies and related display environments like 3D cinema and television. As virtual reality makes a comeback with successful crowd funding project Oculus Rift [15] the need to acquire 3D content also increases.
There are many different ways to capture 3D scenes, ranging from active methods using for example moving light sources, time-of-flight lasers, structured light or ultrasound to passive methods using non-intrusive sensors like CCDs to capture the scene. With the decreasing price of camera image sensors and (mobile) computing power it is inevitable 3D reconstruction using images (in sequence or parallel) will be used in most (if not all) of the previously mentioned applications.
As trivial it seems to humans to perceive depth, as tricky and difficult this is done with a machine. In the early days of 3D reconstruction most of the focus was on deriving depth from stereo image pairs of static scenes. For this application two images from a scene are made from slightly different positions, similar to human perception. The pair of images is then processed using a comparison method to derive depth information.
More recent studies have also been focusing towards reconstruction from a sequence of images from a moving camera through a (partly static) scene [6]. As computational power now permits it and the demand is getting higher, most recent research is deriving 3D information in real time from sequences of (noisy) images from dynamic changing environments like a car moving through traffic. This is a challenging task.
However in most image-based methods the crucial part of the reconstruction process is to determine point correspondences between images. This project focuses on that challenge.

1.1 Related work

In "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms" Scharstein and Szeliski [1] present an excellent overview of methods for 2-frame stereo correspondence algorithms. Azaros, Sirakoulis and Gasteratos augment their work in 2008 by giving a review with more emphasis on hardware implementation methods in [2].
Both papers show that research in this field is done since at least before 1980. The number of papers presented in this area are therefore numerous. An early example is [3] where Barnard and Thompson describe an iterative way to match candidate points between two images.
Our research proposes a dynamically changing comparison window. Related research has been done, for example in 1994 when Kanade and Okutomi present their research related to adaptive windows [5]. Their proposed algorithm dynamically selects the window size based on intensity and disparity. In [7] Lotti and Giroudon present a method (suggested by Kanade) which also dynamically adapts window size near disparity discontinuities and near occlusions. They also change shape of the window (rectangular) but conclude that precise information about the location of discontinuities needs to be known beforehand.
Methods that calculate disparities using multiple window sizes and select afterwards are also researched in [11] and [12]. In [8] Yoon and Kweon describe a method using adaptive weighting of the window. They calculate the weights of the pixels in the support region based on colour similarity and geometric proximity. More recently Wang and Fu also used rectangular adaptive windows in combination with a new similarity measurement function to improve upon SSD [9]. An extensive (comparative) study on adaptive support windows is performed by Hosni, Bleyer and Gelautz [10].

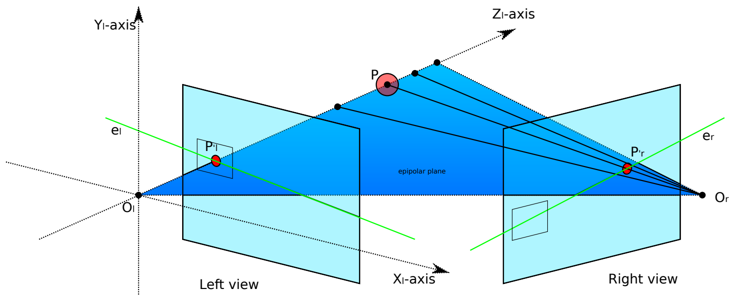


Figure 1: Epipolar geometry. Projections of a point P in both image planes lay on the epipolar plane.

2 Theory

2.1 Epipolar geometry and image rectification

In point correspondence algorithms, two images of the same scene of slightly different perspective are given and the goal is to find equivalent points. These images can be described as projection planes in 3D space between the scene and the centre of projection (O) of a camera. The intersection of lines from points in the 3D space and the origin with this plane create the image. This is a somewhat simplified view, as real cameras have a point-symmetric inverted projection behind the centre of projection and do not have a perfect projection (as their lenses do not behave like a perfect pinhole model). But given that the images are corrected for lens distortion, the projection can be simplified by this model. In this model if we take one point in one projected image with centre of projection O_L, this point, say P_L can only originate from somewhere on a line $O_{(L)} - P_L$ in the scene (see Figure 1). When any point on this line is projected onto the another plane with centre of projection O_R, these points all lay on the a line in the other image, called he epipolar line (e_r). This fact can be used when searching for a point: When searching for a projected point P_L in one image, we have to search the corresponding point (P_R) on (only) a line in the other image. Symmetrically a line can be defined in the first image by projecting back the epipolar line from the second image. One can see this creates a plane, called the epipolar plane which contains all possible points which must have a projection on the two epipolar lines on both images.
To optimise searching, the cameras (and/or images) can be aligned such that the epipolar lines are horizontal and each vertically aligned [18]. This process is called rectification and results in co-planar images. The net

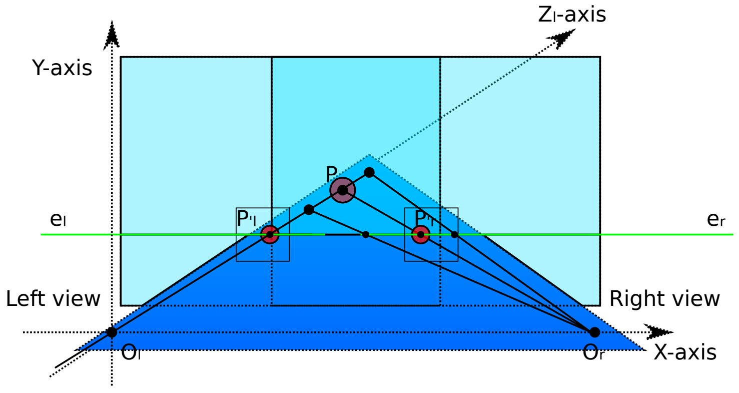


Figure 2: Rectified images. Both epipolar lines coincide which simplifies searching. Note that the epipolar plane actually is (near) perpendicular to the image planes.

result after rectification is that points on each horizontal (scan) line of each image correspond with points on the same horizontal line of the other image. For this research the images are assumed to be rectified and aligned.

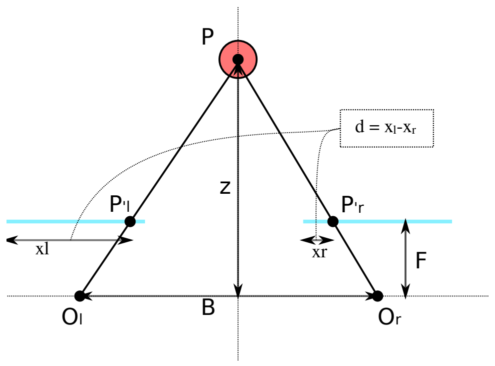


Figure 3: Camera parameters focal length (F) and baseline (B) shown in relation to camera origins. P is projected onto the two blue image planes. The disparity d is derived from the distances of the projections from the edge of the projection planes: $d = x_l - x_r x_l$

2.2 Disparity and depth

When humans perceive the world, since both eyes are a few centimetres apart, objects appear shifted horizontally when comparing left and right view. That is, when the focus is at (near) infinity. The amount of displacement of the projection of one feature point in one image with respect to projection of that point in the other image is called disparity. This disparity d is inversely proportional to the distance z to the feature point [4]:

$$d = BF \cdot \frac{1}{z}$$

Figure 3 gives a graphical representation. Here B and F are camera parameters baseline and focal distance. This means objects nearby have greater disparity value than objects further away.
If one would create a map of disparity for each point in the source image and translate these into for example grey scale values the result can look like something like figure 4. Here light values indicate greater disparity (and thus greater proximity to the camera).
In most real life situations there are areas visible in one image which cannot be seen by the other camera because something (an object in closer proximity; which has a greater disparity) is obstructing the view. These area's are called *regions of occlusion*.

2.3 Correspondence problem

This project focuses on localised search algorithms. This means that for each interest point in the reference image a corresponding point is sought in the target image. Global information, for example about neighbouring disparity or derived information about object geometry is not used.
A general correspondence method can be described as having at least three parts [1]:

1. Evaluation of the matching metric
2. Aggregation of the metrics
3. Disparity deduction

To *evaluate* the matching metric, a function is defined to give an estimation of similarity (or dissimilarity) of a single pixel comparison. Most often this is taken to be either the absolute difference (AD), squared distance (SD) or normalised correlation in colour or intensity values. This function can be seen as a cost function.
For comparing a single point, it is inevitable include the surrounding area of the pixel in the comparison, as single pixels do not hold enough information for finding a match (due to for example noise, resolution, pixel depth). Thus comparison of points include some surrounding pixels called the *support region*. This leads to the need for having an aggregate function to combine the comparison of pixels in this support region.
This research has its focus on the influence of this support region. A simple and straightforward region is a square window around the pixel. A general used *aggregate function* is to simply take the sum of costs for all pixels in the window (also known as the SSD function). A more elaborate way would be to use a Gaussian

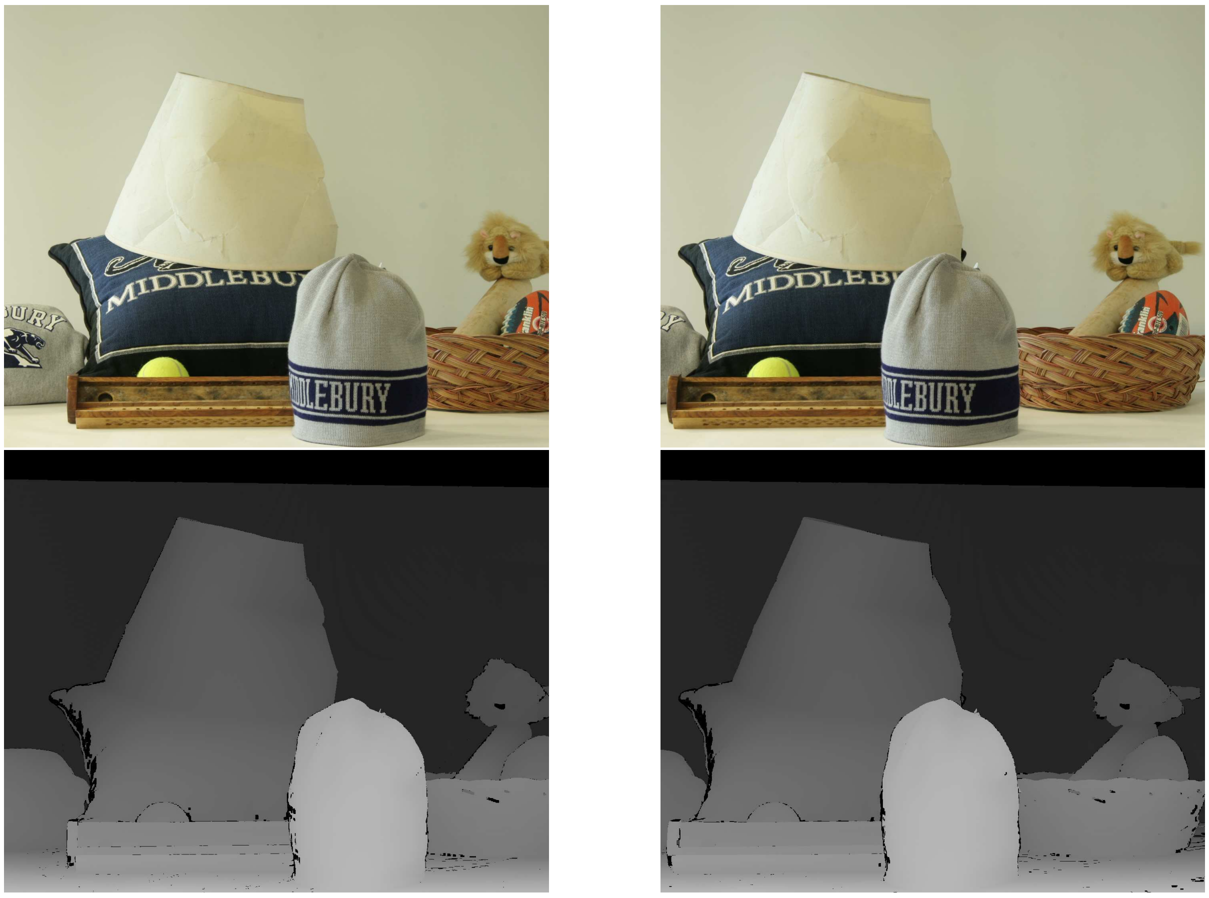


Figure 4: Example of image pair and ground truth disparity maps. The upper two images show two views of the MIDD2 scene and the lower two the disparity maps. An example of an occluded area is visible on the cushion, where the letter ‘U‘ in the word is fully visible in the left image and only partly in the right.

kernel to weigh each pixel before summing (based on distance from the centre point). This makes it possible to put emphasis on matching pixels near the centre (feature) point.
The last step, *determining the disparity* is as simple as evaluating the window comparison function on a range of possible disparities and select the point at the best of all evaluated disparities. For this the algorithm calculates the cost for every single disparity value and selects the disparity with lowest cost.

2.3.1 Pixel matching metric

For the algorithms used the matching metric for two pixels P,Q is a cost function given by the root of the sum of squared distances of the three colour components (red R, green G and blue B):

$$d_{P,Q} = \sqrt{(P_R - Q_R)^2 + (P_G - Q_G)^2 + (P_B - Q_B)^2}$$

where P_R,P_G,P_B mean the red, green and blue component values of pixel P, and Q_R,Q_G,$Q_B$likewise. This metric can be seen as a cost function: an increasing value means less similarity.

2.3.2 Reference sum-of-square distances algorithm (SSD)

As mentioned before, this an aggregate function most used as standard reference. For each pair of points P and Q, where P in the reference image and Q in the target (search) image, the matching cost function C is given as the sum of squared distances (like in Section 2.3.1) of pixels in a square window of $N \times N$ surrounding the points:

$$C_{SSD} = \sum_{i,j \in [\lfloor -\frac{1}{2}N \rfloor, \lfloor \frac{1}{2}N \rfloor]} \left(d_{P+(i,j)\,,\,Q+(i,j)}\right)^2$$

This function is also used as the reference function in evaluating the researched B_n function.

2.3.3 Proposed partially counting (best of) window comparison function (B_n)

The proposed comparison function B_n is based on selecting a subset of pixels in the support region to count. This subset is calculated to be $n\%$ of best matching pixels. The idea is to increase correctness in areas of disparity jumps and occlusions. The cost function $C_{B_n}$for a window comparison for B_n is given as:

$$C_{B_n} = \sum_{i,j \in [\lfloor -\frac{1}{2}N \rfloor, \lfloor \frac{1}{2}N \rfloor]} \left(selected \cdot d_{P+(i,j)\,,\,Q+(i,j)}\right)^2$$

where $selected$ is defined to be 1 if $d_{P+(i,j)\,,\,Q+(i,j)}$ is part of $n\%$ of pixels with lowest cost per pixel (d) and $selected = 0$ otherwise. In other words: B_n takes only the $n\%$ of all best matching pixels into account for comparison. Observe that with $n = 100\%$ the function is equal to the SSD metric.

2.4 Problem areas

There are regions in images which can be defined as hard to match. A few of these type of regions are discussed next.

2.4.1 Plain areas and geometric distortion

Areas in the picture that do not hold a great amount of distinctive information can not be easily matched correctly, if at all. For example the window around a point on a non-textured (plain) surface with little color gradient will match with all windows near that point (when the window fits within this plain area). These points will only possibly match correct if the window size is big enough that details make it distinguishable from neighbouring windows. This might suggest that increasing windows size is always better. However when the real surface is slanted (not parallel with the image plane), these features that finally match the over-sized window will lay far from the centre and most likely give a skewed disparity. Additionally, apart from the obvious increasing computational costs associated with larger window sizes, another problem with increasing window size is that it increases the chance of including a region of occlusion or disparity discontinuity within the window (see also Section 2.4.3). To summarise areas with not a lot amount of information are hard to match and there is a trade-off between small window size and large window size.

One way to overcome the first problem (information lacking areas), is to only focus on information rich areas. Intuitively it seems there must be something specific, something unique in the window, what can be searched for. One might calculate the entropy of a window, and use this value to determine the amount of information. Another way to define a window that is distinguishable is to compare it with its neighbouring windows in the same (source) image. If the used comparison function does not give a confident discrimination of this window and its neighbours, it will most likely also not in the target image. (Or produce ambiguity whether to match the neighbouring window)
A more simple approach is to define "information rich" points. We define points that are interesting to use for comparison feature points or interest points. There is a slight nuance between a interesting region or interesting point. An area with "information" does not have to contain specific interest points as the information might be spread out over the window, but an area with some salient points definitely has information. People often think about corners and edges when finding corresponding points. This leads to the basic idea to use an edge detection algorithm to define interest points.

