Semantic annotation of urban scenes: Skyline and window detection

MSc Thesis

written by

Tjerk Kostelijk
mailtjerk@gmail.com
(born June 14th, 1983 in Alkmaar, the Netherlands)

under supervision of **Isaac Esteban** and **Prof. dr ir Frans C. A. Groen**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

Master of Science in Artificial Intelligence

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
Juli 6th, 2012	Prof. dr ir Frans C. A. Groen
	dr. P.H. Rodenburg
	dr. Arnoud Visser

Universiteit van Amsterdam

Contents

Acknowledgements

During this research I received a large amount of support from a variety of people. For this I am very grateful and I take this change to thank all of them.

I'd like to thank my first supervisors Isaac Esteban for developing an idea that forms the start of this thesis. Id like to thank him for sharing all his knowledge about the field of multiple view geometry. Although there was a large difference in expectations on both sides, I learned a lot of him. In my final period he offered his time to gave me feedback about the parts which we did together even though he was very busy writing his own thesis.

Furthermore I'd like to thank my second supervisor Frans Groen who gave me such professional supervision. He learned me the ingredients that a professional thesis needs. Although he is almost retired, he came with very innovative ideas. I loved the sessions where we brainstormed about the techniques we used and where we inspired each other by thinking out of the box. Although he was very busy finishing his job at UVA and TNO, he gave me full support and spend large amounts of his time for my supervision.

Next I'd like to thank dr Piet Rodenburg who came from another research group and was willing to join the committee on such a short notice. Not only was the subject new for him, also the time to read this extensive thesis was very short. dr Piet Rodenburg even took some time of is holiday to read through a draft version of my thesis.

Furthermore I'd like to thank dr Arnoud Visser, who immediately said yes when I invited him to the committee. To participate my defense, Arnoud even arranged his family to pick up his son from his last day of school. Also thanks for providing me latex formats and for reading my thesis in such a short time.

I would like to thank my girlfriend Anne de Graaf who supported me during the entire process. I learned a lot from her motivation, her passion, her gratefulness and her mentality to work hard without forgetting to enjoy life. She used her excellent English skills and invested a large amount of her expensive time to provide me spelling and grammar feedback. Anne, thanks for believing in me and for all the support that you gave me with so much love.

I would also like to thank my parents for their financial support, their latent pressure, there advise and, most important, their love.

A large part of this thesis was written in company with Toine van Asten who is my best friend. He is one of the most motivated students I have ever met. Although he studies at Delft, we worked large amount of time together. During his exam periods and my thesis deadlines he learned me to take over hours as a dessert. Toine, thanks for supporting me with your commitment, insights, out of the box thinking, positive energy, jokes, and your friendship.

I'd like to thank Bram Stoeller, my fellow classmate and, more important, warm friend, for offering a large amount of his own time to read my thesis.

I'd like to thank Stefanie Kooistra who took a large amount of her time to evaluate my thesis without expecting something in return. Not only did she highlight my spelling and grammar errors, she also learned me how to use the grammar rules properly which was very helpful.

Furthermore, I'd like to thank Gineke Sietsma who trained me in Personal Leadership where I learned to develop my full potential.

Abstract

We build an automatic system that adds semantics to an urban scene. A 3D model was extracted by combining the *FIT3D toolbox[11]* with a skyline detection algorithm. The skyline detector extracts straight line segments from the upper edge of an image. These are used to determine the wall heights of an extended 2D model extracted from *Openstreetmap[1]*. Next, windows are detected by two different methods:the first method is invariant to viewing direction and detects the windows by searching for connected horizontal and vertical edges. The second method is applied on rectified facades and consist of 1) the determination of the alignment of the windows and 2) the classification of the windows. The alignment and classification are based on the interpretation of a Histogram function that contains Hough lines extracted from edges. Both methods detect at least 99% of the windows.

1 Introduction

When we humans look at an urban scene we immediately can tell which part represents a building, a tree, a door, a window or a parked car. Even if the scene suffers from high occlusion (e.g. a tree occluding the largest part of a building) or extreme perspective distortion (a building seen from the corners of your eye) we perform this task with a very high accuracy. For a computer system however, this task is far from trivial.

Let us address the question that arises many times in Artificial Intelligence: Why are we humans so good in this task? What can we learn from ourselves and how can we apply this on a computer system?

The most important reason of our excellent visual perception is that we combine a series of depth cues [25] (which enables us to experience depth) with feature-matching (which enables us to classify objects). One of the most important depth cue is binocular disparity. We use two eyes and look at the same scene from slightly different angles. This makes it possible to triangulate the distance to an object with a high degree of accuracy [25] [12]. Figure 1 illustrates this.

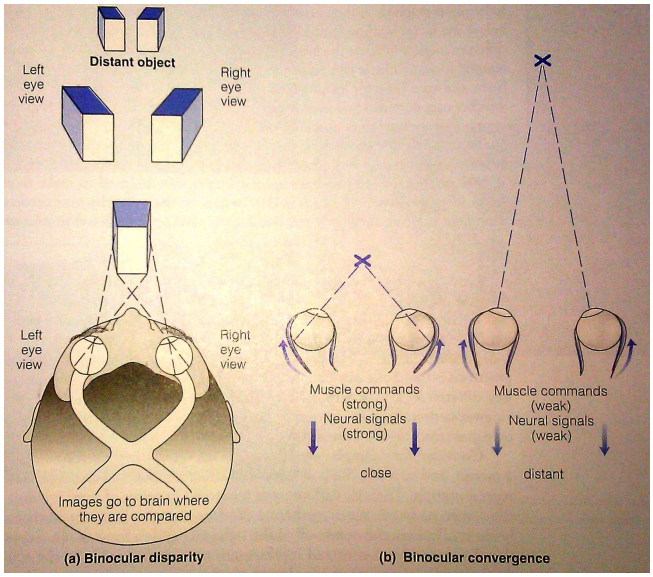

Figure 1: Two cues that play a central role in depth perception. Source: [25]

Furthermore we classify objects, according to a widely supported theory in psychology, using feature-matching [4] [25]. We do this by matching discriminative features of an object to feature sets stored in our memory. In this theory a retinal image is passed on to a set of feature demons, which proces for example horizontal/vertical lines or right angles in the image. In a next level of processing, decision demons are activated if a combination of these features is detected. An example of the perception of a letter R is illustrated in Figure 2.

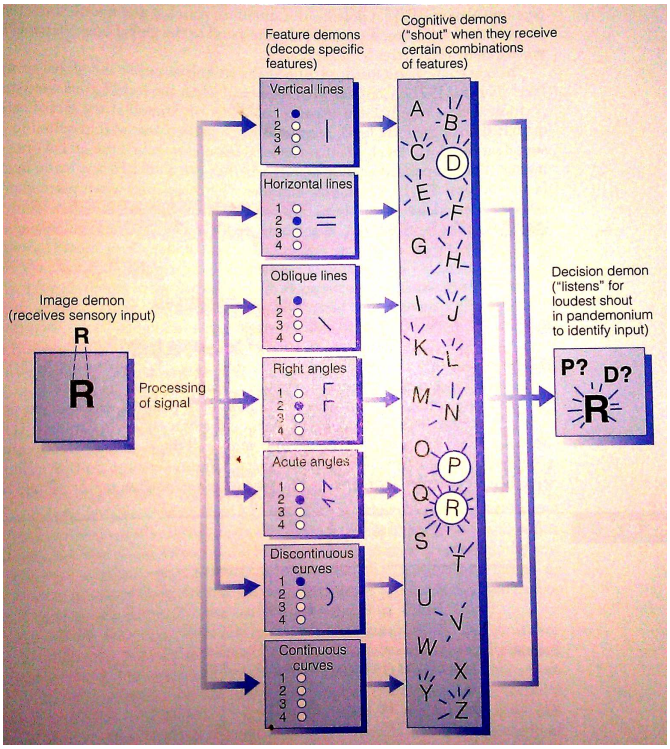

Figure 2: Matching observed features to feature sets already stored in memory. Source: [25]

.
These visual processes are extremely informative if we want to build a computer system that acts accordingly. This thesis is about our work of a system that adds semantics to an urban scene inspired on the human brain.

We use stereopsis to generate a 3D model of the building. And feature-matching, similar to the discussed model, is applied to detect skylines and windows where we use descriptors based on edge features like straight lines and right angles. Before we explain our methods let us first share the variety of applications that use semantical interpretation of urban scenes.

1.1 Application examples

3D City models

Manual creation of 3D models is a time consuming and expensive procedure. Therefore semantic models are used for semi automatic 3D reconstruction/modelling. The semantic understanding is also used in 3D city models which are generated from aerial or satellite imagery. The (doors and) windows are mapped to the detected 3D model to increase the level of detail [16]. Some other applications can automatically extract a CAD-like model of the building surface.

Historical buildings documentation and deformation analysis

In some fields of research, historical buildings are documented. The complex structures that are contained in the facades are recorded and reconstructed. Another field of research is the analysis of building deformation in areas containing old buildings [22]. Window detection provides information about the region of interest that could be tracked over time for an accurate deformation analysis.

Interactive 3D models

There are some virtual training applications that are designed for emergency response requiring interaction with a 3D model. For the simulation to be realistic it is important to have a model that is of high visual quality and has sufficient semantic detail (i.e. contains windows). This is also the case for a fly-through visualization of a street with buildings. Other applications that require semantic 3D models are virtual tourism, visual impact analysis, driving simulation and military simulation systems.