2.4.2 Feature point detection

For feature point detection a (variation on a) Sobel operator is used. In essence the Sobel operator is a convolution that works on an intensity image in two directions. It is assumed that the image is a discretisation of an intensity signal. The operator calculates the gradient magnitude of this signal in both horizontal and vertical direction using two convolutions and combine these. The result is a gradient magnitude map: where the intensity changes more quickly, the value is higher. To use this in practice two 5×5 convolution matrices (Figure 5) are used, one for horizontal orientation and one for vertical orientation.

$$S_v = \begin{bmatrix} 2 & 2 & 4 & 2 & 2 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -2 & -1 & -1 \\ -1 & -2 & -4 & -2 & -2 \end{bmatrix} \qquad S_h = \begin{bmatrix} 2 & 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 & -2 \\ 4 & 2 & 0 & -2 & -4 \\ 2 & 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 & -2 \end{bmatrix}$$

Figure 5: Sobel 5×5 convolution matrices. One horizontal (S_h) and one vertical (S_v) operator.

For this application the convolutions are done on each colour intensity channel (R, G, B) separately, which results in six gradient intensity maps (with values ranging from -28×255 to $+28 \times 255$):

$$\begin{aligned} S_{Rh} &= S_h \ast R \\ S_{Gh} &= S_h \ast G \\ S_{Bh} &= S_h \ast B \\ S_{Rv} &= S_v \ast R \\ S_{Gv} &= S_v \ast G \\ S_{Bv} &= S_v \ast B \end{aligned}$$

A value θ is chosen to define a certain threshold for which the absolute gradient magnitude is high enough to to define the point as an interest point. The results for each channel are combined using a binary OR operation to derive a map of feature points:

$$IsFeaturePoint \vdash (|S_{Rh}| > \theta \vee |S_{Rv}| > \theta \vee |S_{Gh}| > \theta \vee |S_{Gv}| > \theta \vee |S_{Bh}| > \theta \vee |S_{Bv}| > \theta)$$

Thus if the (absolute gradient) in only one channel in only one direction is greater than the threshold value, this marks the point as interest point.
Now to reflect back on the original idea that interest points must have something specific, some unique information. This does not have to mean there is a sharp gradient change. One could imagine a region with a very gradual change in intensity still having some unique pattern to match on. These points will be missed as interest points. The opposite could also happen. A perfect horizontal line as separation between two plain (intensity different) areas will be seen as a line of interest points, but in fact since the line expands horizontally the area is similar in horizontal direction and precise horizontal matching is difficult (if not impossible) along this line, similar to matching in a plain area. Additional research can be done to evaluate whether disregarding the vertical Sobel operator does improve feature point selection.

2.4.3 Regions of occlusion and disparity discontinuous regions

When part of an image is a projection of a geometrically smooth surface the resulting region has a fluent change in disparity. These are called *disparity continuous* areas. The opposite is where there is a sudden change in depth (and hence disparity), for example at the border of objects. We hypothesise that the proposed B_n method will not perform better on disparity continuous areas when compared to the reference SSD method, since B_n takes a subset of pixels to take into account, thus reducing information density. This might result in less optimal matching when applied to (disparity)continuous areas.
On the other hand B_n might perform better in areas where the disparity is not continuous, for example where there is a sudden change in distance at the edge of an object. Figure 6 illustrates this hypothesis. This figure shows a fragment of the Bowling scene, in which the left and right images are compared with each-other. The centres of the windows (dashed lines) are aligned and should match. However the contents only partly match as part of the window is occluded. In this particular case B_n (with n around 60%) should perform better than SSD as then only the perfect matching (in the figure marked as green) part will count and should evaluate to a low cost, making it the most likely match.

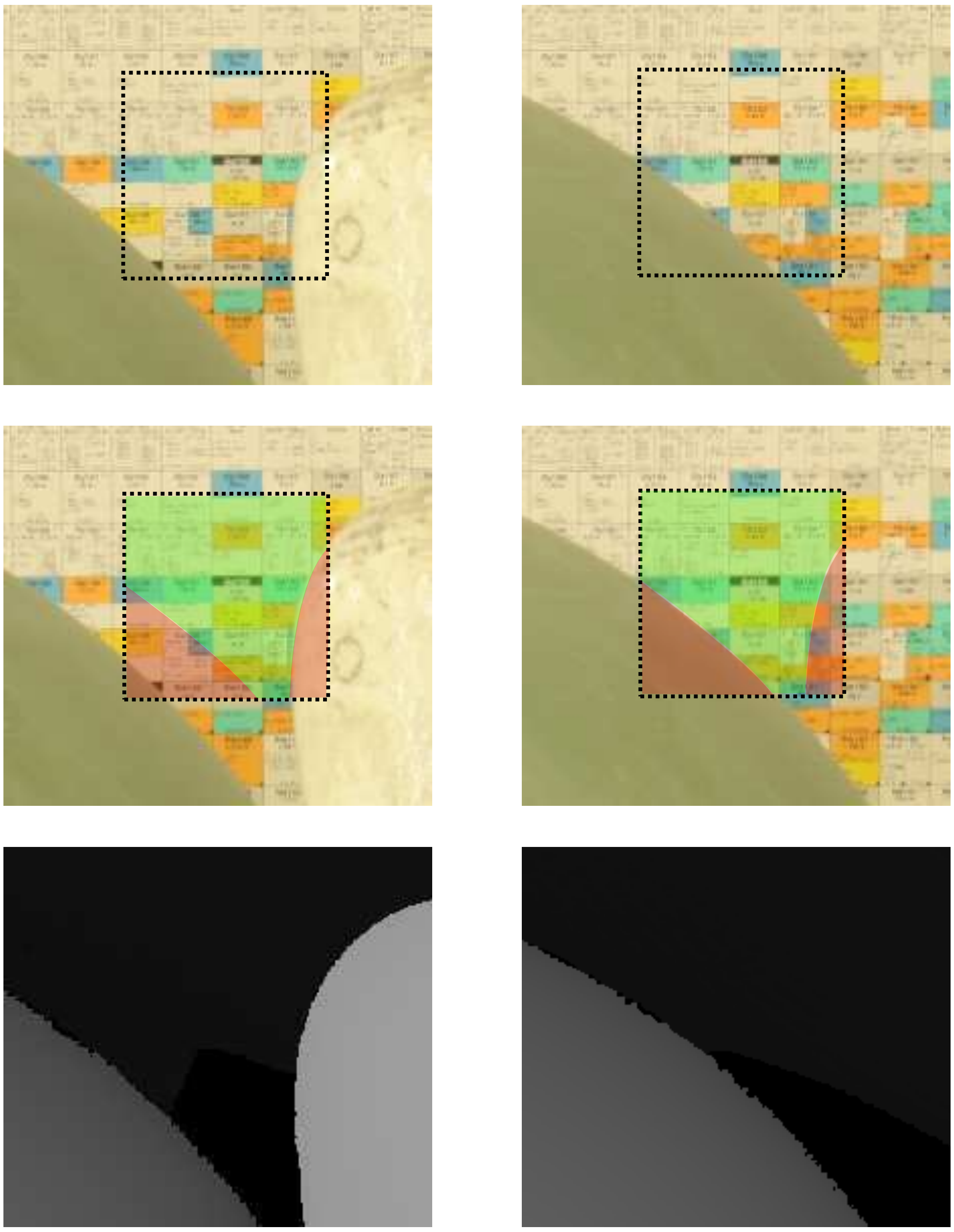


Figure 6: Example of non continuous disparity. Left and right images next to each other. Upper set: image fragment and comparison window; middle set: annotated and lower set: true disparity. The windows have partly perfect match (green area) and partly no match (red area).

One hard problem for any algorithm is that of occluded regions. Feature point selection (such as suggested in Section 2.4.2) will not discriminate on occluded regions. If a feature point is chosen within a fully occluded region and it thus is visible in one image but not in the other, any of the proposed search algorithms will fail to derive a correct disparity, as those will only match on visible regions (and hence it will not be possible to find a match in this case). One way to try to cope with these regions is to calculate for each found match a confidence metric, for example the calculated difference, the significance of the differences with respect to other differences for the same window or found disparity related to that of neighbouring interest points. Using this extra information a kind of "failed match" (or low confidence) criteria could be applied to discard very uncertain disparity predictions. As this is not done in the framework, it will inevitably lead to poor results in those regions.

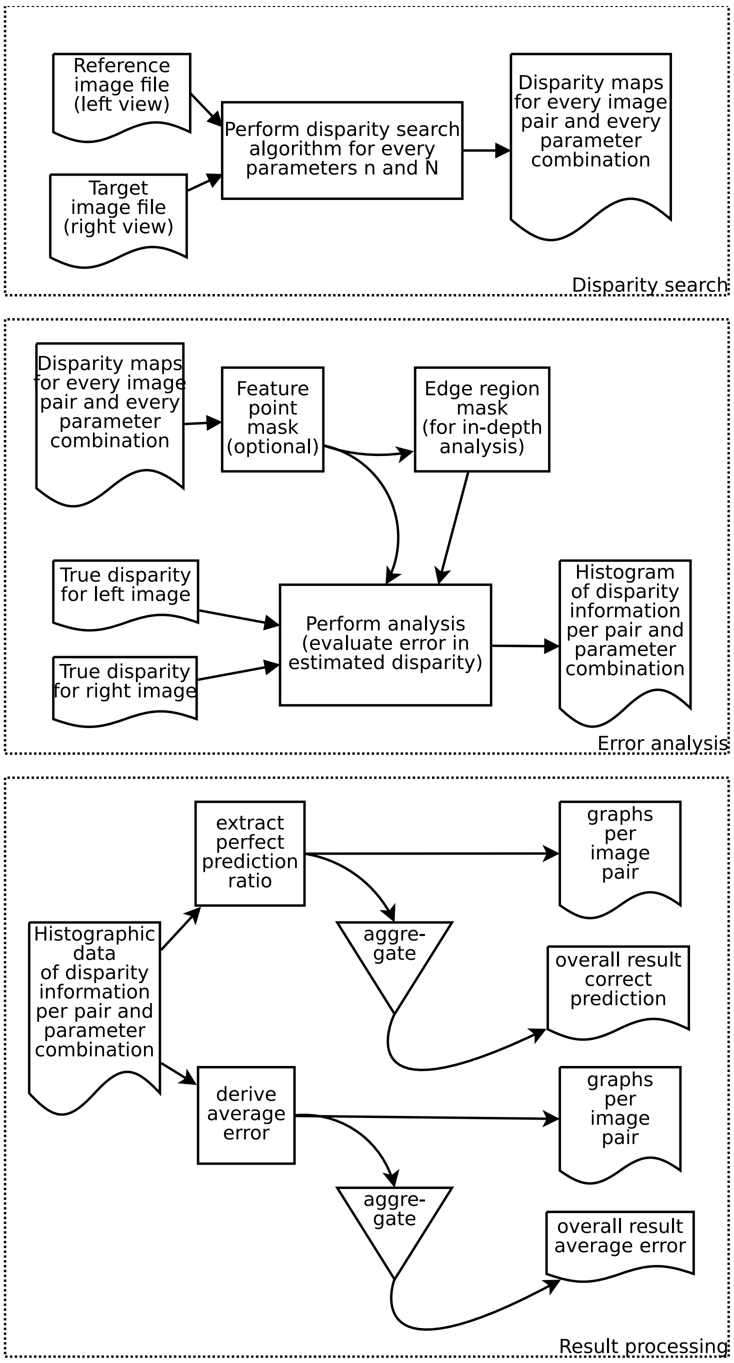

Reference image file (left view)
Target image file (right view)
Perform disparity search algorithm for every parameters n and N
Disparity maps for every image pair and every parameter combination
Disparity search
Disparity maps for every image pair and every parameter combination
Feature point mask (optional)
Edge region mask (for in-depth analysis)
True disparity for left image
True disparity for right image
Perform analysis (evaluate error in estimated disparity)
Histogram of disparity information per pair and parameter combination
Error analysis
Histographic data of disparity information per pair and parameter combination
extract perfect prediction ratio
graphs per image pair
aggre-gate
overall result correct prediction
derive average error
graphs per image pair
aggre-gate
overall result average error
Result processing

Figure 7: Data flow

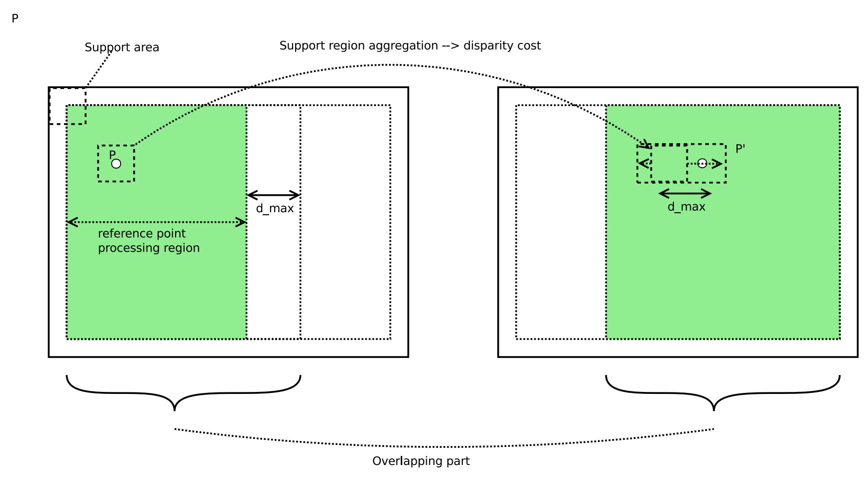


Figure 8: Defining search region and performing search

3 Implementation

3.1 Framework

A basic framework to evaluate the algorithms is written in C++. Apart from the C standard libraries it is using the SDL2 library for input/output of image files and graphics and a subset of STL to provide some support for standard data structures. The implementation is made with readability and "tweakability" in mind. It performs the proposed algorithms in a very naive, straightforward and unoptimised way and is intended to do analysis of the point searching algorithms only. It is by no means optimised for performance in any way.
The functional layout basically consists of two classes which define two modes of operation. First one that performs the algorithms, the other which does very basic analysis of the resulting data by comparing with reference (ground truth) data. Some shell scripts augment the process to generate plot scripts for GNUPLOT, whose output is presented in Section 4.

3.1.1 Input / output

The framework expects input data to be organised as in the reference Middlebury data set; it automatically searches, detects and reads input sets, including ground truth data. During disparity search a set of intermediate result data (the disparity maps) is written to disk as image files. The result data is processed further with a separate program invocation. This results in disparity histograms, which are processed even further to graphs using Bash shell scripts. A graphical overview of the information flow is given in Figure 7.

3.2 Performing disparity search

First it must be noted that the disparity search is done in only one way: First a region is defined in the first (left) image. For all points included in this region a single pass disparity search is done by searching in the other (right) image. No reverse search (target to source) is done and thus no correlation between both searches is done or anything like that. For each point the maximum searched disparity depth is 255, which is conform the scene / camera setup and preprocessing of the source data set.

3.2.1 Search region

For each image pair a subset of pixels are processed. To avoid difficulties that arise at pixels near borders of the images the search space is reduced to exclude a border larger than the size of the diameter of the largest support zone. In this case this is done with a big margin of 26 pixels. Furthermore, on the side where images do not overlap, the search is stopped at the distance of the maximum disparity range away from the reduced border (see Figure 8). The resulting setup is that every point searched for has a full maximum size window of pixels of support zone and searching can be done within the full possible disparity range in the other image. This is to avoid possible bias in evaluating performance.

3.2.2 Feature point selection

The results include only pixels that are defined as interest points by the feature point detection as described in Section 2.4.2. Using this method a mask is generated which is used to select these points for further processing. The threshold value θ for determining the interest points is set (after visually checking the resulting masks) at 255.

$$\theta = 255$$

Appendix C shows the source code for the feature point selection and Figure 9 gives an indication of resulting feature point masks.

3.2.3 Window sizes and values of n for B_n

The experiments are done for a broad amount of window sizes and percentages for B_n. Figure 10 shows two tables with used windows sizes and percentages. These tables also indicate the reference number used in the framework and naming of files. The windows are odd sized which puts the search feature point location in the centre and range from 5×5 to 25×25. Appendix B shows the source of the B_n algorithm.

3.3 Data set

For evaluation of the algorithm a data set with rectified images and known disparity was needed. In 2002 D. Scharstein and R. Szeliski at the Dept. of Math and Computer Science of Middlebury College, Vermont, USA, have produced stereo correspondence software and have composed a reasonably sized (about 30 items) data set with static indoor content. Using a different method (involving structured light; presented in [13]) they derived ground truth (true disparity) data which is included with the data set.
For this project a subset of their data set, the '2005' and '2006' data sets are used. Each item in the original data set consists of 7 photo views (each with a (slightly) different camera baseline, numbered 0 to 6), but only for two angles (1 and 5) disparity data is given. The corresponding image pairs are used in this research. All experiments are done on unmodified data of this data set. That is no scaling, blurring or other modification of the data is done beforehand. Appendix A contains a table of used images.

3.4 Evaluation methods

3.4.1 Input domains for algorithms

To evaluate the quality of the algorithm and get idea on what areas the algorithms perform well analysis is done on three different (input domain) pixel subsets:

1. Full pixel set
2. Feature points only
3. Feature points within regions of disparity discontinuity

The *full pixel set* includes all pixels regardless whether they are marked as feature points. This data was gathered to get a general overview of the algorithms performance early on in the project. Although these results gave a good insight into the algorithm initially, they were perceived as disappointing. That is why the *feature points only* pre-selection was included. All of the subsequent work and analysis was done using this feature point pre-selection. To get insight into the algorithms performance in true regions of disparity discontinuity an extra evaluation was done.
The analysis section will include some results from the full pixel comparison, and more in depth results of the feature point selection and regions of disparity discontinuity.
For each parameter the basic evaluation is a comparison of the predicted disparity to the reference (true) disparity: For each predicted disparity pixel the distance between the known ground truth disparity and the calculated disparity value is determined.
We define d_P^{true} the true disparity for some point P and $d_P^{predicted}$ the predicted disparity. The disparity error d_P^{error} is defined as the distance between $d_P^{true}$and $d_P^{predicted}$:

$$d_P^{error} = \left| d_P^{true} - d_P^{predicted} \right|$$

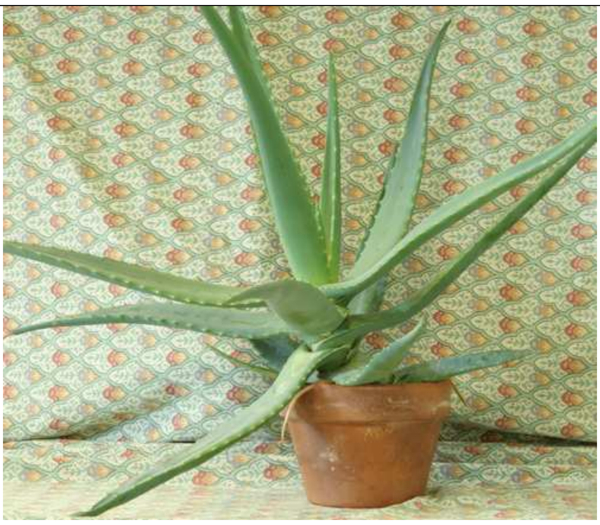
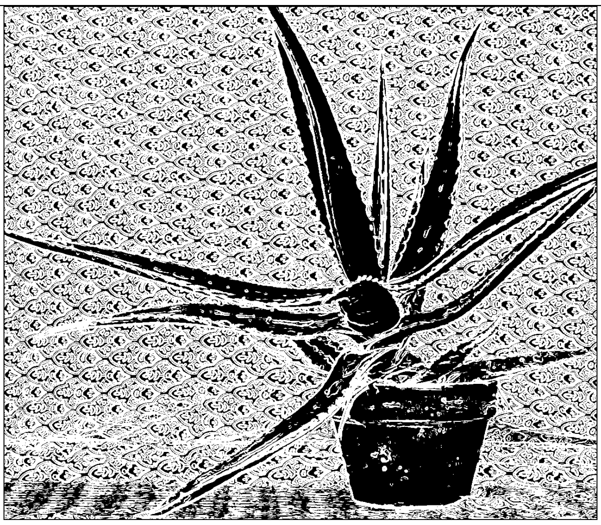
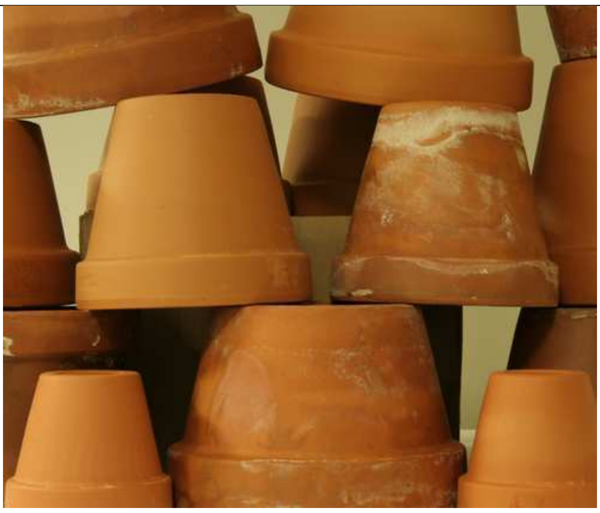
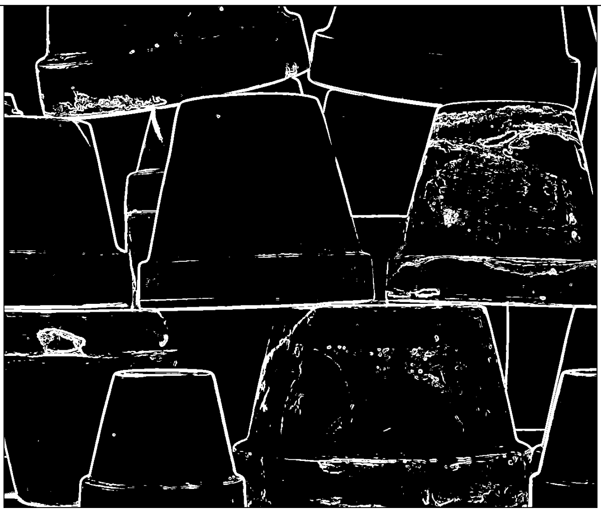
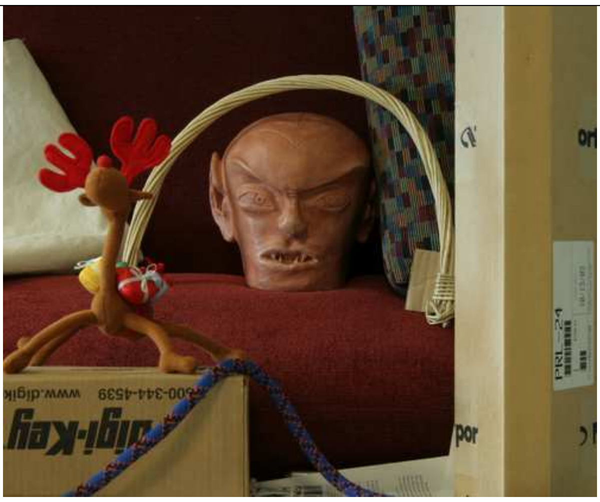
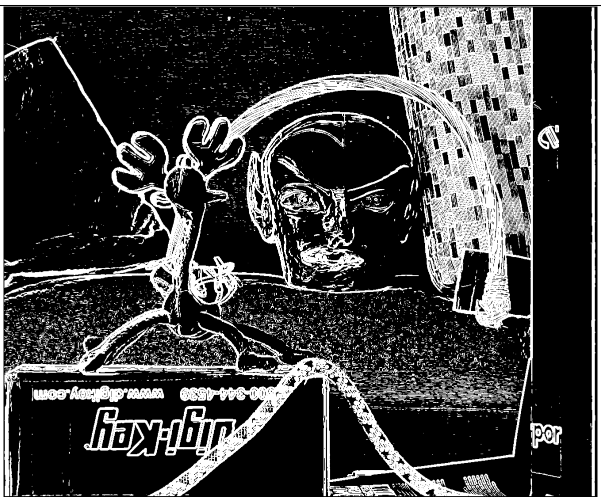
Source image	Resulting mask
Aloe	
	
Flowerpots	
	
Reindeer	
	

Figure 9: Examples of feature point masks for three items. White means included as feature point.

ref #	Window Size
0	5×5
1	7×7
2	9×9
3	11×11
4	13×13
5	15×15
6	17×17
7	19×19
8	21×21
9	23×23
10	25×25

ref #	$n\%$
0	10
1	20
2	30
3	40
4	50
5	60
6	70
7	80
8	90
$9/SSD$	100

Figure 10: Included window sizes and values used for parameter n of B_n

Two different evaluation measures are calculated: the number of correct disparity predictions and the average disparity error in predictions. This is done for all image pairs, a set of window sizes and a set of parameters for B_n (see Figure 10).

3.4.2 Missing ground truth data

In some cases the ground thruth was not known. These points were omitted from the evaluation. The amount of missing ground truth information is in order of several percent. A visual evaluation reveils the distribution over the images: Some of these pixels lay on boundaries of disparity discontinuities, more inside occluded regions and most within very dark regions. This seems to be consistent with the method used to obtain these ground truth values (using structured light).
Without further analysis it is not possible to evaluate the impact of this missing data on the algorithms evaluation. It is assumed that the amount of missing ground truth data is negligible, since it seems to be spread over different interest area's.

3.4.3 Correct disparity classification fraction

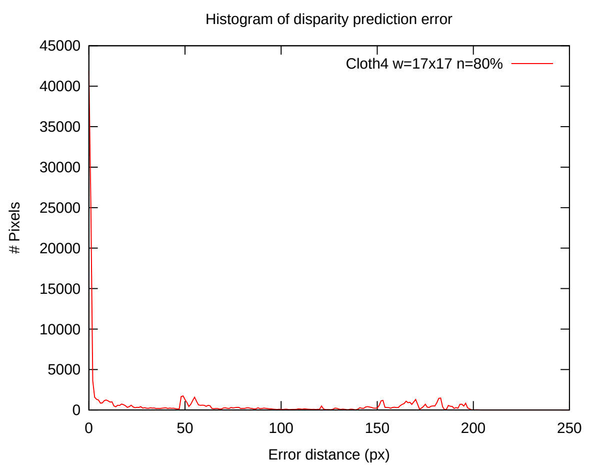
To get an rough estimation of performance measure the number of perfect classified pixels is counted. That is, the number of disparity predictions that are less then some threshold value η difference from true disparity.

$$frac_{correct} = \frac{\sum_{p \in TP} \left(d_p^{error} < \eta \right)}{\|TP\|}$$

Here TP is the set of points of which disparity is predicted and the true disparity is known.

$$TP = Search\,Region \cap Feature\,Points \cap Known\,Ground\,Truth\,Disparity$$

The challenge here is to determine what a good threshold value η is. For this purpose a few histograms of disparity error are made. The visual inspection of the disparity error histogram (the error distribution) might indicate problems and it is used to choose η.



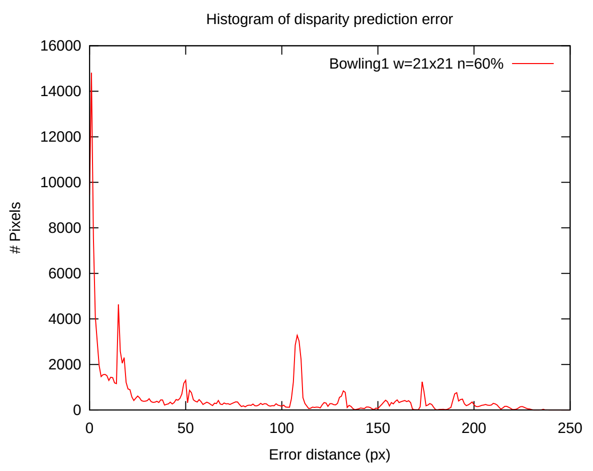


Figure 11: Histogram of deviation from true disparity. Shown is the amount of pixels as a function of offset from true deviation. The two example histograms show typical disparity prediction error distribution. Most pixels fall in range of 0 to 1. Sometimes there are other peaks which can be explained by matching on repetitive patterns.

It is expected that the histogram shows that most pixels have no error (perfect prediction) and the amount of pixels decrease with increasing deviation (error) value. A non zero peek of disparity error might indicate some bias (offset) of all disparity values. Two typical histograms are shown in Figure 11. By visually inspecting these and a lot of others the threshold for 'perfect classification' η is taken to be 3.

$$\eta = 3\text{px}$$

This translates into an error margin of $\frac{3}{255}$ or about 1.2%. This means that all disparity predictions which are off by one or two pixels also count as perfect prediction. The reason to widen the error is somewhat arbitrary. The right graph in Figure 11 shows a disparity peak at about 1 pixel, that is most predictions are off by one. It is not inconceivable that this offset is result of noise in the input data and to make the measurement more robust the threshold is chosen to compensate for possible noise in this way. In any case, it is a fair comparison measure when consistently used over the data set.

3.4.4 Average deviation from true disparity

The second method in evaluating the algorithms is to calculate the average error in disparity prediction. This is done simply by calculating the sum of all disparity errors divided by the total number of selected points

$$error_{avg} = \sum_{P \in TP} \frac{d_P^{error}}{\|TP\|}$$

Here P is again a search point and TP is the whole set of processed points.

3.4.5 Comparing the algorithms over the whole data set

Lastly the results of the analyses described in Section 3.4.4 and Section 3.4.3 are combined to produce two overall measurements of the B_n method. This is done by aggregating the results by averaging over the whole data set: The global correct classification fraction is taken as sum of all calculated fractions per item divided by number of items in the data set. Likewise the global average error in disparity prediction is calculated as the sum of average error for all items divided by the number of items. The results are shown in Section 4.2.1.

4 Results

4.1 Evaluation of B_n without feature point detection

The results of B_n per pair are shown in Appendix F. The global average overall performance is shown in Figure 12.

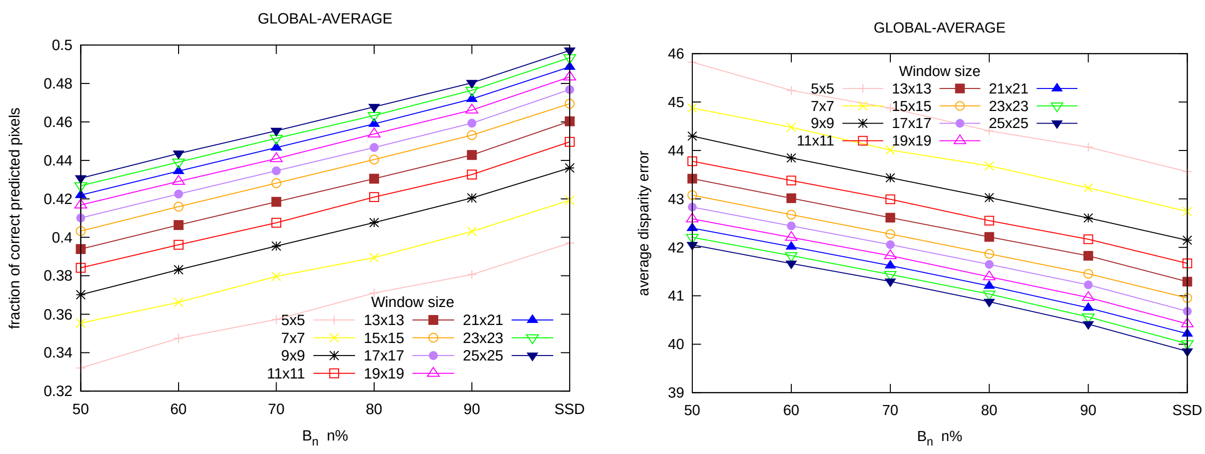


Figure 12: Overall performance: average fraction of correct pixels (left) and average error (in pixels) (right)

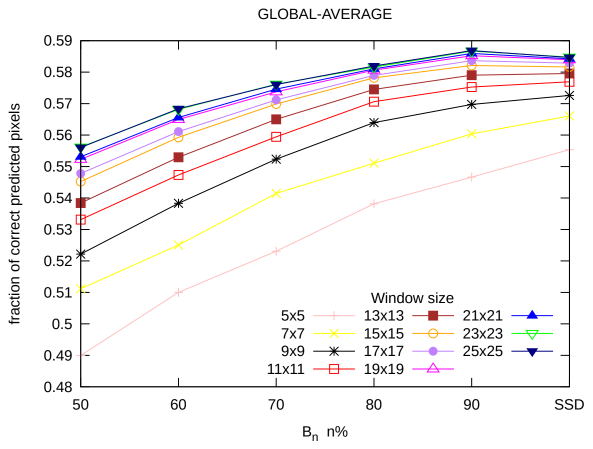
The strict rising lines in the left graph show that performance for B_n increases with rising n for all window sizes, with best performance at $n = 100$, which is equivalent with SSD. Likewise the right graph shows the average error decreases when getting closer to SSD. This shows that SSD outperforms B_n on the dataset when no feature point selection is done. The dominating factor in this situation is most likely the amount of areas without much feature points, like discribed in Section 2.4.1. In these areas every pixel counts to get a better match. Figure 9 shows that some of the images are very sparsely populated with feature points. For these images the performance of B_n with n smaller than 100% is worse than SSD. This result was not unexpected. That is why the rest of the research is based on performing B_n and SSD with feature point detection.

4.2 Evaluation of B_n (when using feature point detection)

For all image pairs B_n (when using feature point selection) is evaluated. The graphs in Appendix E show the results.