Augmented reality

Some mobile platforms apply augmented reality using facade and window detection to make an accurate overlay of the building. An example overlay is the same building but 200 years earlier. Semantical information is used to not only identify a respective building, but also find his exact location in the image. The accuracy and realistic level of the 3D model are vital for a successful simulation. And because the applications are mobile, very fast building understanding algorithms are required. Window detection plays an important role in these processes as the size and location of the windows supply an effective descriptor that can be used for robust and fast building identification. Furthermore it provides an accurate alignment of the overlay.

Building recognition and urban planning

Building recognition is used in the field of urban planning where the semantic 3D models are used to provide important references to the city scenes from the street level. Building recognition is done by using large image datasets where

Figure 3: Simulation environment

the buildings are mostly described by local information descriptors. Some approaches try to describe the 3D building with laser range data. Some methods fuse the laser data with ground images. However, those generated 3D models are a mesh structure which do not make the facade structure explicit. For a more accurate disambiguation, other types of contextual information are desired. The semantical interpretation of the facade can provide this need. In this context, window detection can be used as a strong discriminator.

We can conclude that semantic interpretation plays an important role in the interpretation of urban scenes and is applied in a wide range of domains.

1.2 Thesis outline

The outline of this thesis is as follows:

We start with explaining basic computer vision techniques and the *FIT3D toolbox* in Chapter 2. These techniques are the driving force behind the algorithms used in both skyline detection and window detection. In Chapter 3 we explain a novel application of skyline detection: the detection of building contours in urban scenes. Next, we use this result to extract a 3D model of a building in Chapter 4.
In Chapter 5 we start a new topic: window detection. First we propose a window detection method that operates on an unrectified facade. The second method uses a rectified facade. We discuss and compare two window alignment and classification methods. We conclude in Chapter 6 and we finish with additional results in the Appendices.

Many methods used in this thesis are independent from each other. Therefore we choose to discuss the results, discussion and future research for each method separately. Also the chapters are independent from each other. Therefore, a reader only interested in one particular topic may read only the associated chapter while skipping the other chapters.

2 Preliminaries on Computer Vision

In this chapter we discuss the basic computer vision techniques that are used for the skyline detection, and window detection. Furthermore we discuss 3rd party software, the *FIT3D toolbox* [11] which is used for 3D building extraction and facade rectification.

2.1 Hough transform

2.1.1 Theory

A widely used method for extracting line segments is the Hough transform [10]. In the Hough transform, the main idea is to consider the characteristics of a straight line not as its image points $(x1, y1)$, $(x2, y2)$, but in terms of the parameters of the straight line formula $y = mx + b$. i.e., the slope parameter m and the intercept parameter b.
The Hough transform transforms the line $y = mx + b$ to a point (b, m) in parameter space. With this representation it is impossible to describe a vertical line as the slope m is infinite. Therefore it is better to use a different set of parameters, denoted r and θ . These are the Polar Coordinates.
The parameter r represents the distance between the origin and the line and θ is the angle of the vector orthogonal to the line. Using this parameterization, the equation of the line can be written as

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

$$r = x\cos\theta + y\sin\theta$$

This means that a point in (x, y) space appears as a sinusoidal curve in the Hough parameter (r, θ) space. Furthermore a line in (x, y) space appears as a point in (r, θ) space.
Let's see an example, the image of Figure 4 is transformed into the space (r, θ).

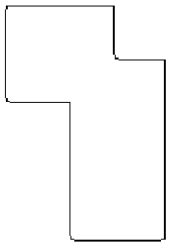

Figure 4: An input image, consisting of eight straight lines, for the Hough transform

(a) (r, θ) values

(b) (r, θ) accumulator array (quantized)

Figure 5: Hough transform

As you can see for every edge point in Figure 4 a curve is generated in (r, θ) space in Figure 5(a). On eight positions (dots) the number of intersecting sinusoidal curves is high. These position correspond to the eight separate straight line segments in Figure 4 .
This simplifies the problem of detecting straight lines in finding peaks in the Hough parameter (r, θ) space.

2.1.2 Implementation

The input of a Hough transform is a binary image. In our research it is the output of the skyline detector (Chapter 3). In the case of window detection (Chapter 5) it is the output of an edge image.

The Hough transform develops an accumulator array of a quantized parameter space (r, θ). It loops through the binary image and for each positive value it generates all possible lines, quantized (r, θ) pairs, that intersect with this point. For each candidate it increases a vote in the accumulator array. Lines (r, θ) that receive a large amount of votes i.e. the dots in Figure 5(a) are the found straight lines in the (x, y) space. These positions are found by looking for local maxima in the accumulator array.

2.1.3 θ constrained Hough transform

The accumulator array consist of two dimensions r and θ. θ typically ranges from $[-90..90)$ resulting in 180 unique bins. Note that a line with $(r, \theta) = (t, j)$ can also be represent by the identical $(-t, j - 90)$, e.g. $(4, 135) == (-4, 45)$. This makes it possible to represent every line with the interval $[-90..90)$.

Sometimes we want to find lines that have a certain angle. For example the skyline of a building will appear about horizontal. If we want to detect windows we would like to detect edges in the horizontal and vertical directions. This can easily achieved by adjusting the θ range. For example if one would detect lines in the horizontal direction, $\theta = [-10..0..10]$. Intervals are used because in practise the lines often differ slightly from an exact horizontal line where $\theta = 0$.

2.1.4 MATLAB[15] parameters

We used a standard MATLAB[15] implementation of the Hough transform. This implementation comes with some interesting parameters:

The *MinimumLength* parameter specifies the minimum length that a line must have to be valid. This is especially interesting if we want to detect a large straight skyline or if we want to discard lines that are to small to form for example a window.

Furthermore it contains the parameter *FillGap* that specifies the distance between two line segments associated with the same (r, θ) pair. When this inter

line segment distance is less then the *FillGap* parameter, it merges the line segments into a single line segment. In our application this parameter is of particular interest when we want to merge lines that are interrupted by for example an occluding tree or street lamp.

2.2 Coordinate systems

In this thesis three different coordinate reference systems are used. The first one is located at the image level and describes the location of the pixel in 2D. It is called the Image Coordinate Frame (ICF), in Figure 6 the ICF is spanned by (x_0, y_0). The second is the Camera Coordinate Frame (CCF), the system is aligned with the camera: the origin is fixed at the the camera center (center of projection) and the CCF is rotated with a rotation vector that is equal to the camera's viewing direction. The coordinates are expressed in (XYZ).
The last system is the World Coordinate Frame (WCF) and it is used to describe the positions of the 3D model of the scene and the position of the cameras in the world in (ijk). An overview of the systems can be found in Table 1.

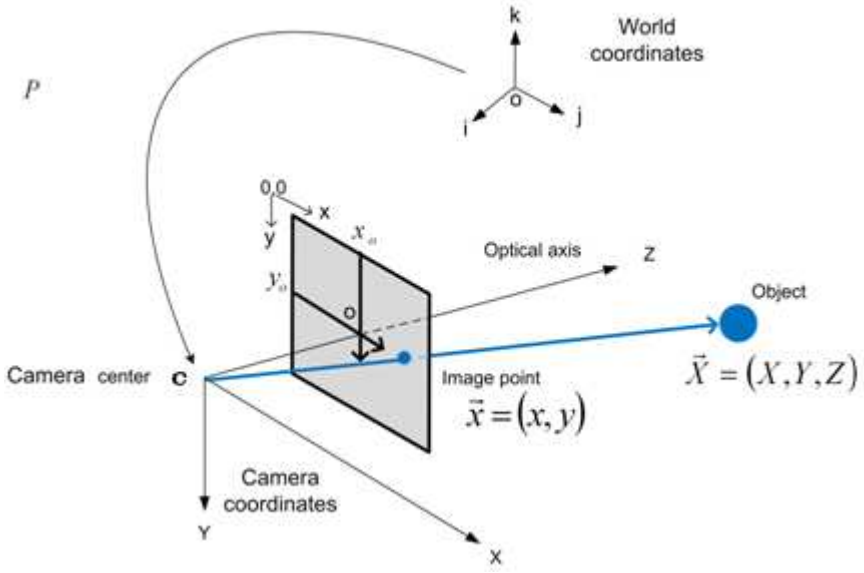

Figure 6: Coordinate systems

2.3 Camera calibration

Camera calibration is done to determine the relationship between what appears on the image (or camera sensor) and where it is located in the 3D world. This involves calculating the camera's intrinsic and extrinsic parameters.

Table 1: Coordinate reference systems with their properties

Reference system name	abbreviation	dimensions	axis
Image Coordinate Frame	ICF	2D	x,y
Camera Coordinate Frame	CCF	3D	X,Y,Z
World Coordinate Frame	WCF	3D	i,j,k

Intrinsic parameters

The intrinsic parameters contain information about the internal parameters of the camera. These are focal length, pixel aspect ratio, and principal point.
These parameters come together in a calibration matrix K. The Floriande dataset (originated from the Fit3D toolbox [11]) comes with a calibration matrix K. For the other datasets we calculated K with the Bouguet toolbox [5]. This method involves taking images of a checkerboard in different positions and orientations. An algorithm detects the grid on the chessboard and monitors its transformations under the different images. If enough images are given (at least 10) and the images contain enough variety in chessboard pose, the Bouguet toolbox can calculate the intrinsic parameters quite accurate. More details about this method can be found in [5].

Extrinsic parameters

The extrinsic parameters present the center of the locations of the camera and the camera's rotations. These are unique for every view. In some systems these are recorded by measuring the cameras location and rotation at the scene. In other systems this is computed afterwards (from the images). We calculated the values also afterwards and used existing software for this: *FIT3D toolbox* [11], details of this process is explained next.