4.2.1 Overall performance

To evaluate the overall performance of the data set the average fraction of correct pixels over all items in the data set is calculated and shown in figure 13.



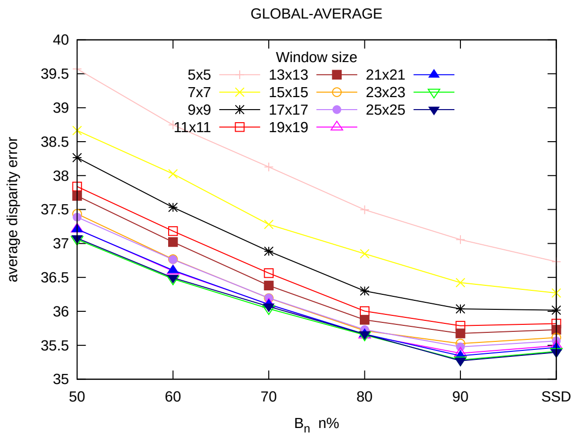


Figure 13: Overall performance: average fraction of correct pixels (left) and average error (right)

This graph shows the optimal window size for SSD is 25×25 with a correct prediction ratio of 58.5%. The minimal average error for SSD is around 35.5 pixels with the same window size. The best parameters for B_n are is the same window size of 25×25 and $n = 90$ for a correct all-round prediction ratio of 58.7%. This is an increase in correctness which is in the order of gain by increasing the window size. When viewing the global average error the for B_n, the optimum window size is 23×23 or 25×25 with an average error of about 35.3 pixels, slightly less then SSD.

4.3 Interpretation

4.3.1 Overall performance gain

Results of the overall performance in figure 13 show the B_n algorithm performs slightly better at n=90% than the reference SSD method for big window sizes (19×19 and higher). The performance gain on these higher window size is about the same as an step increase in window size. The absolute best performance is with window size 25×25, $n = 90\%$. This point also gives the highest relative gain to SSD of about 0.4%.
A lot can be said about the method to evaluate the performance. First the data set is relatively small. With only 27 items, a few bad-performing items will skew the results. The data set has for some scenes a few variations, which might skew the results further.
When inspecting the graphs per image pair, they clearly show a great amount of variation in performance. This requires further investigation.

4.3.2 In depth research of performance on single scenes

When reviewing the single image result graphs (Appendix E) one can observe an great diversion of performance. For this analysis we focus on the correct prediction fraction only, since the average error is strongly related. We categorize roughly 3 kind of behavior.

1. There is an optimum $n\%$ for which B_n has best performance. These curves are mostly parabolic in shape (see Figure 14). Aloe, Art, Cloth3 and Dolls are examples. In the case the optimum of B_n is with bigger window size and the optimum of SSD is with smaller window size. It is interesting to notice the crossover area where the order of window size flips around (low to high at lower n), (high to low at higher n). In these cases B_n has an advantage. The scenes show complex shapes with little or none plain areas and a lot of disparity discontinuities.

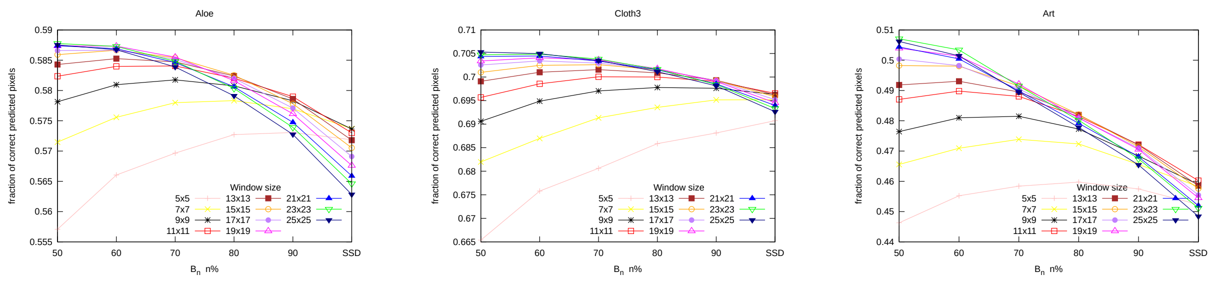


Figure 14: Typical examples where with increasing n the decrease in window size lead to better relative performance. These graphs have an absolute optimum with lower n and bigger windows size. Observe the smaller window size performance start low at low n but crosses over the higher window sizes to end relatively high at high n (SSD).

2. The optimum $n\%$ is at SSD. These graphs show a characteristic strict rising line for all window sizes (see Figure 15). Typical examples are Bowling2, Lampshade1, Monopoly and Plastic. B_n does not have an advantage in these cases. In most of these case the windows size either does not really make a difference or increasing is better. The scenes show simple big plain areas with little disparity discontinuities.

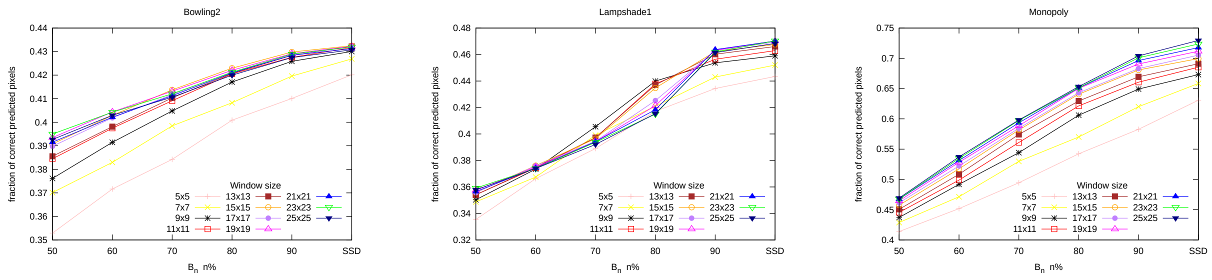


Figure 15: Typical examples where increase in n leads to better performance, with best at n = 100= SSD.

3. The last interesting type of graphs show an effect where window size seems to have a dominating effect on all n: bigger window is better. Examples are Books, Midd1, Reindeer and Monopoly. In some cases B_n has an advantage, but in the other it doesn't. These scenes show some big plain areas, but also a fair amount is disparity discontinuities. It seems that in some of these cases there might be some kind of balance between the effects discribed in 1 and 2 (Midd2 and Reindeer). On the other hand Books and Monopoly do not show that.

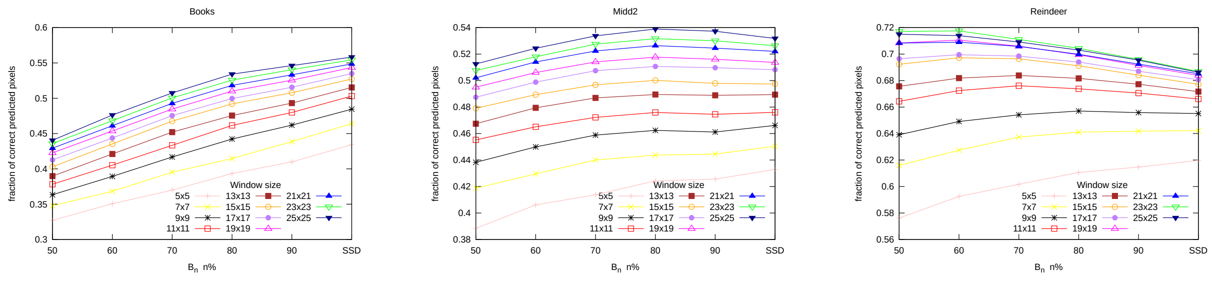


Figure 16: Typical examples where increase in window size leads to better relative performance unrelated to n.

Most other scenes are a combination of the behaviours described above. Now that some typical behaviour is shown, the next sections try to describe and uncover a bit of the reasons of this behaviour of B_n.

4.3.3 In depth: pro's and cons of B_n

An illustrating example of the algorithm working well on ALOE is shown next. It shows the influence of $B_n$on the amount of error near (disparity) edges. A graphical representation of correct vs. incorrect disparity is given in Figure 17. Here the red and green tinted pixels indicate incorrect respectively correct disparity prediction. The image shows a small part upper right corner of the ALOE scene; a textured background with two leaves in front of it. The red 'shadows' are occluded areas created by the same leaves (left) of them. One can observe the left red 'shadow' leaf (occlusion zone) is smaller with $n = 40\%$ and gets broader when increasing n to 100% (SSD). Also the incorrect edges around the real leaves get an increasingly thicker edge around them with increasing n. This is probably the result of better matching of the window with $n = 40\%$. On the edge of objects and occluded areas the background can still get matched with lower $n\%$, which results in smaller prediction error.

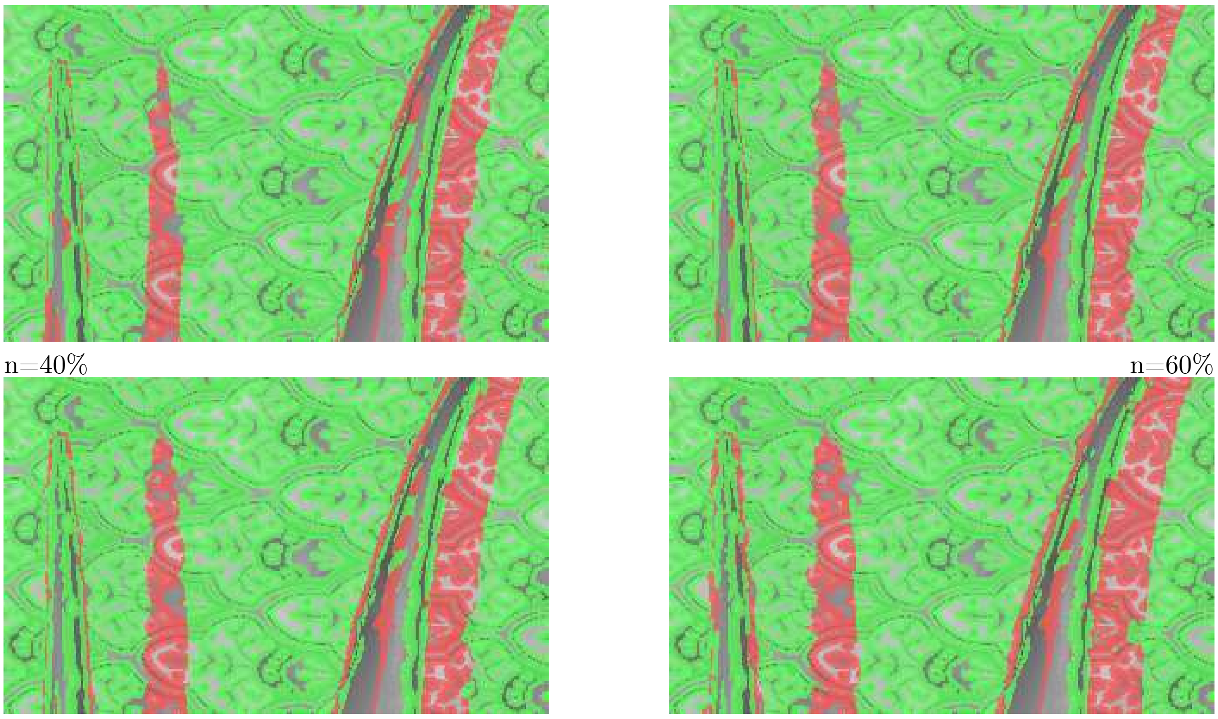


Figure 17: Visual representation of performance on Aloe with window size 23×23 and $n = 40\%, 60\%, 80\%$ and $100\%(= SSD)$. Green, red and greyscale mean correct, incorrect and not evaluated pixels respectively.

The drawback of smaller n (which lead to greater prediction errors in continuous areas) is less evident. In the middle of the left leaf there is a growing patch of incorrect pixels with smaller n, which is assumed to be this drawback.
To evaluate the problems area where the algorithm does not work well a visual inspection of the MONOPOLY scene is done.

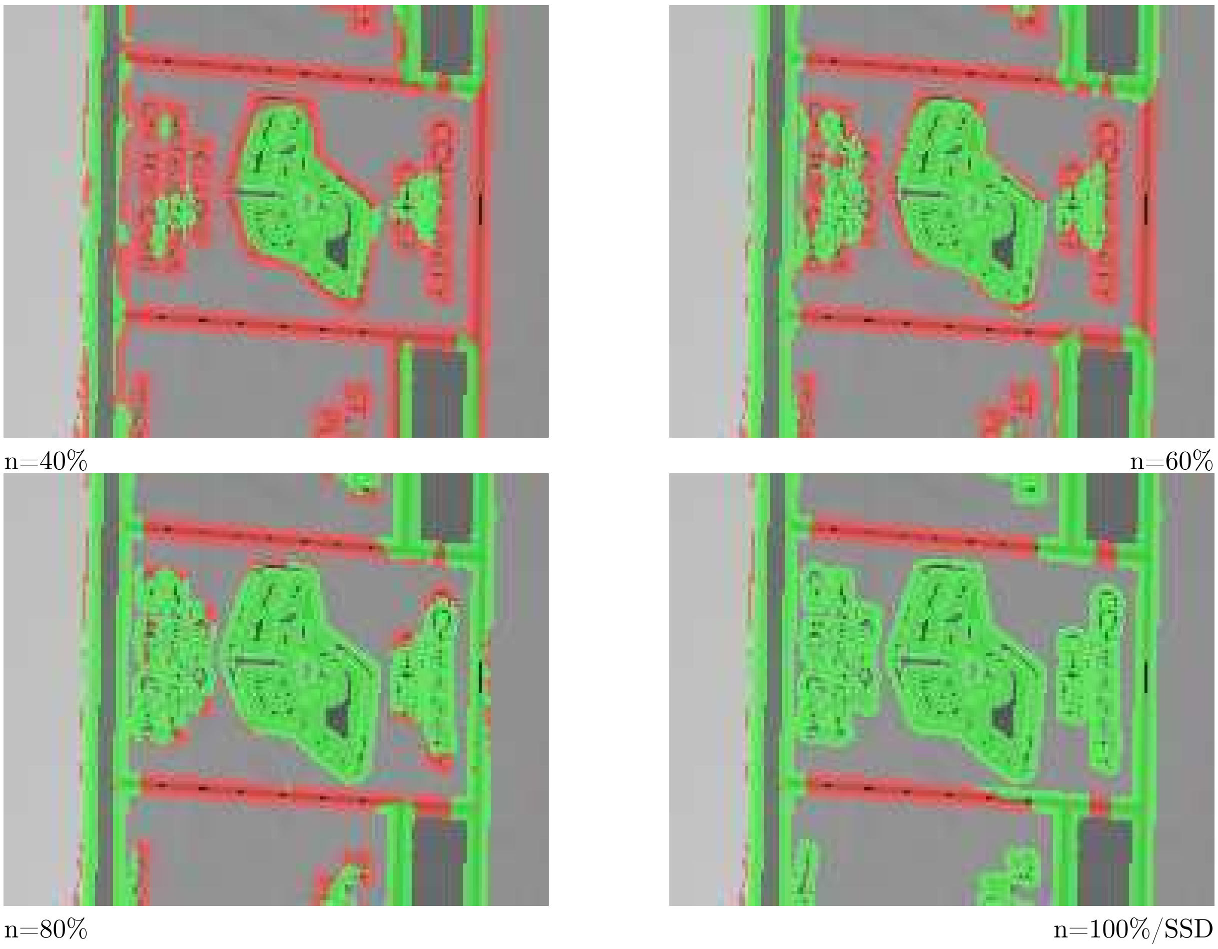


Figure 18: Visual representation of performance on Monopoly with window size 23×23 and $n = 40\%, 60\%, 80\%$ and $100\%(= SSD)$. Green, red and greyscale mean correct, incorrect and not evaluated pixels respectively.

Figure 18 represents a section of the scene; a part of the game board. Around the icons and text on the board a lot of errors can be seen which decrease when increasing n. The left edge in the image is a disparity discontinuity of the board edge against the background. Unlike the edges in the ALOE scene there is no observable increase in correctness at this edge.
Both phenomena could be explained by observing the surrounding area of these feature points: it is plain and contains no feature points (hence the gray area – the points in this area are not marked as feature points due to lack of information. A problem with the $B_n$algorithm is revealed: if a feature point is inside a plain area, this information rich feature point can be discarded by the algorithm. It will therefore reduce the window searching problem to match only a plain area, which is known not to work well. (To put it another way: if a plain window with a few interest points is selected and matched against a series of likewise plain windows with no or little feature points, the likelihood of finding a good match is highest when disregarding the 'noisy' feature points). The more slack the algorithm has with disregarding points (lower n) the more likely it is to disregard important feature points.
The fact that the disparity discontinuous border does not increase performance is that most of the points near this border are not marked as feature points. This is in contrast with the situation in the ALOE scene.
Another observation is the failure in the almost horizontal lines, which shows as the two horizontal red lines. It is possible that the same argument holds partly, but then the SSD method should work better. In this case the most likely cause is that since the area is a slightly tilted line (within a slanted area), the perspective projection on the other camera makes a slightly different angle of line. When inspecting the true deviation, it shows 5 a 6 pixels all over the line. Figure 19 shows the slight difference in slope. Both images have a relative skew. Since the window size is square, the matching of the slightly different angle of lines will happen over a small range of disparities. The algorithm will select the last one (and should select the middle one).

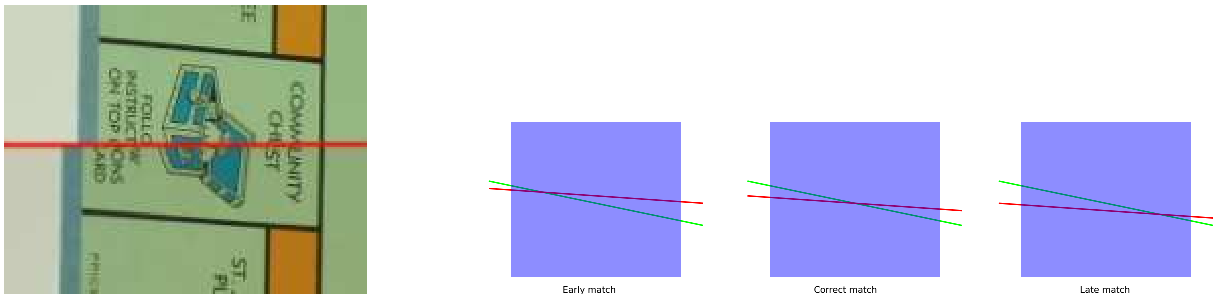


Figure 19: Slanted line problem. Left image: upper and lower half of image show different camera angles. For comparison purpose these images are aligned on the vertical black line. Right three figures: The green line indicates the target line being matched with the red source line. Within the blue support window there is no difference in SSD. The algorithm will select the last best matching window, which is either late or early. Thus the slanted line results in an disparity match offset.

4.4 In-depth analysis on disparity discontinuous regions

To evaluate the algorithms performance on non-continuous regions only, one has to determine these regions. For this analysis this is done using the known true disparity data with the following procedure: A Sobel edge detection algorithm is ran on all disparity ground truth images. This results in lines where disparity is not continuous. To select a region near the lines, the image is thresholded, Gaussian blurred and thresholded again. These regions are used as a mask on the original true disparity data. An example of this process is shown in figure 20. The right image shows the disparity values near the contours of the plant.
An observant viewer could notice most of the lines in the right image actually consists of two lines with different shades of gray as a result of the disparity jump.
A practical problem with the original reference data is that, especially near edges, sometimes no disparity data is given. These unknown disparity spots / regions are filled with average values of their surrounding area before edge detection.

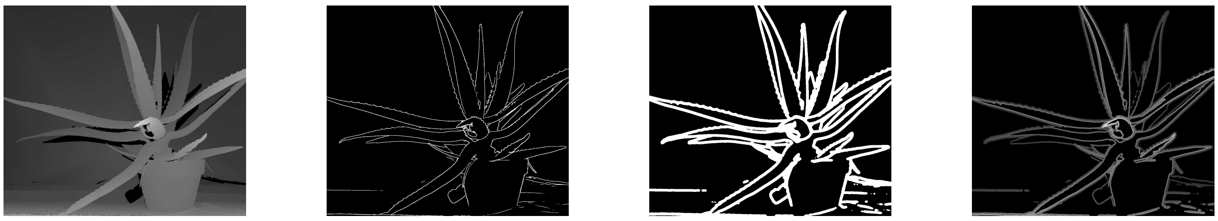


Figure 20: Steps in selecting interest areas for evaluating the B_n algorithm. The left image gives the original true disparity image, the right the selected remaining parts.

These filtered ground truth disparity images are used instead of the original ground truth data and otherwise the analysis is done the same way.
Allowing to evaluate the algorithm only on these interest areas, which are defined by the result of the disparity search in the first place, is creating an as we dutch say a 'kip-ei-probleem', a 'chicken-egg-problem' when implementing the algorithm for use on new data. A resolution is proposed in the next section.
The aggregated results are shown in Figure 21.

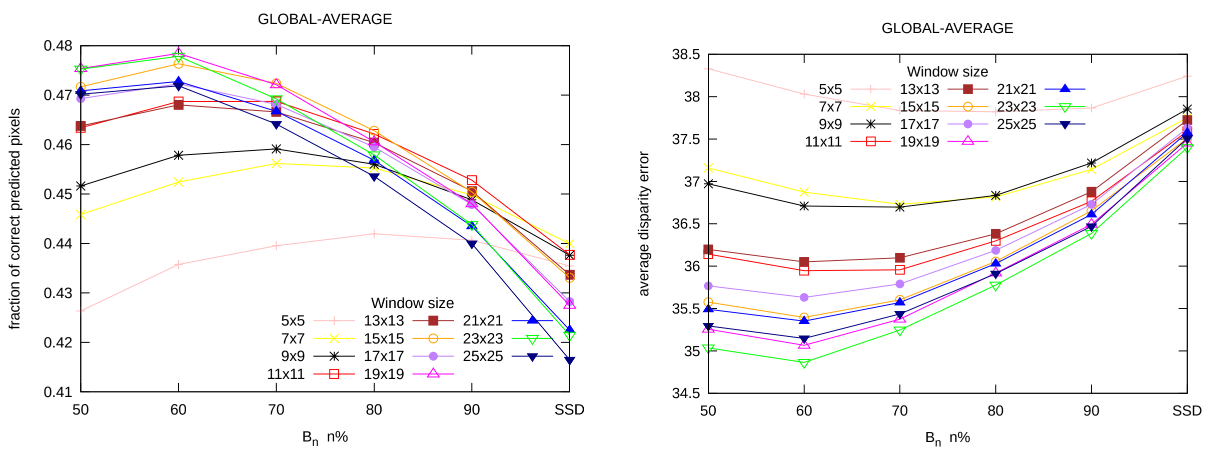


Figure 21: Overall performance of B_n when evaluated only near disparity discontinuities: average fraction of correct pixels (left) and average error (right)

This graph shows the optimal window size for SSD is 7×7 with a correct prediction ratio of 44%. The minimal average error for SSD is around 37.4 pixels with window size 23×23. The best parameters for B_n are window size 19×19 and $n = 60$ for a correct all-round prediction ratio of 47.8%. This is an increase in correctness of 4.8%. The minimal average error can be reduced to 34.8 with window size 23×23 and $n = 60$. This is a reduction of 7.2%.

4.5 Comparison of optimal performance B_n on full range vs. disparity discontinuous area

To conclude Figure 22 shows the best configuration of B_n vs. SSD related to the type of area it is used on.

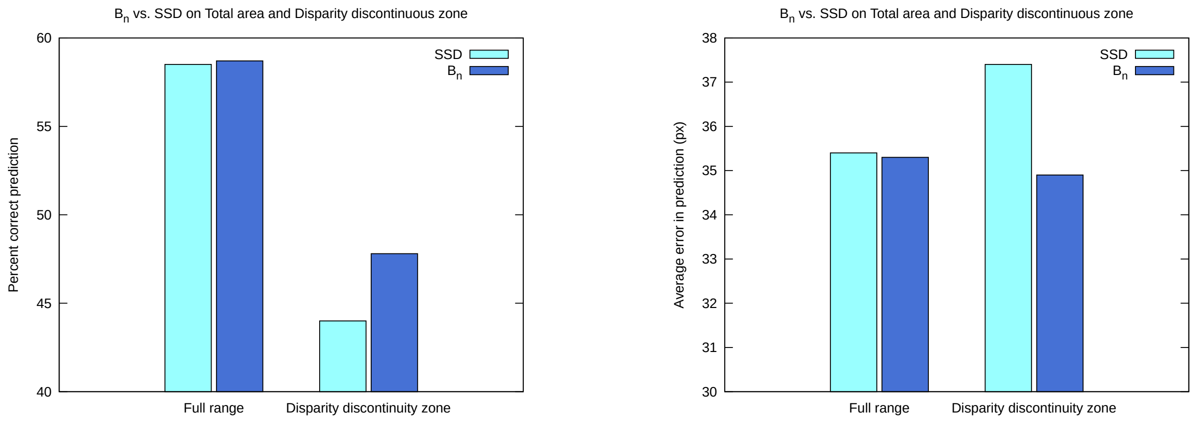


Figure 22: Representation of optimal performance of B_n vs. SSD on full range vs. disparity discontinuity regions only. These graphs show the optimal average performance based on the determined optimal parameters.
Full range parameters: Window sizes for both B_n and SSD on full range: 25×25. $n = 90\%$
Disparity dicontinuity zone parameters: Window size for SSD : 7×7 Window size for B_n: 19×19 and $n = 60\%$.

Both SSD and B_n perform worse in regions of disparity discontinuities, but $B_n$shows significant improvement over SSD in those regions. The average error made by optimal B_n is even better than either SSD or $B_n$on the full range.

5 Conclusion and Discussion

The results of the in depth analysis of algorithms performance per scene show that the performance of B_n is greatly dependent on the type of scene. The algorithm has a potential to perform significantly better at disparity prediction than SSD in scenes with mostly information rich areas (dense with feature points) and many areas with disparity discontinuities. The gain in these situations can be attributed in particular to its correct prediction rate in areas with disparity discontinuities. In these cases the gain is highest when using large window sizes and n around 50%.
The drawback is that B_n will not perform better than SSD in scenes with less regions of disparity discontinuities and will perform significantly worse in areas sparsely populated with feature points. In this case the algorithm might disregard the feature points (information) that reside within the window and match using non-feature point information only; which results in performance worse than SSD.
This divergence between both very good performance on one type of scenes and bad performance on other type of scenes leads to mediocre performance when applied to the whole evaluated data set. On average the best n is 90% with a performance gain of only 0.5%. This gain is marginal and it is therefore it is not really suitable to use B_n without any knowledge about the type of scenes and/or predetermining regions.
The results of the performance of the algorithm in the regions of discontinuity show more promising results. An increase in correct prediction ratio of 4.8% and a decrease of average error of more than 7% can be reached. However, since the data set is rather small, further research may need to be done in order to evaluate whether this holds in a general case.

5.1 Selection of disparity discontinuities

The best performance for B_n is gained in regions with known disparity discontinuity. This happens for example at the edges of objects. This research created a laboratory situation where the disparity edges were known in advance to evaluate the performance in these regions. When applying this in real life this information is not yet known. One suggestion to resolve this problem is to use another algorithm (for example just SSD) first to get an estimate of locations where the disparity jumps might occur and and fine-tune the edge regions with the B_n method. Since the B_n method is shown promising to be successful in these regions, and since this will most likely also disregard the regions where most error by B_n is made, we estimate the gain of this combined method could be in order of $3 - 5\%$ on correct prediction ratio. Due to time and resource constraints this is unfortunately not done and is left for future research.

5.2 Optimizing B_n

The issue with bad performance with windows with sparse feature points could be addressed by augmenting the algorithm with feature point selection density criteria. The idea is that only windows densely populated with feature points would be used. The amount of feature points could be related to the used n for B_n. For example if n is 40%, this could mean that the window should have at least 40% feature points. Otherwise the algorithm could 'choose' to avoid using all feature points in the window.
Another way could be to weigh the importance of single pixels somehow and disregard accordingly. Pixels marked as feature point might be weighed and using this separate weight matrix make them less likely to be excluded by B_n.
This idea leads to another idea which is based on the idea that regions to disregard are most likely connected points. B_n includes or exclude pixels without regarding their distribution in the window.

5.3 Unevaluated areas of interest

The selection of pixels to discard by B_n is unrelated to the connectivity to the centre (main) feature point of the window nor to each other. This makes it prone to match the wrong area. For example it can match on the area around a wrong feature point which also resides in the window. In cases with a region of occlusion this is no problem (these feature points will be occluded), but on disparity discontinuity edges where both edges are visible this might occur. However, this is not yet really observed. Research in optimisation of B_n by including connectness to the centerpoint and/or connectness between disregarded points is suggested.

6 Final word

When starting this project I had the ambition of creating an application that performed a reconstruction of a (model of a) $3D$ scene using a series of perpectives (photographs) on this scene. My supervisor Dr. Michael Lew suggested to focus on this point correspondence problem, which was in my eyes then an irrelevant, trivial and tiny, detail of this challenge.
In the process of working on this project it became clear that I could not be more wrong. The amount of research done on this area did overwhelm me with information and I did not really know where to begin and what was relevant or not. This whole domain including its terminology and known do's and don'ts was totally unknown for me. Thinking of my original goal I finally just got started with writing some code to do something, without really understanding the difficulties, thinking that I had found the solution.
In the subsequent period of time I learned that doing research is a cyclic process. One has to read literature to use information that others provide, "building on the shoulders of giants", but at the same time apply this information to get a grasp of the meaning of this knowledge. And then dive back again into the literature. With increasing understanding of the subject the literature reveils more and more secrets. Which then need to be grasped again...
In the course of this research I have learned a lot about the basics in this problem domain and have seen some very interesting work, which sparked a lot of ideas. But I also know that this is just the tip of the iceberg, it is just the beginning and a lot of work in this field of research and cool application can still be done.
In the future I will be careful with making assumptions, as I have discovered that you can be almost sure that the assumptions you make at the start of your research are probably wrong or at least not near complete nor insightful. But thats what research is all about, right?
I want to thank Michael for showing me the way on my first trip into real research. His supervision gave me not only direction but provided me with exactly enough to grasp on when I got stuck.

References

[1] Scharstein, D., & Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International journal of computer vision, 47(1-3), 7-42.

[2] Lazaros, N., Sirakoulis, G. C., & Gasteratos, A. (2008). Review of stereo vision algorithms: from software to hardware. International Journal of Optomechatronics, 2(4), 435-462.

[3] Barnard, S. T., & Thompson, W. B. (1980). Disparity analysis of images. Pattern Analysis and Machine Intelligence, IEEE Transactions on, (4), 333-340.

[4] Kanade, T., Kano, H., Kimura, S., Yoshida, A., & Oda, K. (1995, August). Development of a video-rate stereo machine. In Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on (Vol. 3, pp. 95-100). IEEE.

[5] Kanade, T., & Okutomi, M. (1994). A stereo matching algorithm with an adaptive window: Theory and experiment. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 16(9), 920-932.

[6] Pollefeys, M., Koch, R., Vergauwen, M., & Van Gool, L. (2000). Automated reconstruction of 3D scenes from sequences of images. ISPRS Journal of Photogrammetry and Remote Sensing, 55(4), 251-267.

[7] Lotti, J. L., & Giraudon, G. (1994, August). Adaptive window algorithm for aerial image stereo. In Spatial Information from Digital Photogrammetry and Computer Vision: ISPRS Commission III Symposium (pp. 517-524). International Society for Optics and Photonics.

[8] Yoon, K. J., & Kweon, I. S. (2006). Adaptive support-weight approach for correspondence search. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(4), 650-656.

[9] He, Y., Wang, P., & Fu, J. (2013, December). An Adaptive Window Stereo Matching Based on Gradient. In 3rd International Conference on Electric and Electronics. Atlantis Press.

[10] Hosni, A., Bleyer, M., & Gelautz, M. (2013). Secrets of adaptive support weight techniques for local stereo matching. Computer Vision and Image Understanding, 117(6), 620-632.

[11] Fusiello, A., Roberto, V., & Trucco, E. (1997, June). Efficient stereo with multiple windowing. In 2013 IEEE Conference on Computer Vision and Pattern Recognition (pp. 858-858). IEEE Computer Society.

[12] Kang, S. B., Szeliski, R., & Chai, J. (2001). Handling occlusions in dense multi-view stereo. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on (Vol. 1, pp. I-103). IEEE.

[13] Scharstein, D., & Szeliski, R. (2003, June). High-accuracy stereo depth maps using structured light. In Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on (Vol. 1, pp. I-195). IEEE.

[14] Geiger, A., Roser, M., & Urtasun, R. (2011). Efficient large-scale stereo matching. In Computer Vision–ACCV 2010 (pp. 25-38). Springer Berlin Heidelberg.