2.4 FIT3D toolbox [11]

The *FIT3D toolbox* [11] is used for several aims in this thesis. It is used in the window detection module to rectify the facades. and the skyline detection used *FIT3D* to extract a 3D model.
In order to extract this 3D model a series of frames (originating from different views) is used to estimate the relation of the camera coordinates to the world coordinates, Next, the result is used to extract a point cloud of matching features. Finally this point cloud is converted to planes which correspond with the walls of the building.
Because the toolbox plays an assisting role we explain the steps briefly. Detailed knowledge about the methods can be found in [11].

2.4.1 Multiple views

FIT3D uses multiple views to gather information about the 3D structure of the building. The toolbox comes with a prepared dataset of 7 consecutive images (steady (zoom, lightning, etc.) parameters) of a scene. *FIT3D* doesn't require the input images to be chronological however they need to have sufficient overlap.

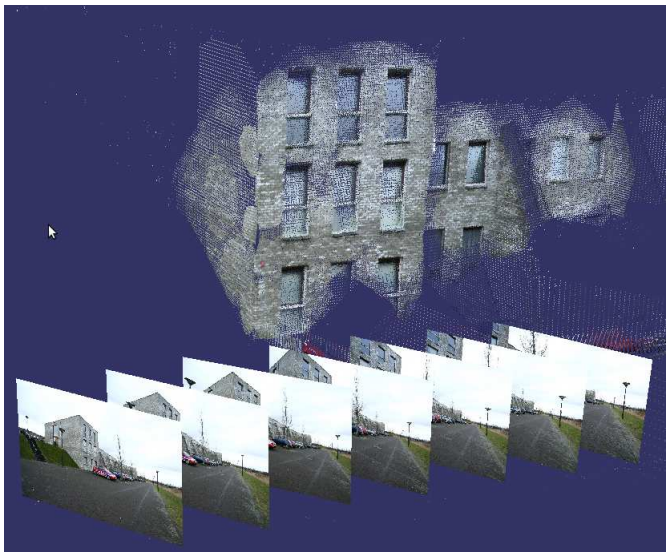

Figure 7: Example series of 7 consecutive frames, (dataset: *FIT3D toolbox*[11])

2.4.2 Relating the camera coordinates to the world coordinates

The different views are used to estimate the relation of the camera frame coordinates to the world coordinates, (the extrinsic parameters). In other words the different positions of the camera centers (from the origin of the WCF to the camera center) and the camera's rotation are estimated. This is done by calculating the relative motion between the different views.

The frame to frame motion is calculated by extracting about 25k SIFT features of each frame. Next, SIFT descriptors are used to describe and match the features within the consecutive frames. Not all features will overlap or match in the frames therefore RANSAC[2] is used to robustly remove the outliers. After this an *8-point algorithm[12]* together with a voting mechanism is used to extract the relative camera motion.

The frames are matched one by one which returns an estimation of the camera motion. Because this estimation is not accurate enough, a 3-frame match is

applied next. This result is more accurate but comes with a certain amount of re-projection error which is minimized using a numerical iterative method called *bundle adjustment[26].*

From every frame the camera motion is stored relative to the first frame. The motion is stored as a rotation and translation form (3x4) $[R|t]$. This is gathered for all 7 frames in a (7x3x4) projection matrix P. P can be used to translate camera coordinates of a specific view to 3D world coordinates and vice versa.

2.4.3 3D point cloud extraction

The next step is to use this projection matrix P to obtain a set of 3D points. which correspond to the matching SIFT features of the different views. The set of 3D points is extracted using a *linear triangulation[12]* method.
The 3D point cloud of the building is illustrated in Figure 7. After obtaining the point cloud, a RANSAC based plane fitter [2] is used to accurately fit planes through the 3D points, see Figure 8.

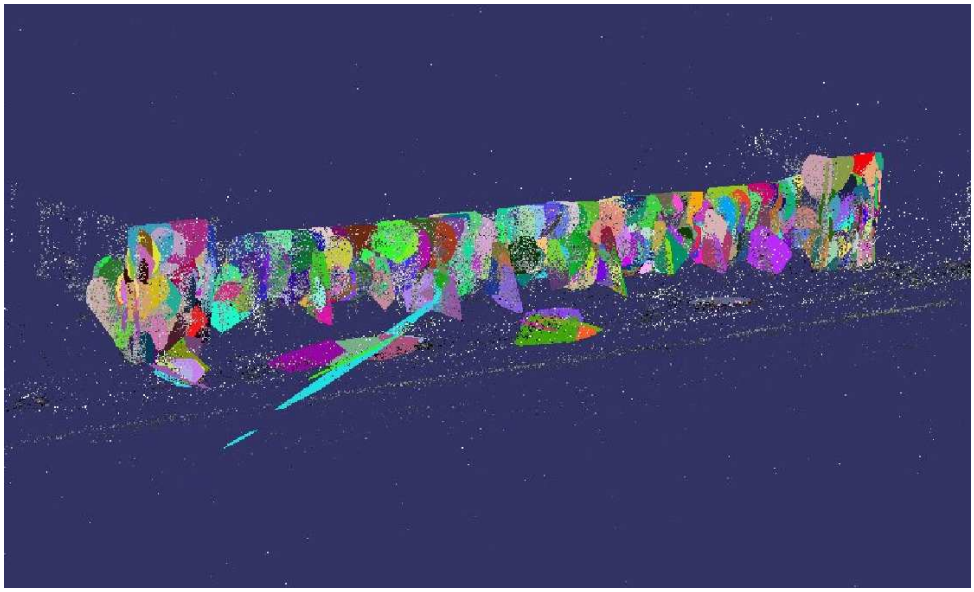

Figure 8: Fitted planes

In this thesis the plane fitting step is skipped and a new method is used to obtain the 3D model. This is explained in 4.

3 Skyline detection

3.1 Introduction

If we take a regular image on which both sky and environment are present, there is often a clear separation between them. This separation is called the skyline. The detection of this skyline has proven to be a very successful computer vision application in a wide range of domains ranging from object detection[8], guiding micro air vehicles[18], car localization, etc.
In this research skyline detection is applied on different views of a scene to estimate heights of facades of a 3D model. This is a novel application for skyline detection.

The organization of this chapter is as follows: first we give a summary of related work on skyline detection. Next we explain how we developed a new robust skyline detection algorithm. Then we present and discuss some results and, finally, conclusions are given.

3.2 Related work

Castano et al. [8] present a clear introduction of different skyline detection techniques.

Detection of dust devils and clouds on Mars

In [8], mars Exploration Rovers are used to detect clouds and dust devils on Mars. Their approach is to first segment the sky from the ground and then determine if there are clouds in this region. The sky is detected by an innovative algorithm that is similar to the one of Cozman et al. [9]. This time the variation in intensity is used to discriminate the sky from the ground. The algorithm uses a sliding window that slides from top to bottom. If looks for an location where the intensity variation is high. This location is classified as a skyline point. The sky is segmented by taking the area above these skyline points. Next, this segmented sky is used for the detection of clouds and dust devils.

This method looks like a very sophisticated one, as it is accurate and autonomous. However, in our research we have a stable scene with sharp edges at the building contour so this method would be an implementation overkill.

Horizon detection for Unmanned Air Vehicles

In the domain [18] of unmanned vehicles, scientists detect the horizon to stabilize and control the flight of Unmanned Air Vehicles.
S.M. Ettinger et all [18] use a horizon detector that takes advantage of the high altitude of the vehicle, in that way the horizon is approximated to be a straight line. This straight line separates the image into sky and ground. They use color

as a measure of appearance and generate two color distributions: one for the sky and one for the ground. They use the covariance and the eigen values of the distributions to guide a bisection search for the best separation. The line that best separates the two distributions is determined to be the skyline.

This work is not applicable for detecting a building contour as the straight line assumption doesn't apply. But it needs to be mentioned that some ideas for section 3.7.2 are inspired by this method.

Planetary Rover localization

Cozman et al. [9] use skyline detection in planetary rovers to estimate their location. To recover the rover's position they match image structures with a given map of the landscape (hills, roads, etc). One of the matching image structures they use is the shape of the skyline. First a ground truth skyline model m is obtained using a full 360 degree panorama of the scene. Next, a matching function is applied to the skyline observations and m to estimate the rovers location, see Figure 9.

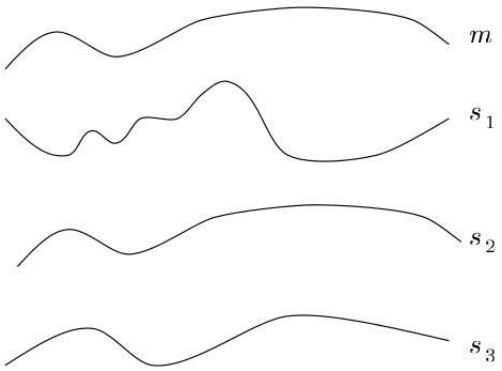

Figure 9: By matching map m to the observations s_1, s_2 and s_3 the position of a planetary rover is estimated

The skyline is detected using the fact that at a large change in intensity occurs at skyline points. This is obviously because the color changes from sky to ground. For every column of the image a search is done for a high intensity change. The largest change is determined to be the skyline.

The advantage of their algorithm is the simplicity and effectiveness, this could make their algorithm suitable for this project. A big drawback is that they prefer speed over accuracy. To increase accuracy, the detector is part of an

interactive system where an operator refines the skyline. For our application the skyline detector must operate without any user interaction. This brings us to our research question.

3.3 Method

Research question

Can we build a skyline algorithm that operates without any user interaction, is simple, is fast, and yet accurate enough to provide a solid skyline that can be used to estimate wall heights?

3.3.1 Situation and assumptions

As the Rover method [9] is simple and effective we used it as a basis and build a custom algorithm with higher accuracy on top of that.

Situation

Before we present the method, we define the situation and make some assumptions.

Definition: skyline in urban scene

A skyline in an urban scene is a set of points of the size w (where w is the width of the image) where each point describes the location of the transition from the sky to an object (e.g. a building) which is connected to the earth.

How are we going to detect this sky-building transition point?
In general, the color of the sky is very different from the color of the building. A color-based edge detector would be an intuitive decision as this produces edges on regions where intense color transitions appear. However, the sky and the building itself also contains color transitions (caused by for example clouds and windows). So how do we determine the right transition (edge)?

One of the solutions is to increase the threshold of the edge detector. In this way the detector will only return intense color transitions. Note that this will only pay off if the building-sky transition is the biggest transition in the image. Its easy to see that this is a tricky assumption as other objects may contain larger color transitions. Furthermore it would not be robust to a change in the illumination conditions, influenced heavily by the weather.

To solve this problem we draw an assumption that is based on the idea of [9]. Instead of using the sharpest edge we take the most upper sharp edge and classify this edge as the skyline.

Top sharp edge assumption

The first sharp edge (seen from top to bottom) in the image represents the skyline.
Having defined the situation and assumptions we now explain our algorithm.

3.3.2 Related algorithm

As our algorithm is based on a related algorithm presented in [9], this is in detail described first.
The algorithm uses three main steps first it applies a smoothing preprocessing step then it calculates the intensity gradient to find a big color transformation and finally it searches for the highest transformation.
Because the related algorithm searches for strong edges a preprocessing step is applied to remove all vague edges. The pre-processing consist of a sizable Gaussian smoothing window that is applied on the input image. The size of the window size is correlated to the amount of vague edges that are removed.
Next the smoothed image is sliced in $\#w$ pixel columns. Each column represents the $\#h$ intensity values of the image (where w and h are the width and height of the input image). These columns are transformed to their derivative, called the smoothed intensity gradient. The values of this transformed column are high when a big change in color happens (e.g. when an edge is detected) at that location on the image.

Next the system walks through the values of a column, starting from the top. When it detects a value with a gradient higher then a certain threshold it stores its y-position (the height) and continues to the next column. After the position in each column of the highest sharp edge is determined the algorithm is done. The result is a set of y coordinates of length w, that represent the skyline.

3.3.3 Improved algorithm

Taking the smoothed intensity gradient is a computational cheap way to detect edges. It also has a big disadvantage because it is not robust to vague edges (they don't survive the threshold). It is not surprising that the algorithm in [9] was used in an interactive system where the user has to refine the result.

Our aim is to develop an autonomous skyline detector, the only user interaction that we allow is to set some parameters of the system. Furthermore the vague edges need to be detected if they are part of the skyline. We will now discuss the adaptations that we developed with respect to the related algorithm.

The column based approach of the related algorithm seems to be very useful and is therefore unchanged. To be robust to vague edges we explored and tested edge detecting methods that are different then the smoothed intensity gradient based method.

The output of the different edge detection techniques was studied on an empirical basis and the Canny edge detector [7] was a clear winner. This is probably because Canny is a more advanced edge detector:
It uses two thresholds, one to detect strong and one to detect weak edges. It starts with the strong threshold and when it finds an edge it calculates his direction. This direction is used to search for weak edges that are connected at one of the sides of the strong edge. In this way noisy weak edges are discarded because they are not connected to a strong edge. Furthermore it doesn't discard the vague edges that are desired. In Table 2 we list MATLAB[15]'s built-in edge detectors together with the method explanation. In the section 3.4.1 one can find the results of the different edge detection methods.

Table 2: Different edge detectors explained, Source: MATLAB[15] Documentation

Name	method
Sobel	The Sobel method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of the image is maximum.
Prewitt	The Prewitt method finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of the image is maximum.
Roberts	The Roberts method finds edges using the Roberts approximation to the derivative. It returns edges at those points where the gradient of the image is maximum.
zero-cross	The zero-cross method finds edges by looking for zero crossings after filtering the image with a filter that has to be specified.
Laplacian	The Laplacian of Gaussian method finds edges by looking for zero crossings after filtering the image with a Laplacian of Gaussian filter.
Canny	The Canny method finds edges by looking for local maxima of the gradient of the image. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds, to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

We classify Canny as the most robust edge detector, and use it for the skyline detection algorithm: the Canny edge detector outputs a binary image, therefore the column inlier threshold is set to 1, which means that it finds the first pixel that is white. This is, as in the related algorithm, done from top to bottom for every column in the image.
Because we know we are looking for sharp edges, we improved the algorithm by introducing two preprocessing steps. First the contrast of the image is increased, this makes sharp edges stand out more. Secondly the image undertakes an extra

Gaussian smoothing, this removes a large part of the noise.
The system now has several parameters which have to be set manually by the user:

- Contrast
- Window size of Gaussian smoothing
- Edge detector threshold

If the user introduces a new dataset these parameters need to be configured as the image quality and illumination conditions are scene depended.

3.4 Results

3.4.1 Edge detection

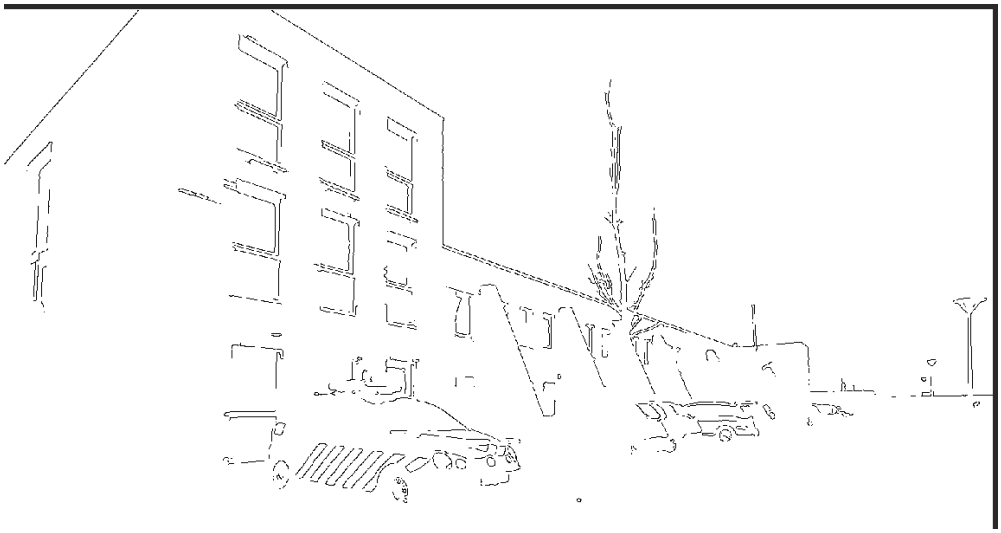

Figure 10: Edge detection results. Method: Canny

The edge detection results of the other methods can be found in the appendix (A.2).

3.4.2 Skyline detection

Datasets

The skyline detection algorithm was applied on three datasets.

Table 3: Dataset properties

Name	Resolution	Source	Location
Floriande	1728x1152px	FIT3D [11]	Unknown
Bram	3072x2304px	Author	Amsterdam, 'Postjesweg' west side
Folkert	3072x2304px	Author	Amsterdam, 'Postjesweg' east side

Figure 11: The output of the skyline detector on the *Floriande* dataset. The skyline elements are marked red.

Figure 12: The output of the skyline detector on the *Bram* dataset. The skyline elements are marked red.

Figure 13: The output of the skyline detector on the *Folkert* dataset: a scene which violates the top sharp edge assumption. The hanging streetlight causes the detection of a sharp edge above the building. This results in a damaged skyline.

3.5 Discussion

Consider Figure 11, the largest part of the building edge is detected. This is a good result, given the algorithm operates without any user interaction.

We assumed that the first sharp edge (seen from top to bottom) in the image represents the skyline. We showed in Figure 13 that the first sharp edge is not always the skyline. This holds for more scenes: e.g. Amsterdam contains a large amount of hanging street lights. Furthermore, other sharp edged objects that appear above the building, e.g. tree's or even an aircraft, will also produce a scene that violates the top sharp edge assumption. Therefore it would be nice to relax the or discard the first sharp edge assumption. This implies that we have to extend the column based approach which is done theoretically and described in Future research (3.7).

3.6 Conclusion

Let's answer our research question. *Can we build a skyline algorithm that operates without any user interaction, is simple, is fast, and yet accurate enough to provide a solid skyline that can be used to estimate wall heights?*

Beside some scene dependent parameters (like the threshold of the skyline) the system works without any user interaction. No manual refinement step is needed because the algorithm is robust and accurate enough to provide a base for the next module in the system. Furthermore the algorithm is simple and has a low complexity.

It is interesting to point out that the skyline detector is a stand alone method and it can be optimized individually without any knowledge of the other modules of this project.

Because the top sharp edge assumption closes the door for a large amount of scenes, an alternative is desired. We designed a concept that relaxes this assumption which is shared next.

3.7 Future research

3.7.1 Automatic thresholding

As the threshold that decides whether an edge is strong enough to represent the skyline is manual and scene dependent. Therefore a method for automatic thresholding is desired. There exist many studies on this topic, most of the methods are based on the statistical analysis of the image. Detailed literature research needs to be done and an implementation must be made to provides a value for the threshold. If this is done the system will operate 100% automatic.