[15] Oculus Rift. http://www.oculusvr.com/rift/

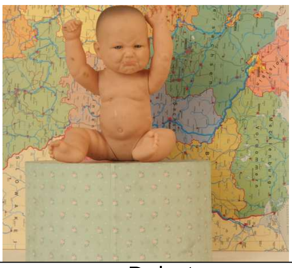
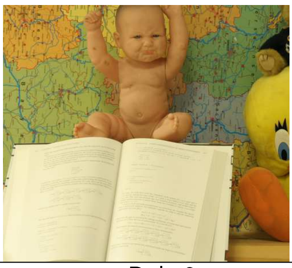
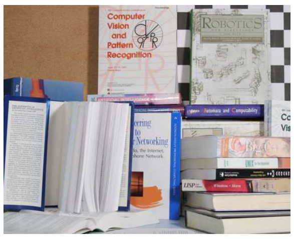
[16] Google self driving car project. https://plus.google.com/+GoogleSelfDrivingCars/posts

[17] Fusiello, A., Trucco, E., & Verri, A. (2000). A compact algorithm for rectification of stereo pairs. Machine Vision and Applications, 12(1), 16-22.

[18] Papadimitriou, V., & Dennis, T. J. (1996). Epipolar line estimation and rectification for stereo image pairs. Image Processing, IEEE Transactions on, 5(4), 672-676.

[19] Kumar, S., Micheloni, C., Piciarelli, C., & Foresti, G. L. (2010). Stereo rectification of uncalibrated and heterogeneous images. Pattern Recognition Letters, 31(11), 1445-1452.

A Table of images in dataset

			
Aloe	Art	Baby1	Baby2
			
Baby3	Books	Bowling1	Bowling2
Cloth1	Cloth2	Cloth3	Cloth4
Dolls	Flowerpots	Lampshade1	Lampshade2

Laundry	Midd1	Midd2	Moebius
Monopoly	Plastic	Reindeer	Rocks1
Rocks2	Wood1	Wood2	

B Code listing of C++ implementation of B_n algorithm

Algorithm 1 B_n aggregate window function and comparison

```
// returns sum of squared RGB distances
// max: 3*255^2 = 195075
unsigned int static RGBdist(Pixel pixela, Pixel pixelb) {
    return (sqr(R(pixela) − R(pixelb)) + sqr(G(pixela) − G(pixelb))
            + sqr(B(pixela) − B(pixelb)));
}

// compare a block of pixels, but count only the best matching n% in 9 steps
// for a window size of NXN
b_n_res DisparityMapping::bestncomparenxn(int const n, int const rx,
        int const ry, int const tx, int const ty) {
    b_n_res results;

    static unsigned long ssdsum;
    static int vi;
    static unsigned long ssd_values[25 * 25]; // sized for max window size

    vi = 0;
    for (int y = −(n / 2); y < ((n + 1) / 2); ++y)
        for (int x = −(n / 2); x < ((n + 1) / 2); ++x)
            ssd_values[vi++] = RGBdist(ref[ry + y][rx + x],
                    seek[ty + y][tx + x]);

    // sort array
    std::sort(ssd_values, ssd_values + n * n);

    ssdsum = 0;
    // sum only first n% (lowest) values
    int si = 0;
    for (int c = 1; c < 10; c++) {
        for (; si < (n * n * c / 10); si++)
            ssdsum += ssd_values[si];
        results.n[c − 1] = ssdsum;
    }

    return results;
}
```

C Code listing of C++ implementation of feature point selection function

Algorithm 2 Feature point selection algorithm (generates mask)

```
// returns true if x > delta or x < -delta
static bool thresh(int const x, int const delta) {
    return (x < -delta || x > delta);
}

// SobelFeatureMask generates mask image of original using a Sobel 5x5 kernel convolution
// (255,255,255): interest point ; (0,0,0): no interest point
Image* SobelFeatureMask(Image* original) {
    size_t width = original->width;
    size_t height = original->height;

    Image* result = new Image(width, height);

    static const int sob_h[5][5] = { { 2, 2, 4, 2, 2 }, { 1, 1, 2, 1, 1 }, { 0,
            0, 0, 0, 0 }, { -1, -1, -2, -1, -1 }, { -2, -2, -4, -2, -2 } };

    static const int sob_v[5][5] = { { 2, 1, 0, -1, -2 }, { 2, 1, 0, -1, -2 }, {
            4, 2, 0, -2, -4 }, { 2, 1, 0, -1, -2 }, { 2, 1, 0, -1, -2 } };

    for (int y = 2; y < height - 2; y++)
        for (int x = 2; x < width - 2; x++) {
            static int res_hR, res_hG, res_hB;
            static int res_vR, res_vG, res_vB;

            res_hR = res_hG = res_hB = 0;
            res_vR = res_vG = res_vB = 0;

            for (int dy = -2; dy < 3; dy++)
                for (int dx = -2; dx < 3; dx++) {
                    res_hR += sob_h[dy + 2][dx + 2]
                            * original->R[y + dy][x + dx];
                    res_hG += sob_h[dy + 2][dx + 2]
                            * original->G[y + dy][x + dx];
                    res_hB += sob_h[dy + 2][dx + 2]
                            * original->B[y + dy][x + dx];

                    res_vR += sob_v[dy + 2][dx + 2]
                            * original->R[y + dy][x + dx];
                    res_vG += sob_v[dy + 2][dx + 2]
                            * original->G[y + dy][x + dx];
                    res_vB += sob_v[dy + 2][dx + 2]
                            * original->B[y + dy][x + dx];
                }

            const static int tv = 255;
            if (thresh(res_hR, tv) || thresh(res_hG, tv) || thresh(res_hB, tv)
                    || thresh(res_vR, tv) || thresh(res_vG, tv)
                    || thresh(res_vB, tv)) {
                result->R[y][x] = 255;
                result->G[y][x] = 255;
                result->B[y][x] = 255;
            } else {
                result->R[y][x] = 0;
                result->G[y][x] = 0;
                result->B[y][x] = 0;
            }
            /*          result->R[y][x] = clip255(res_hR/10);
             result->G[y][x] = clip255(res_hG/10);
             result->B[y][x] = clip255(res_hB/10);
             */
        }

    return result;
}
```

D Code listing of disparity discontinuity region mask creation

Algorithm 3 GIMP code to select interest areas for evaluation

```
(define (bp-filter-edge filename)
  (gimp-file-load RUN-NONINTERACTIVE filename filename)

    ( let ( ( myimage (car (gimp-image-list)) ) (test 1) )
    ( let ( ( baselayer (car (gimp-image-get-active-layer myimage)) ) )
      ( let (
        ( duplayer1 (car (gimp-layer-copy baselayer 0 ) ) )
        ( duplayer2 (car (gimp-layer-copy baselayer 0 ) ) )
        ( duplayer3 (car (gimp-layer-copy baselayer 0 ) ) ) )
            (gimp-image-add-layer myimage duplayer1 -1 )
            (gimp-image-add-layer myimage duplayer2 -1 )
            (gimp-image-add-layer myimage duplayer3 -1 )
            (gimp-threshold duplayer2 0 1 )
          (plug-in-gauss 1 myimage duplayer1 10.0 10.0 1)
          (gimp-layer-set-mode duplayer2 MULTIPLY-MODE) ; multiply
          (gimp-layer-set-mode duplayer3 ADDITION-MODE)
          (gimp-image-merge-down myimage duplayer2 0)
          ( let ( ( smoothed (car (gimp-image-merge-down myimage duplayer3 0) ) ) )
            (gimp-layer-flatten smoothed) ; remove alpha
            (plug-in-edge 1 myimage smoothed 5.0 2 0) ; smear, sobel
            (gimp-threshold smoothed 40 255)
            (plug-in-gauss 1 myimage smoothed 10.0 10.0 1)
            (gimp-threshold smoothed 2 255)
            (gimp-layer-set-mode smoothed MULTIPLY-MODE)
            (gimp-file-save
              1
              myimage
              (car (gimp-image-merge-down myimage smoothed EXPAND-AS-NECESSARY) )
              (car (gimp-image-get-filename myimage) )
              (car (gimp-image-get-filename myimage) )
            )
          )
      )
    )
  )
)
```

E Results of B_n when feature point detection is included

These are the results of B_n when feature point selection is done as described in Section 3.2.2.

E.1 Correct classification fraction per sample pair

For each individual item pair from the data set a graph is drawn of B_n using $n = 10\%$ to 100%, where $B_{100} = SSD$. The graphs in Figures 23,24,25 and 26 show the fraction of perfect predicted disparity. The used threshold $\eta = 3$; the margin for perfect classification is 2 pixels off.