3.7.2 Hypothesis based skyline detection

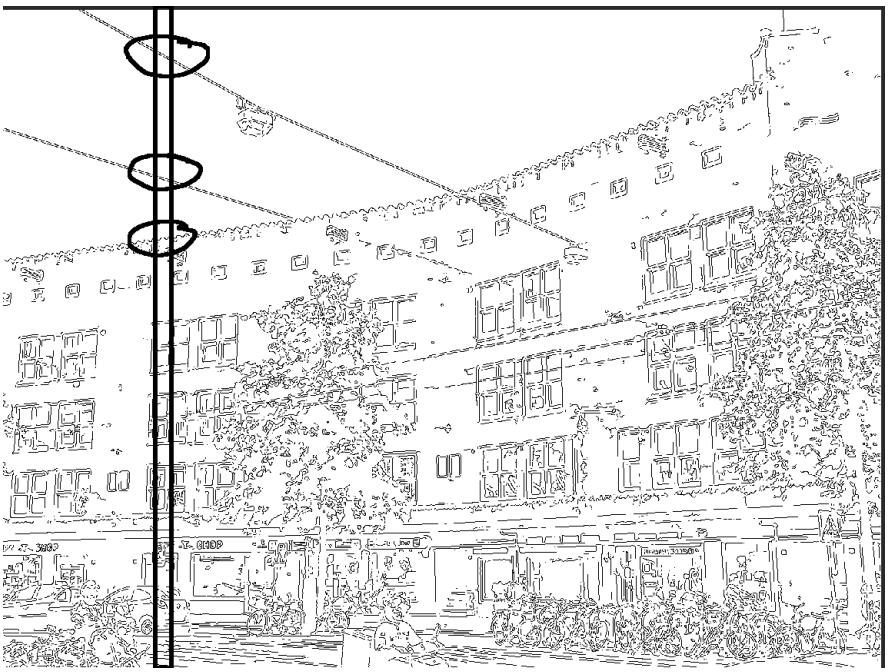

Figure 14: Three top sharp edges of the *Folkert* dataset are highlighted. For the highlighted column the skyline happens to be the 3rd top sharp edge.

The first top sharp edge in Figure 13 is on a large area a hanging street light instead of the skyline. If we take a look at the edge image in Figure 14 we observe that the skyline in this area is the second or third sharp edge. Let's relax the first top sharp assumption to an assumption where the skyline is part of the first n sharp edges (seen from top) with for example $n = 3$. The scene in Figure 13 now agrees with this assumption.
The next challenge is to do detect the skyline properly on the entire image. The existing column based approach could be used to generate n hypotheses and we can build an algorithm on top of that to classify which hypothesis is right (i.e. which edge presents the skyline).
The hypothesis classifier must gather additional information. We could discriminate the hypothesis for example by texture, color distributions or height variation. We discuss the last two.

Color based skyline classification

The buildings color differs from the sky color. Furthermore, as a function of the horizontal location (x-axis), the color in both sky and building only changes slowly.

One could extract the color distribution of the color in a certain area above and under the skyline, let's call these A and U.
A and U should 1) have a significant difference (local) and 2) they should not vary much from other positively classified skyline points (global).

The global color distribution could be calculated by taking the mean of the color distributions at previously detected skyline points. Both conditions (local and global) could have a weight which would be a parameter of the system.

Let's test this conceptual algorithm on the scene in Figure 13. We initialize the algorithm at an interesting location which is highlighted in the figure.
The first (top) highlighted sharp edge doesn't have a large local difference in color distribution above and under the skyline. The color above and under the position are mostly white. Because the first condition fails and the second condition can't be checked as no global color distribution is stored yet, the algorithm continues with the next edge.
For this edge the same holds. The third edge, however, succeeds because it has a large local color difference. Therefore the algorithm classify it as a skyline point. Furthermore it stores his color distribution to compare following skyline points.
Next the algorithm takes for example one x location to the right. It will again agree on the third edge on the same terms. This time it rejected the outliers (first and second edge) with additional because their color distributions differ to much from the color distribution of the previously detected skyline point.

The expected output of the skyline detector using hypothesis based on color is displayed in Figure 15).

Height variation based skyline classification

Let's zoom in to an image of the *Floriande* dataset, see Figure 16. We observe that many outlying skyline points are located at the tree. The algorithm based on the first n sharp edges wouldn't be robust because the tree produces a too large amount of edges. Therefore an alternative classification is desired.

The skyline points on the tree have a large height variation in common. This could be used to isolate this area from the skyline.
Once areas that have a large variation of height are discarded we could connect the positive classified skyline parts to fill the gap, see Figure 17.

Note that we used height variation instead of height difference for a reason. A

Figure 15: The expected output of the skyline detector using hypothesis based on color

large height difference with a neighboring skyline point is allowed because the building contour may contains large steps in height (e.g. at the left side of Figure 16 the skyline suddenly drops in height).

We showed some examples of simple classifiers. Note that there is no limitation to the approach of the classifier because it can be developed individually using the existing column based method as its hypothesis generator.

Figure 16: Zoomed image of the *Floriande* dataset. The height variation is large on the tree area.

Figure 17: The green lines connects the positively classified skyline parts at isolated areas where the height variation is too high

4 Extracting the 3D building

4.1 Introduction

In the previous chapter we explained our skyline detection algorithm which extracted the skyline of a scene. The output is a set of 2D points which was collected for a sequence of images. The aim of this Chapter is to use this set of points together with an aerial 2D model of the building and the 3D point cloud obtained by the *FIT3D toolbox[11]* (2.4) to generate a 3D model of the building.

Research question

Is it possible to use a set of (noisy) skyline points together with an aerial 2D model and a 3D point cloud obtained by the *FIT3D toolbox[11]* to generate a 3D model of the building?

We present a stepwise solution: first *Openstreetmap[1]* is used to obtain a 2D top view of the outline of the building. The line segments of the 2D model represent the walls of the building. Next the 3D point cloud obtained by the *FIT3D toolbox[11]* is used to align this 2D model in the scene. After this, the set of points returned by the skyline detector is transfered to a set of lines. Then each line segment is assigned to a part (wall) of the aligned 2D model. Next the line segments are projected to vertical planes spanned by the 2D model. The result is used to estimate the height values of the walls of the 2D model. The 2D model is transformed according to this height values to a 3D model. We will now elaborate on each step.

4.2 Method

4.2.1 Extracting the 2D model

The basis of the generated 3D model is a map containing the 2D outline polygon of the building originated from *Openstreetmap[1]* (Figure 18) which is a freely accessible 2D map generated by users all over the world. It contains information about streets, building contours, building functions, museums, etc. We are interested in the building contours therefore we take a snapshot of a particular area and extract this building contour. This is a set of ordered points where each point corresponds to a corner of the building. Next we link these points to line segments which represent (the top view of) the walls of the building.

4.2.2 Aligning the 2D model

We want to align the 2D model in the scene which means that we have to position the 2D model in the world coordinate using a translation, rotation and scaling.

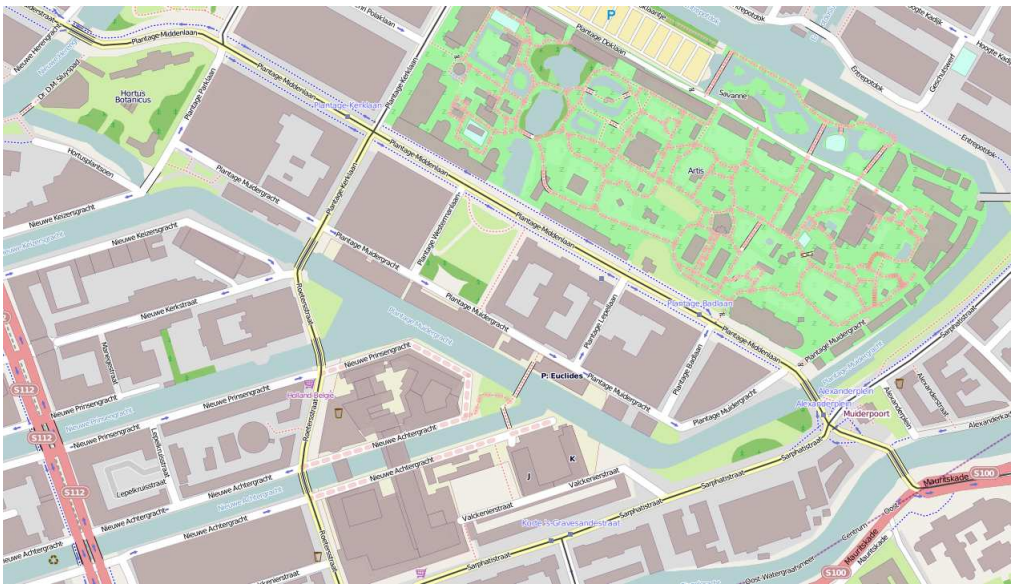

Figure 18: Openstreetmap[1] with annotated buildings

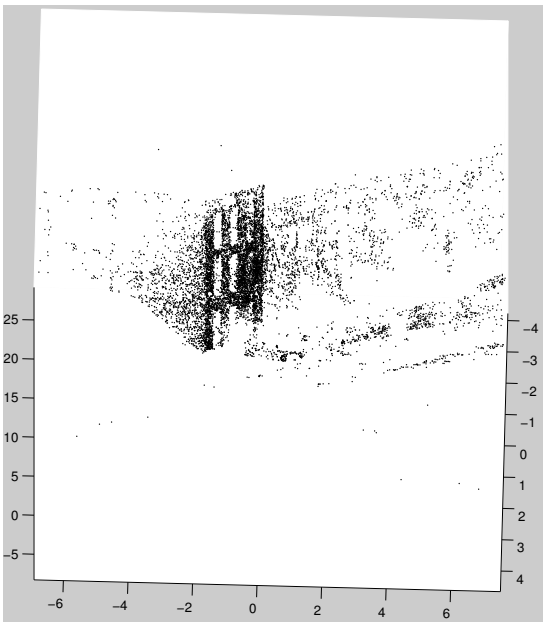

Figure 19: 3D point cloud of the walls of the building

From *FIT3D[11]* we have the 3D point cloud of the building in world coordinates

(Figure 19). The first challenge is to obtain a top view of the 3D point cloud. How can we determine the direction of the top view? We want to project in the direction that is most parallel to the walls and orthogonal to the 2D model.

Gravity aligned walls assumption

The walls of the building are aligned with the gravity direction (y-axis) which is orthogonal to the 2D model from Openstreetmap[1]. This means we assume the images are taken upright: the camera's $roll = 0$ (Figure 35).

Now we have defined the wall direction, we obtain the top view of the 3D point cloud by discarding the y-dimension of the points. Note that this is equivalent to a projection to the x,z plane. The result is a set of 2D points that represent the top view of the 3D point cloud (Figure 21).
We determine the walls by fitting line segments in the point cloud. We annotated these line segments manually. If the system needs to operate automatically, RANSAC can be used to fit lines in this point cloud.

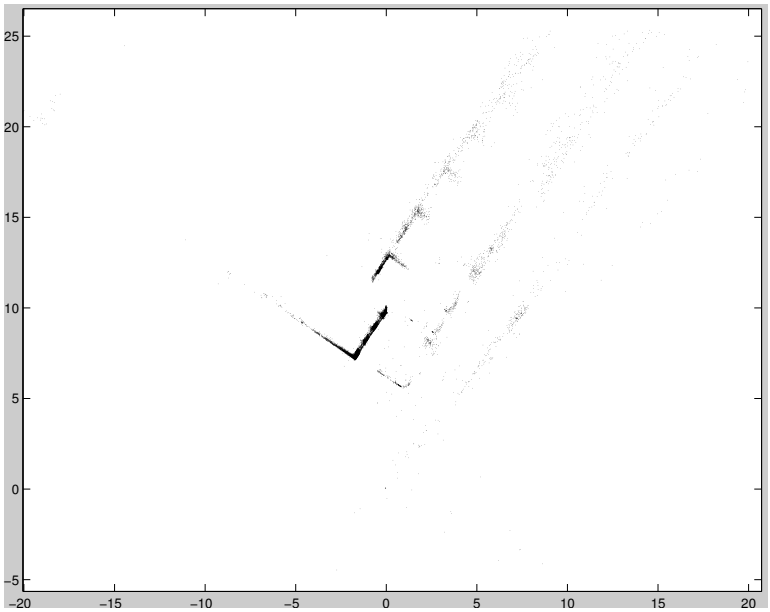

Figure 20: The projected 3D point cloud of the walls of the building

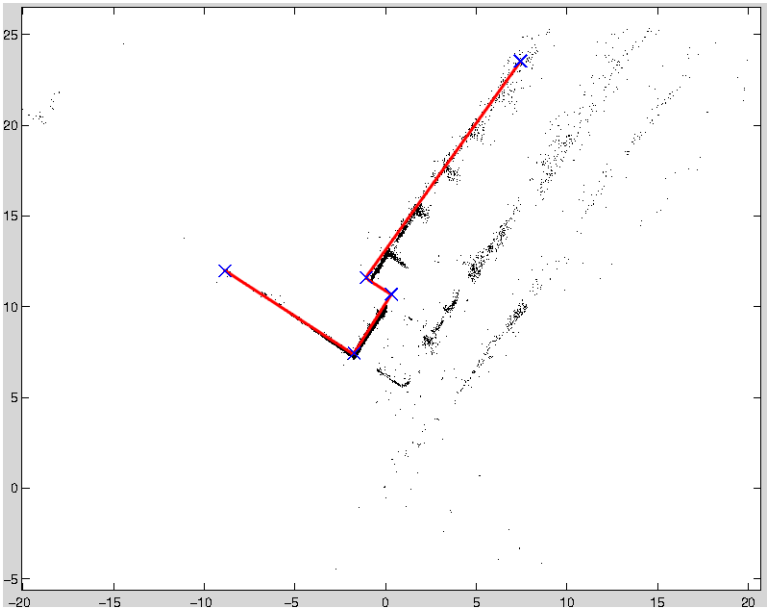

Figure 21: M2, The fitted line segments define (a top view of) the building walls

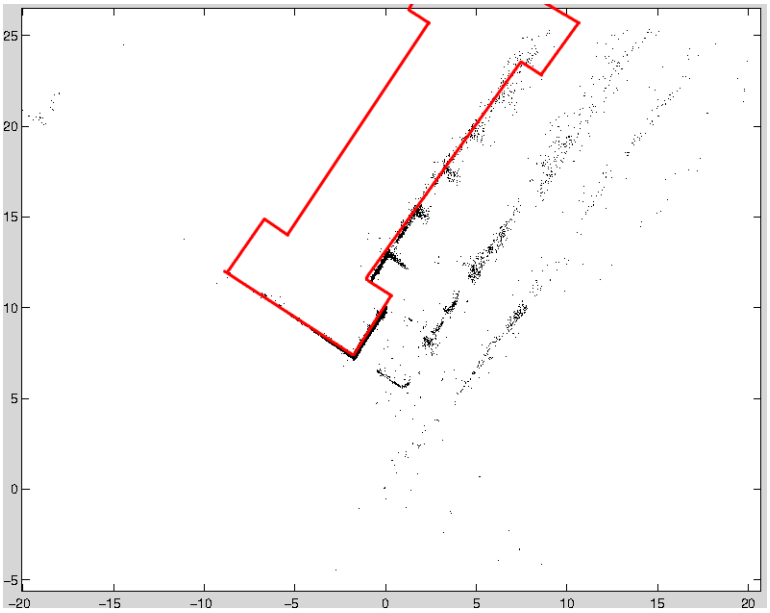

Figure 22: The 2D model M1 aligned with the fitted line segments (M2)

The next step is to align the 2D *Openstreetmap[1]* model $M1$ with these line segments of the projected point cloud $M2$ in the world coordinate frame. First the endpoints of the line segments, which correspond to the wall corners, are extracted in both models. Note that the walls of $M2$, that are located at the back of the building, are missing because they are occluded by the front walls. We only consider the corners that are present in both models.
Next the correspondences between these corners is annotated manually. This is used to generate a set of linear equations which are solved in closed form [12]. The result is a matrix A which represents the rotation, translation and scaling that is needed for a coordinate of $M1$ to be transformed to $M2$ in the world coordinate frame. Finally A is applied on all corner points of $M1$ which results in an aligned 2D model (Figure 22).

4.2.3 Transferring the aligned 2D model to 3D

Because the 2D model is based on aerial images it contains no information regarding the height of each wall. In the next section we explain how we obtain the precise height values.

For the sake of presentation we use an average building height to generate a rough estimate of the 3D model. The 3D model is generated by taking the 2D model and extend it in the orthogonal direction. An example of the 3D model can be seen in Figure 23.

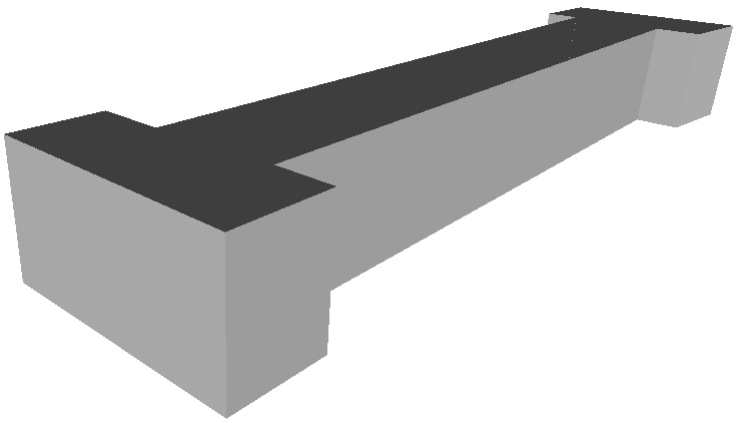

Figure 23: The basic 3D model, generated by extending the (2D) model from *Openstreetmap[1]* to an average building height

4.2.4 Extracting line segments

Because we want to estimate the height of the building walls of the 3D model, we need to know how high the walls in the images are. To estimate this, we first determine which part of the skyline is part of the building contour. We use the idea that straight lines in the skyline area are likely to come from the building contour. In this section we explain how we extract these straight line segments.

Assumptions

Many urban areas contain buildings with a flat roof. Therefore the contour of the building is always formed by a set of straight line segments. Furthermore the building contour is often aligned with the topside of a building wall. If we assume a flat roof, we can use the skyline to estimate the height of the building walls without having to concern for (complex) roof types.

Flat roof assumption

We assume each building has a flat roof, implicating that the building contour is aligned with the topside of a building wall. The building walls may have different heights but the roof should be flat.

Hough transform

As was discussed in Chapter 2, a widely used method for extracting line segments is the Hough transform [10]. We regard this as a suitable method because it is used a lot for this kind of problems. This is probably because it is unique in its low complexity (compared to other methods like *RANSAC*, who often use an iterative approach). For a detailed explanation of the Hough transform, see section 2.1.