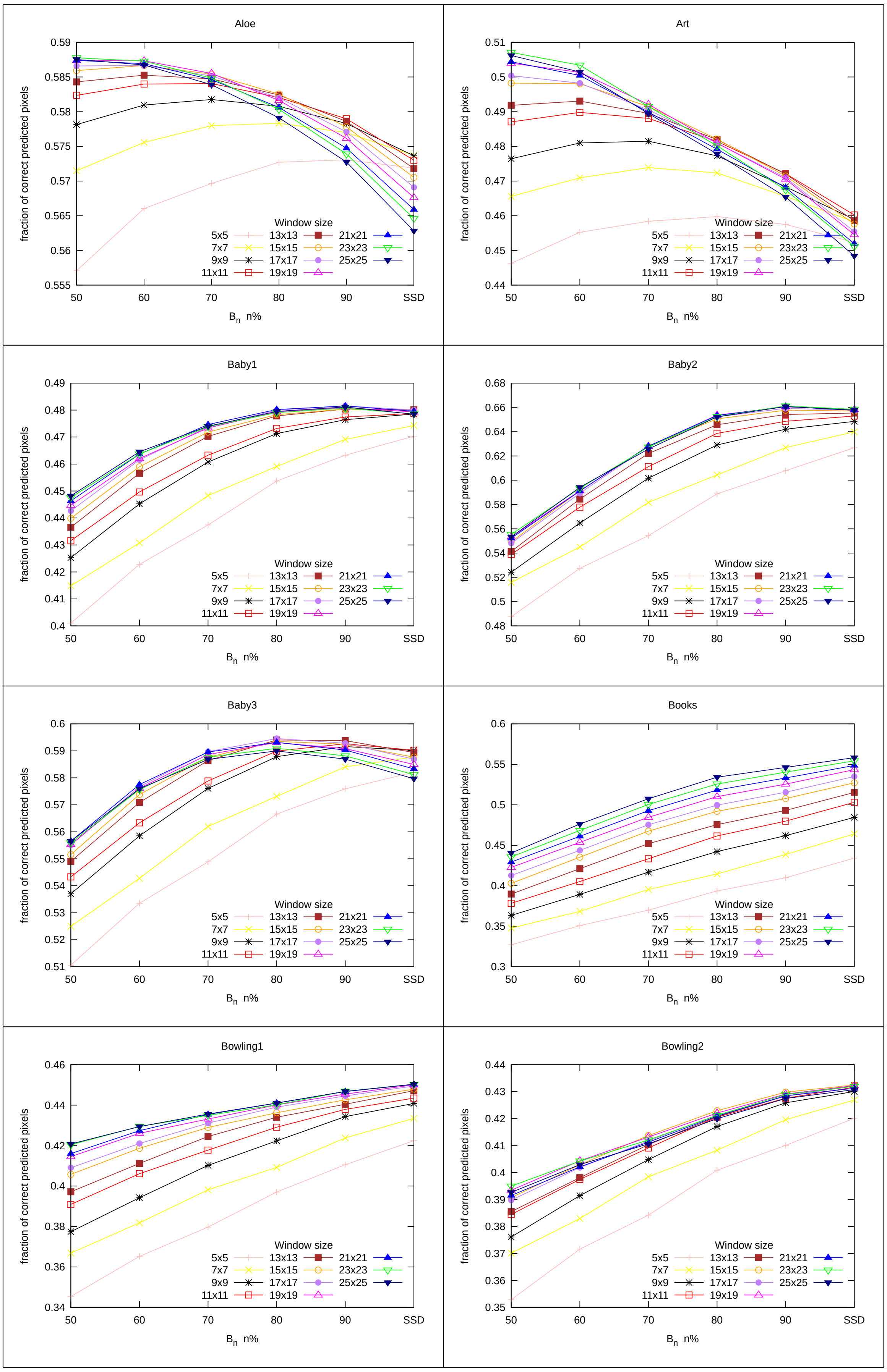


Figure 23: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes

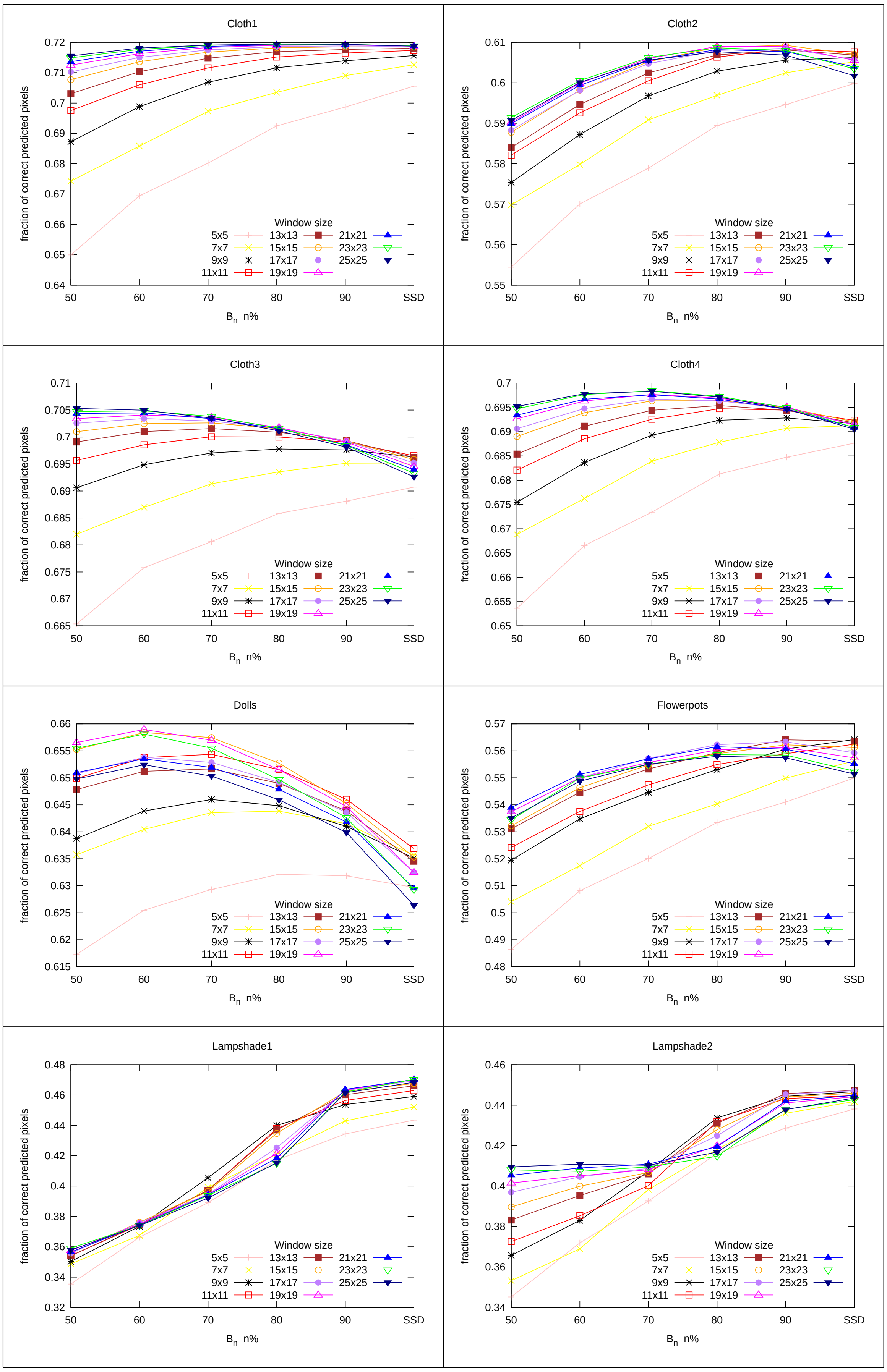


Figure 24: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes

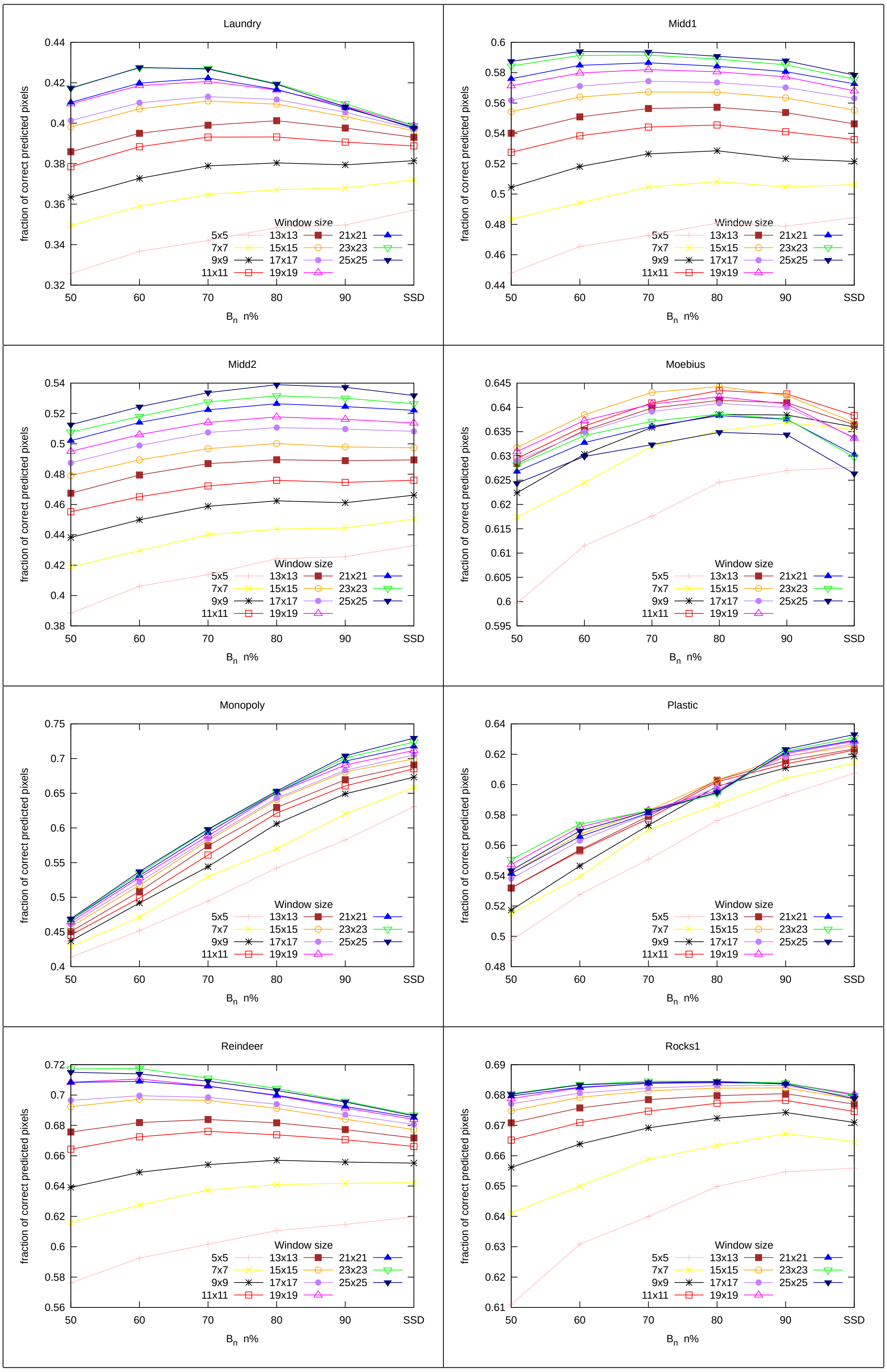


Figure 25: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes

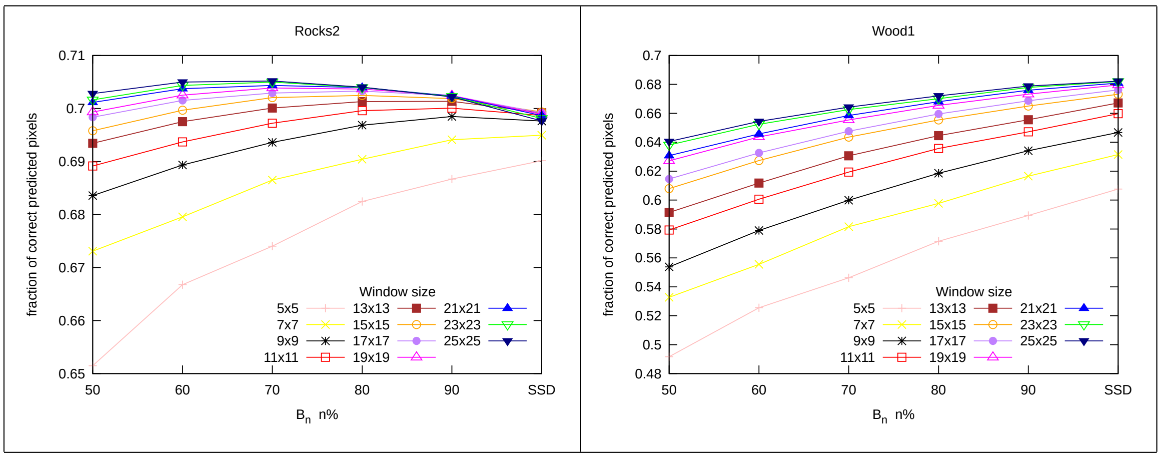


Figure 26: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes

E.2 Average error in predicted disparity per sample pair

The analysis of average error per image is shown in Figures 27,28, 29 and 30. The analysis is done like in section 4.2.

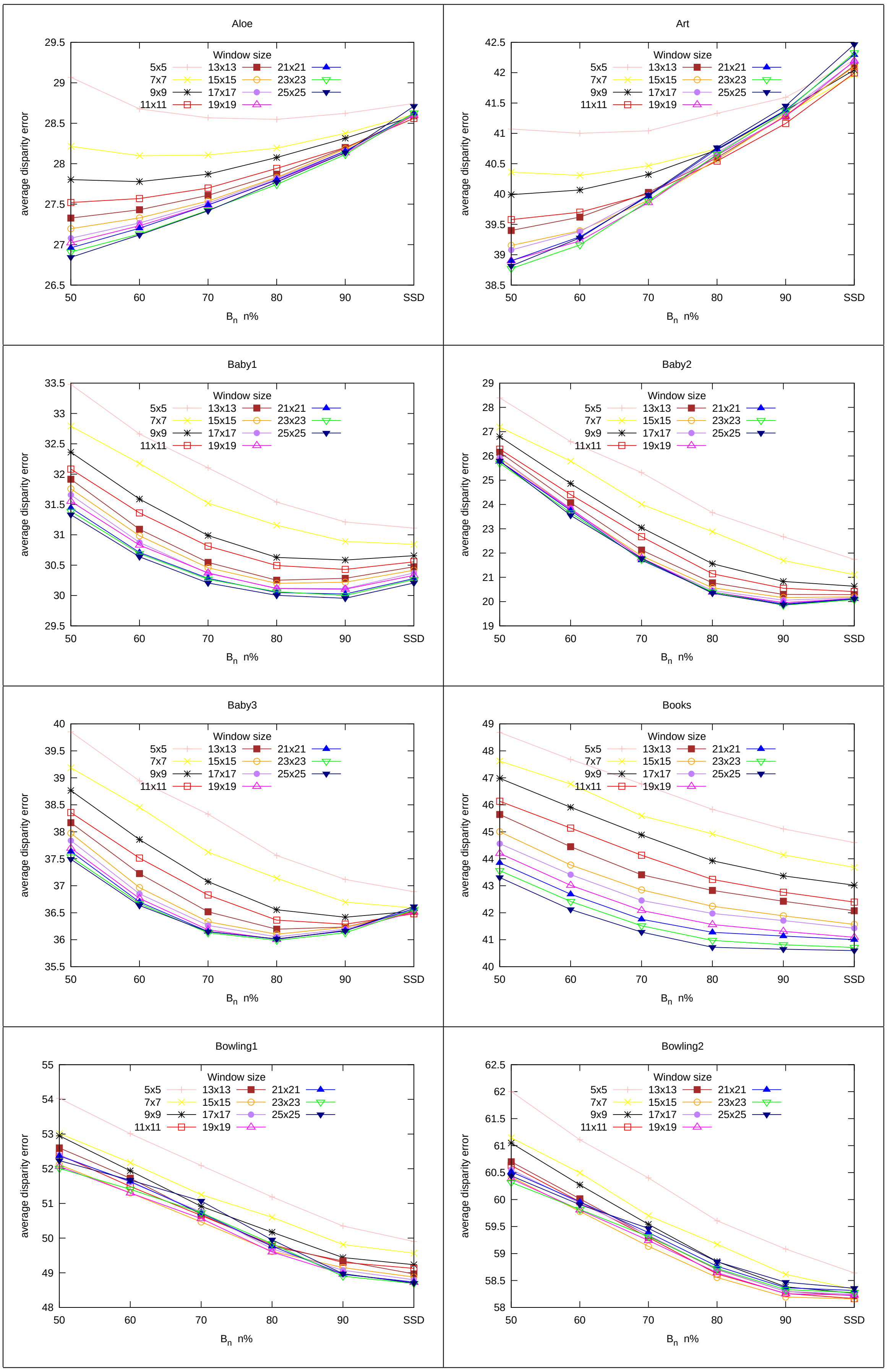


Figure 27: Performance of B_n with respect to n. Average error in predicted disparity shown for each pair and 11 different window sizes.

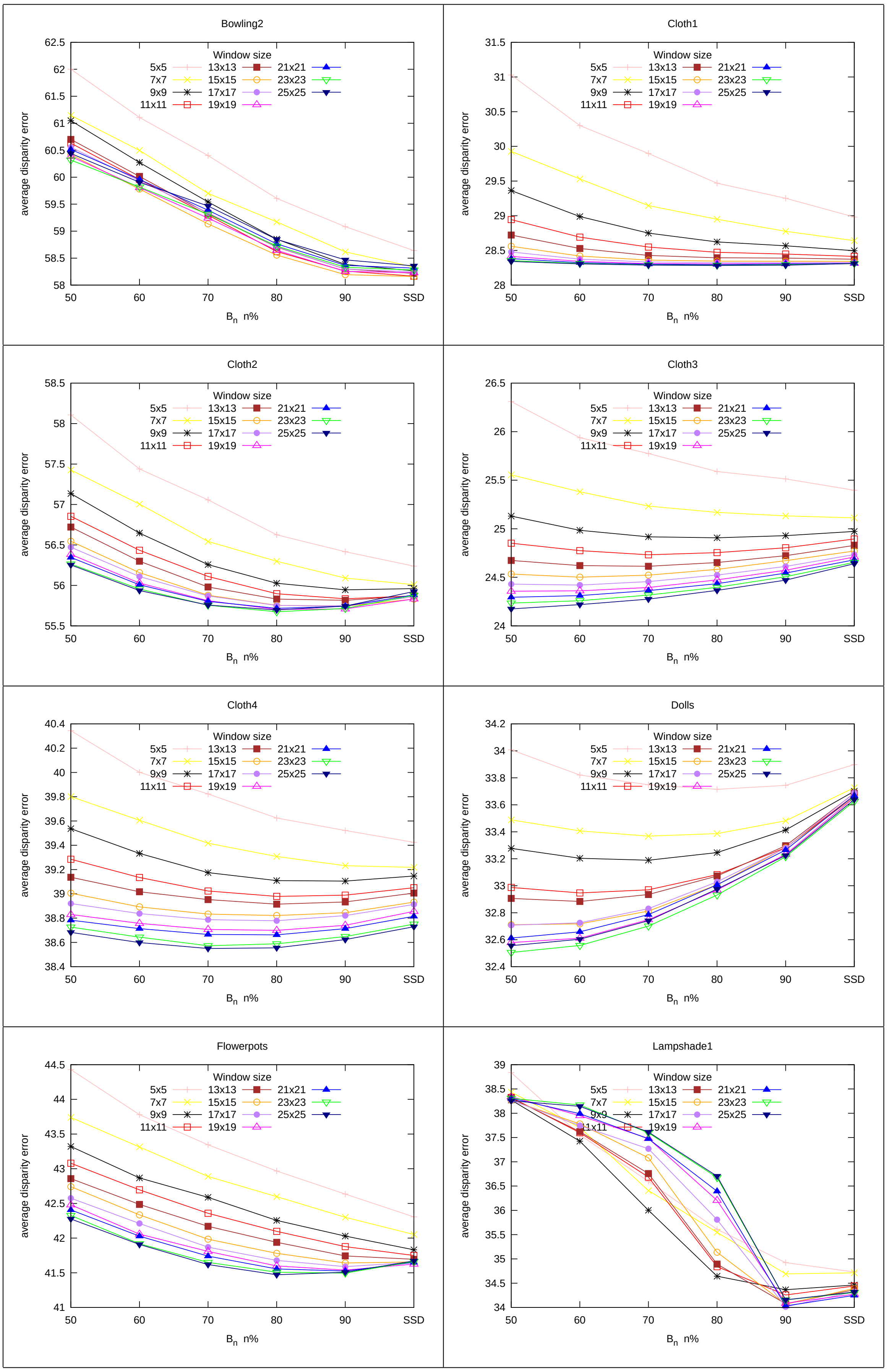


Figure 28: Performance of $B_n$with respect to n. Average error in predicted disparity shown for each pair and 11 different window sizes.

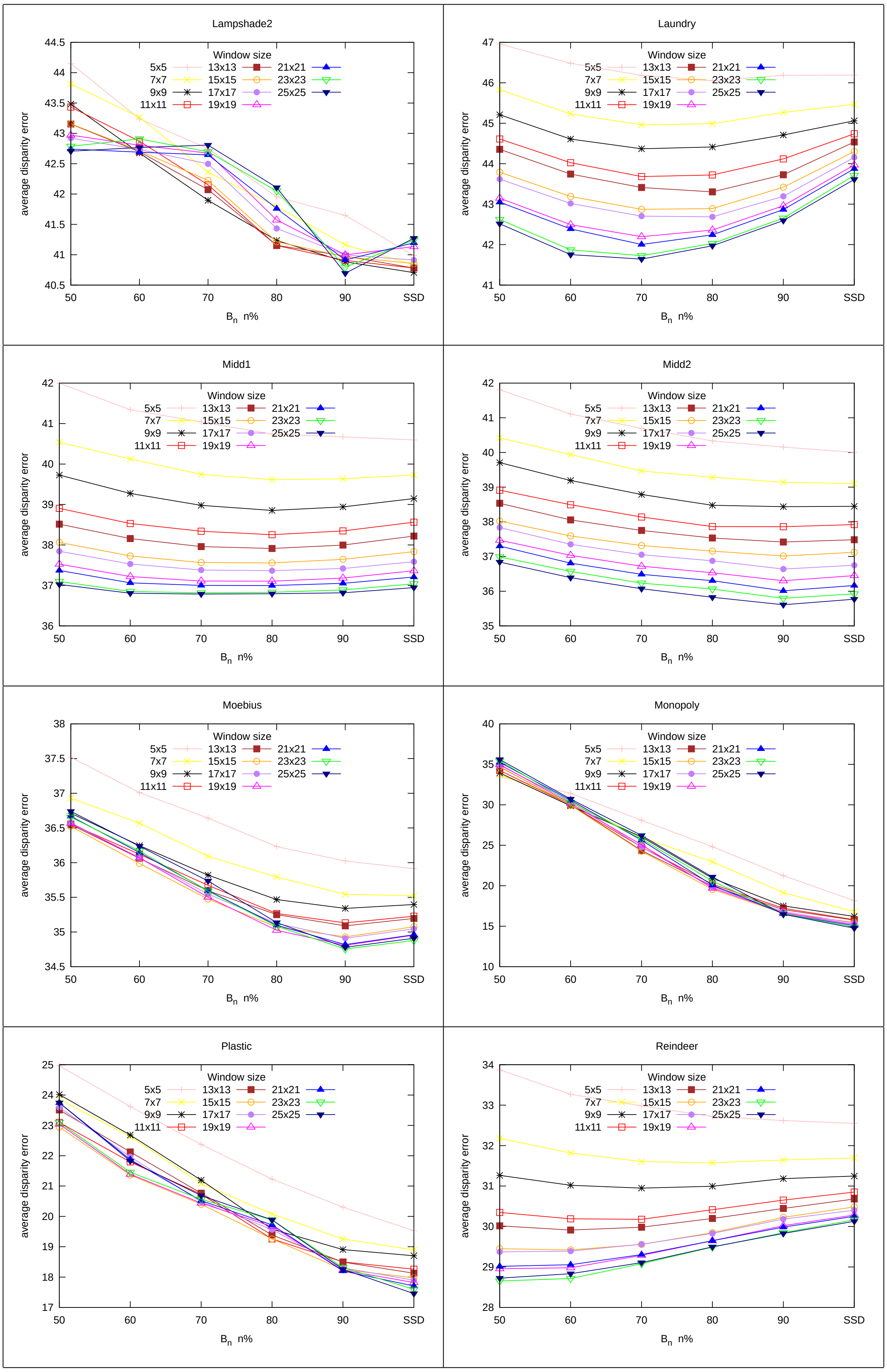


Figure 29: Performance of B_n with respect to n. Average error in predicted disparity shown for each pair and 11 different window sizes.