The input of the Hough transform that is build-in in *MATLAB[15]* is a binary image. This is in our case the output of the skyline detector (Chapter 3).
If a pixel is classified as a skyline pixel (a pixel that lies on the skyline according the skyline detector), the Hough transform increases a vote value for every valid line (r, θ) pair that crosses this particular pixel. Lines (r, θ) pairs that receive a large amount of votes contain a large amount of skyline pixels.
Because the algorithm detects straight lines containing only skyline pixels it returns only the straight parts of the skyline. As these straight skyline parts are likely to come from the building contour, we found exactly what we where looking for. Results of the Hough transform on the output of the skyline detector are displayed and evaluated in the Result section (4.3).

4.2.5 Project the skyline to the 3D model

The Hough transform of the previous section returned a set of 2D line segments which likely present parts of the building contour. As we want to estimate the building wall heights we need to correspond these line segments to the walls. This is done by projecting the line segments to a specific part of the 3D model. We present a stepwise solution: first the camera is calibrated, next we project the line segments to the the building and finally we determine the specific building part that is associate with a line segment.
To project the line segments to the 3D model we need to know the camera's position and rotation when it took the photo (extrinsic parameters). Furthermore we need to know in what way this camera transformed the image (e.g. lens distortion) (intrinsic parameters). This process is referred to as camera calibration and is explained in (2.3)

From image point (2D) to possible points in scene (3D)

What can we do if we computed the camera calibration parameters? The line segment that was returned by the Hough transform consists of two endpoints v and w. These endpoints are in 2D but are recorded in a 3D scene and therefore present a 3D point in space. We don't know which point this is as for example we don't know the distance from the 3D point to the camera that took the picture. However, because we calibrated the camera (2.3) we can reduce these possible points in 3D space to a line. Next we explain how we calculate this for one 2D image point (for example a line segment endpoint).

From the input images and the calibration process (2.3) we have:

- $\vec{x} = (x, y)$, the image point (in the camera coordinates (XYZ))
- $\vec{x}_h = (x, y, 1)'$, the homogeneous coordinate of the image point
- $\vec{c_{ijk}}$ and R, the camera's extrinsic parameters (the center and rotation of the camera in world coordinates (ijk))
- K, the camera's intrinsic parameters
- P, the projection from camera coordinates (XYZ) to world coordinates (ijk). It contains the extrinsic parameters (the camera's center $\vec{c_{ijk}}$ and rotation R).

The image point in the image coordinate frame corresponds to a line of possible points in the world coordinate frame. The two coordinates that span this line are calculated as follows:

- $\vec{c_{ijk}}$, the location of the center of the camera in world coordinates (ijk)
- $\vec{x}_{ijk} = PK'\vec{x}_h$, the image point expressed in world coordinates (ijk)

This is illustrated in Figure 24.

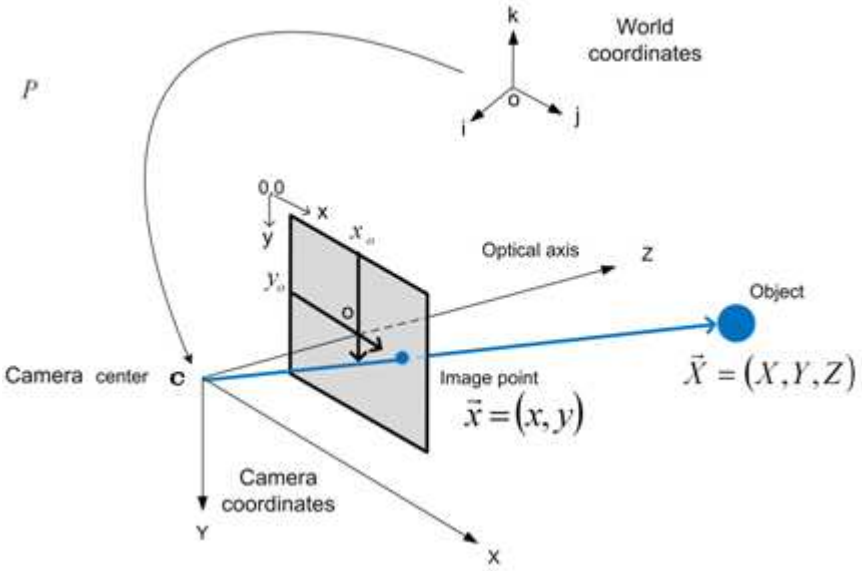

Figure 24: The blue line spanned by the camera center $\vec{c}$ and the image point $\vec{x}$ transfered to world coordinates represent the possible 3D points in space. $\vec{X}$ is a random possible point on the line.

Using the two required coordinates we set up an equation of the line of possible points in 3D.

$$l = \vec{c_{ijk}} + (\vec{x}_{ijk} - \vec{c_{ijk}})t, t \in \mathbb{R}$$

for example if $t = 100$ then the point in the real world lies at 100 times the distance from the camera center to its retina. In Figure 24 t is about 4 (for point M).
Above calculations are done for all the line segment's endpoints. This results in a line of possible points in 3D space for each endpoint. Next we explain how we use this to find the line-wall correspondences.

4.2.6 Associating line segments with building walls

Building wall appearance assumption:

We assume that every straight line segment of the skyline represent (a part of) the upper side of a specific wall of the building.

Unfortunately we don't know which line is associated with which building wall. In this section we determine this association.
First we describe the walls with planes. Next, we project the line segments to all planes of the 3D building by taking their endpoints and using the technique of the previous section (4.2.5) Finally we determine the most likely wall based on the largest line-wall overlap.

Walls to planes

The 3D building model consists of different walls. A wall is described by two ground points, a height, and a direction. The height is based on an average building height, the direction is always the y-direction (see *Gravity aligned walls assumption*). We transform the walls into (infinite) planes. This is done for two reasons: first this transformation is required to calculate the intersection properly. Second, because the 3D model is an estimate, the walls maybe just to small which could result in a missing intersection.

Intersect with all walls

Now we have the building walls transformed to planes, we take the endpoints of the lines and project them to all the planes of the building for further selection. Each 2D endpoint has a line of possible 3D points which we calculated in the previous section. This was the line spanned by the camera center and the image point in world coordinates. This line is intersected with all planes of the building walls.
Every 2D endpoint is now associated with multiple intersections resulting in 2 x l x w points in 3D (grouped by the line segments), 2 means #endpoints of

the line, l is the number of lines and w is the number of walls.

We now calculated every possible intersection with every plane. How do we determine which plane represents a line segments most likely wall? Recall that the wall is a subset of the plane. We only calculated the projection to the planes spanned by the walls hence we don't know which line lies on the wall and which falls outside the wall. To solve this problem let's zoom in to the situation:
If we project a skyline part l in 2D containing two points to the plane spanned by a wall W we get two intersections points that present the projected line l_{projW} in 3D. If we assume l to come from the contour of wall W, then l_{projW} should have a large overlap with this wall W. We call this the line-wall overlap value, lwo. Besides the large overlap with W we expect a small or zero lwo for the other walls, see Figure 10a and 10b.

Largest line-wall overlap assumption:

A line segment is associated with the wall with the largest projection overlap.

Having defined the assumptions, the situation and the idea behind the line-wall association, we can now explain the line-wall matching algorithm.

A line segment is projected to all walls and the amount of line-wall overlap, lwo is calculated. The wall with the largest overlap with the specific line segment is classified as the most likely wall for that line segment. Next the line segments are projected to their most likely wall and the algorithm outputs this set of lines in $\mathbb{R}3$.
This line-wall overlap is calculated in different steps. Before we explain the steps we discuss the different types of overlap. Next we explain how the algorithm determines the *overlap type*. Finally we we calculate the overlap amount and normalize this value.

l_{projW} can overlap W in four different ways, this is illustrated in Figure 10. The wall W is spanned by $abcd$, and l_{projW} is spanned by vw.

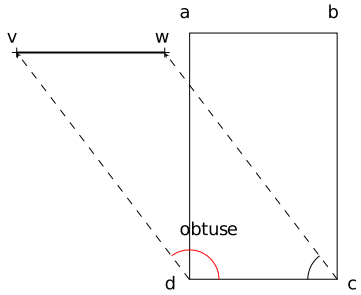

Figure 10a

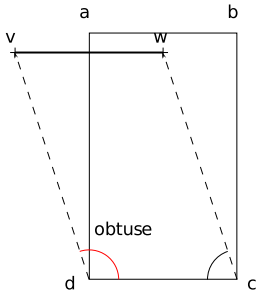

Figure 10b

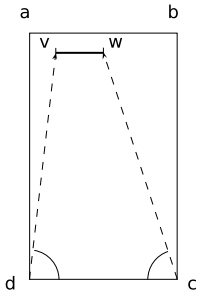

Figure 10c

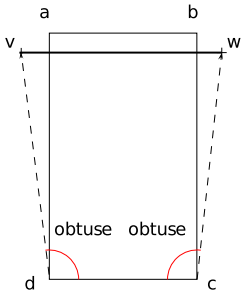

Figure 10d

Table 4: Types of overlap with corresponding number of points in polygon

Type of line-wall overlap	Points in polygon	Line-wall overlap	Figure
No overlap	0	0	10a
Partial overlap	1	[0..1]	10b
Full overlap (included)	2	1	10c
Full overlap (overextended)	0	1	10d

The type of overlap is defined by exposing the endpoints of the line segments to an *in polygon* test, where the polygon represents a wall of the building (e.g. *abcd* in Figure 10).
Table 4 represents the types of overlap with the corresponding number of points that pass the *in polygon* test and their possible line-wall overlap value.