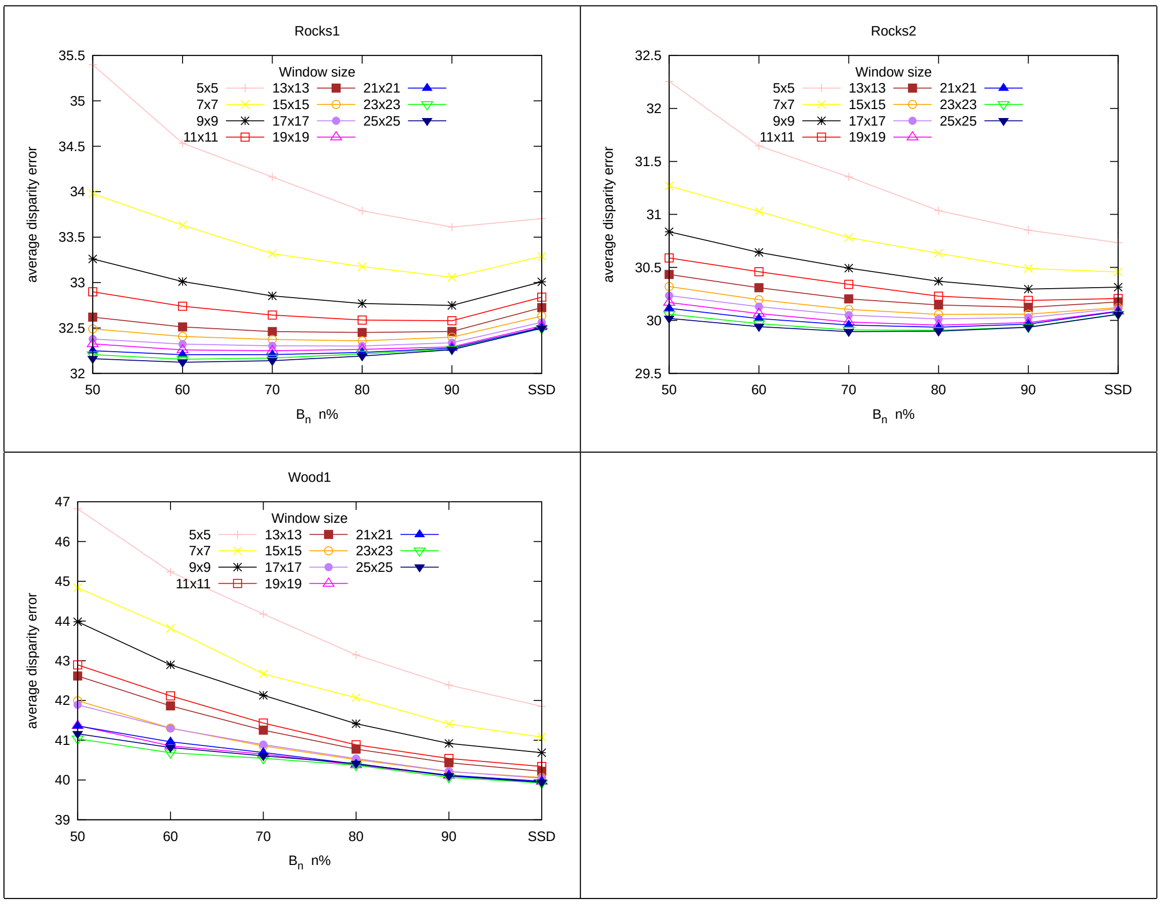


Figure 30: Performance of $B_n$with respect to n. Average error in predicted disparity shown for each pair and 11 different window sizes.

F Results of B_n when no feature point detection is done

This shows the correct classification fraction for each image pair.

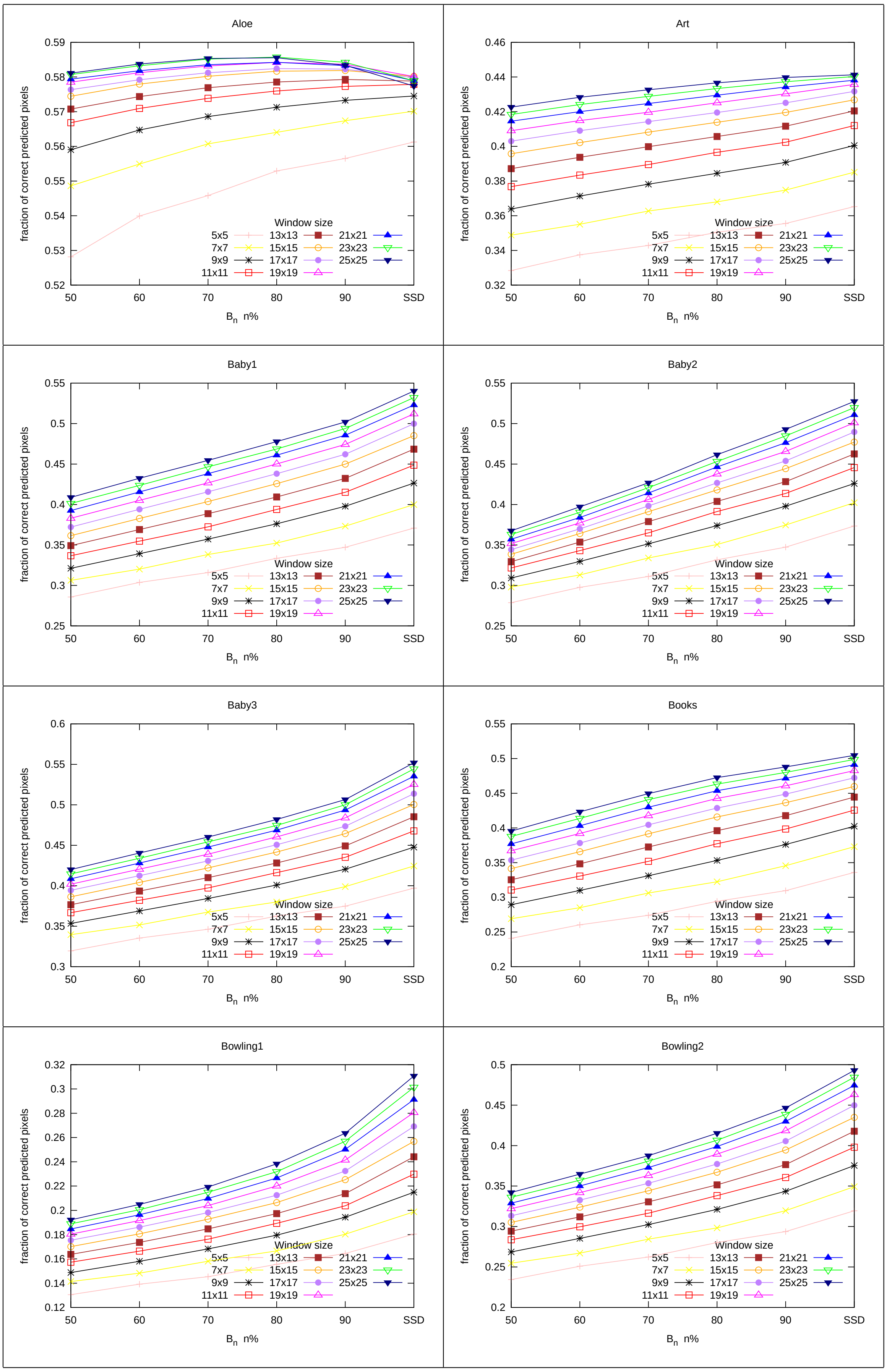


Figure 31: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes

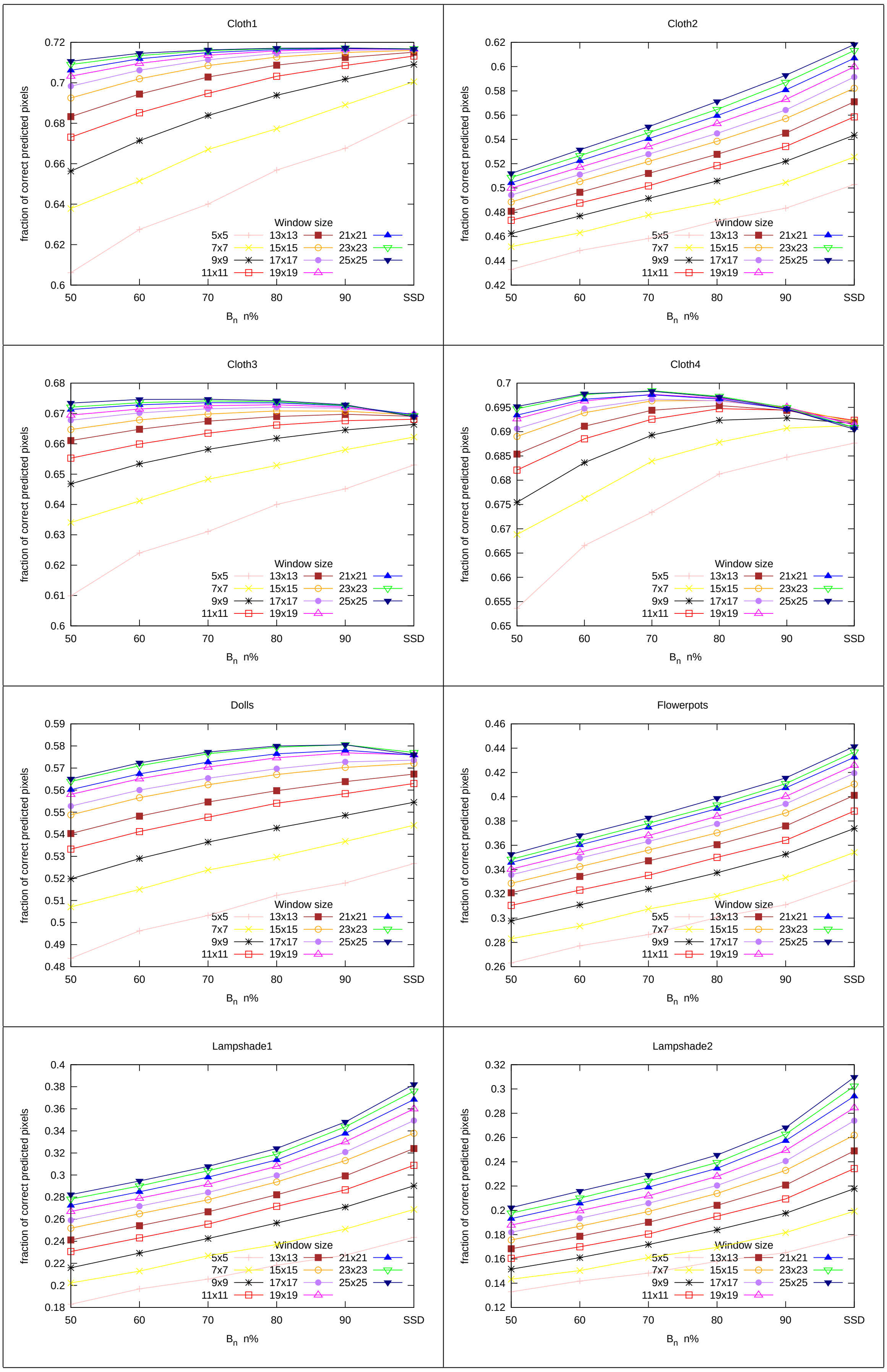


Figure 32: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes

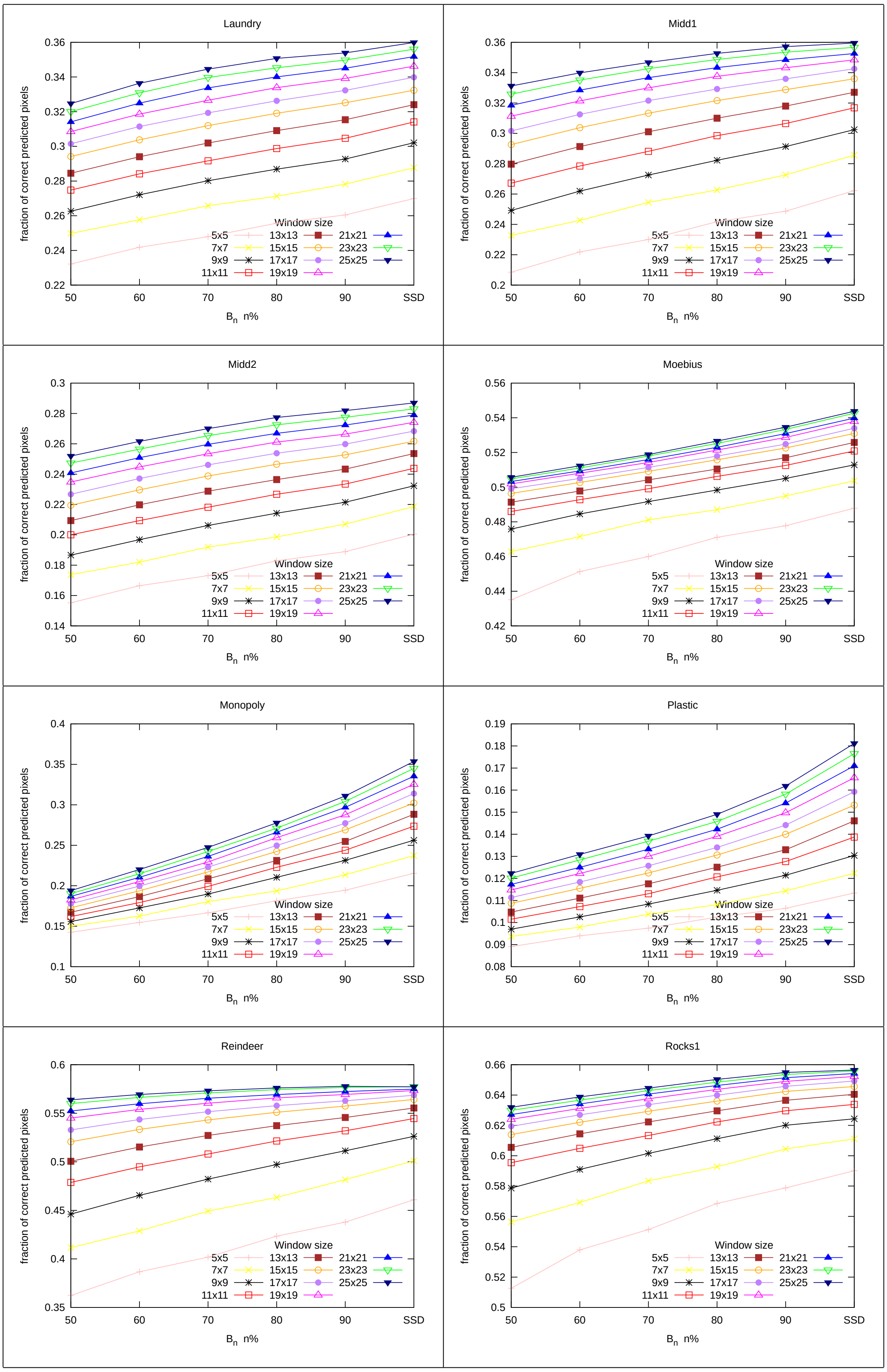


Figure 33: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes

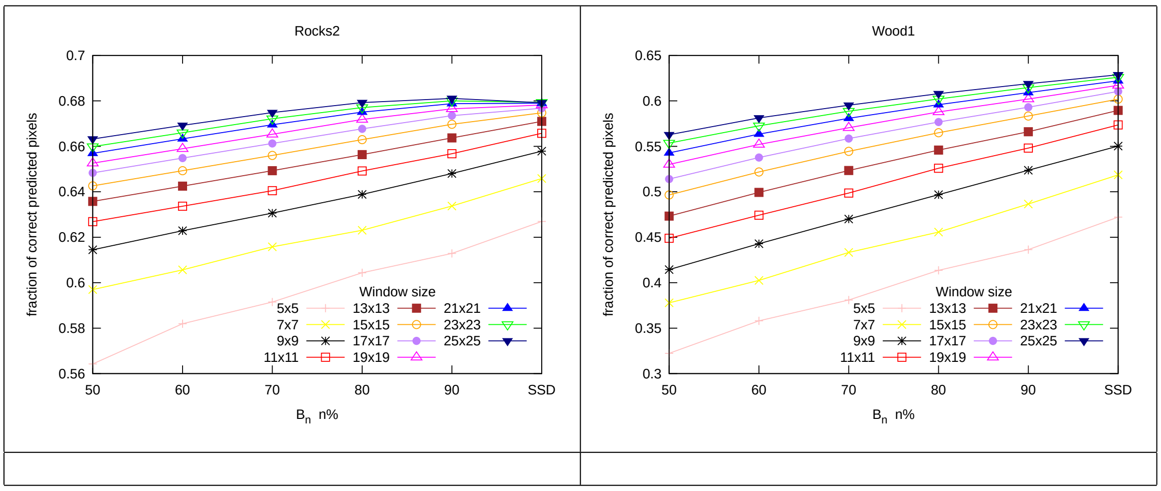


Figure 34: Performance B_n with respect to n. Shown correct match fraction with $\eta = 3$ per image pair. Each graph shows performance for all window sizes