No overlap

If the point in polygon test returns 0, the line-wall overlap calculation is skipped and 0 is returned. The remaining overlap types, partial and full, are treated individually:

Partial overlap

Let's first consider the partial overlap type (Figure 10b), the *in polygon* test returned 1, that means that one of the line segments endpoint lies inside and one lies outside the wall.
To determine the amount of line-wall overlap, the part of the line segment that overlaps the wall is calculated. The length of this part is measured and stored as its *lwo* value.
The trimmed line has two coordinates: 1) the point that passed the *in polygon* test and 2) the intersection of the line segment with one of the vertical wall sides (*da* or *cb* from Figure 10b).
How do we determine which vertical wall side is crossed? We use the fact that one of the line segments endpoints lies outside the polygon next to the vertical wall side. This point is easily determined by an angle comparison: first, two groups of two vectors are defined: dv, dc and cw, cd (see Figure 10b). We measure the angles between the vectors and call them $\angle d$, and $\angle c$. Because one of the line segment endpoints lies outside the wall, either $\angle d$ or $\angle c$ is obtuse. In this example $\angle d$ is obtuse therefor the left wall side is crossed. Note that this only holds because the walls are orthogonal to the basis which we assumed in the *Gravity aligned walls assumption*.

Now we know that:

- If $\angle d$ is obtuse, the left vertical wall side da, is crossed.
- If $\angle c$ is obtuse, the right vertical wall side cb, is crossed.

The angles are acute or obtuse if the dot product of the vectors involved are respectively positive or negative. Note the advantage of this method: it is simple and has low computational costs.

Line-wall overlap calculation

The amount of line-wall overlap is calculated by cutting of the point where l intersects the determined vertical wall side (da or cb) and measuring its remaining length.

Full or no overlap

Now let's consider the overlap types where the *in polygon* test returned 0. As you can see in Figure 10a and 10d this resulted in either full or no overlap. Again, we analyze the vector angles to determine the remaining overlap type. If only one of the angles is obtuse and no points lie in the polygon (Figure 10a), the entire line segment lies outside the wall and $lwo = 0$.
Otherwise, if both angles $\angle d$ and $\angle c$ are obtuse or acute (Figure 10d), both endpoints lie on a different side of the wall, and they cross the wall somewhere in between. Full overlap is concluded here.

The amount of overlap is now calculated by measuring the length of the line segment which is cut down by his intersections with da and cb. In this case this is equal to the length of the line dc, however, it is easy to see that this only holds of vw is parallel to dc.

Line-wall overlap normalization

Finally the line-wall overlap is normalized by the line segments length:

$$lwo' = \frac{lwo}{|l|} \tag{1}$$

Where lwo' is the normalized line-wall overlap, lwo is the length of the trimmed line segment, and $|l|$ is the total length of the line.
The intuition behind this is that line segments that are likely to present a wall not only have a large overlap but also have a small part that has no overlap, the missing overlap should have a negative effect. By calculating the relative overlap $\frac{lwo}{|l|}$, both amounts of overlap and missing overlap are taken into account.

Normalizing the line-wall overlap on the line length implies that the length of the line does not influence anymore. This can be seen as a disadvantage because small (noisy) line segments with 100% overlap will outperform large robust lines with little overlap. This problem is solved by discarding all line segments smaller than a *MinimumLength* value. This is done during pre-processing in the Hough transform module.
Next the normalized line-wall overlap is used to search for the correct line-wall association. This is achieved by associating a line segment with the wall that has the largest line-wall overlap.

To summarize, the overlap type is determined by calculating the numbers of in polygon points and evaluating two dot products. Next the line segment is cut off depending on the overlap type and the line is normalized. The wall where the line segment scores the maximum lwo' value is associated with this line segment.

4.2.7 Improving the 3D model by wall height estimation

In the previous section we associated the line segments with their most likely wall. In this section this information is used to estimate the heights of the walls of the 3D model.
Every different view of the building produced a collection of lines that are associated with a certain wall. For example if we consider 4 views and for wall W the views have respectively 2,4,4,1 lines that are associated with W, we have a total of 11 line segments that correspond with wall W.

Next we re-project the line segment from the different views on their associated walls. The re-projection is done as explained in (4.2.6) by intersecting both endpoints of the line segment to the plane that is spanned by the associated wall.

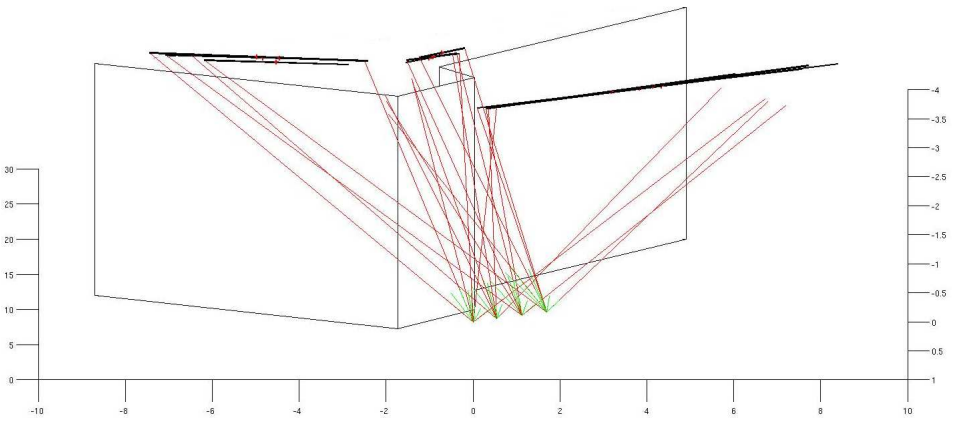

Figure 25: The Houghlines collected from different views projected on their corresponding wall

Next the average height of the projected line segments is calculated for each wall. This is done by collecting and averaging the y-value of the middle points of the line segments and this is done for each wall separately. Next, these averages are used to set the new heights of the planes of the 3D model.
The new individual heights are applied to the 3D model by adjusting the height of the existing upper corner points of the walls. We copy the bottom left and right corner points and add the estimated height from the previous section to its y-value. The y-value is the direction of the gravity which is obtained through the *Gravity aligned walls assumption*.

4.3 Results

Figure 26: Three best ranked lines of the Hough transform on the skyline detector match the three most prominent displayed building walls

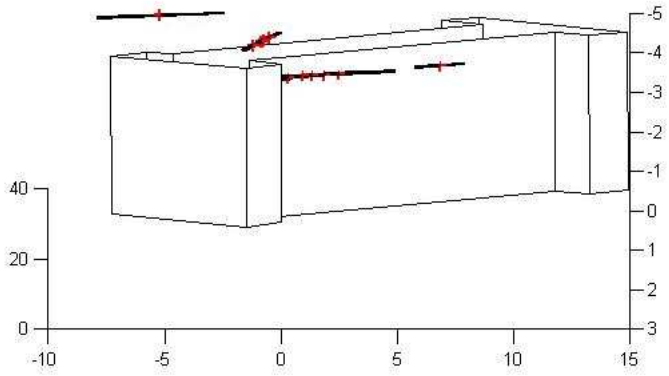

Figure 27: The Houghlines collected from different views re-projected on the 3D model according to the line-wall correspondence.

Figure 28: The walls of the building differ in height: the back part of the building is higher than the middle part.

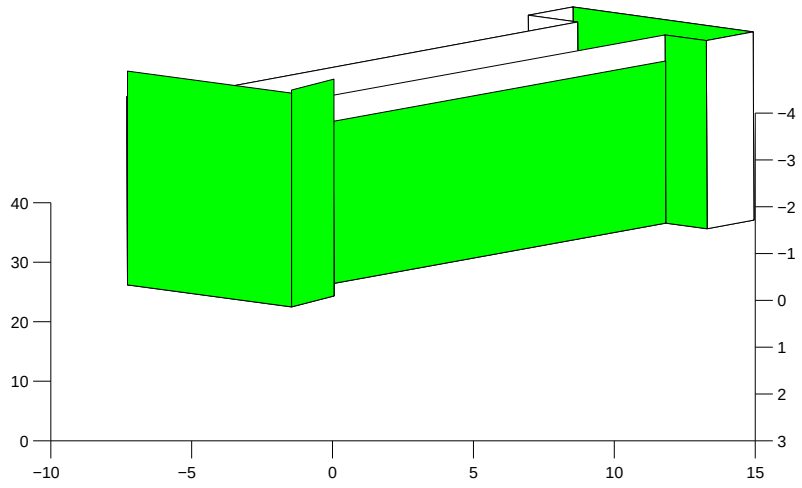

Figure 29: Improved 3D model

Figure 26 shows the top 3 longest Houghlines of a single view. The endpoints are denoted with a black and blue cross. All three line segments lie on the building contour. The left line segment covers only a part of the building wall. The middle line segment covers the full wall contour. The left and middle line segment are connected. The right line segment covers the wall until it occludes the tree.

Figure 27 displays the line segments (collected from Figure 26 and 6 other views) projected onto their associated walls. For a clear illustration we selected just three (prominent) walls. For each view a different red cross is drawn at each wall. It illustrates the average height of the corresponding lines corresponding to that wall.
Figure 29 displays the updated 3D model. The corner points of the walls are adjusted according the calculated wall heights. The green plane displays the modified wall. The left and middle wall are extended whereas the right wall is shortened.

4.4 Discussion

As can be seen in Figure 26, the top three Houghlines correspond to the three most prominent building walls. What also can be seen is that the left line segment doesn't cover the entire building wall. This is caused by the use of a small line thickness parameter in the Hough transform: if some ascending skyline points fall just outside a Houghlines, a gap is created and the line segment is cut down at that point. As we use a large minimum length the residual line segment is to small and is discarded. However, this is not a big problem because the lines are long enough to produce a good wall height estimate. Furthermore, in this example, there are 5 other lines (originated from the different views) that support the height estimate for this particular wall.

The different red crosses in Figure 27 which illustrate the average height per view agree in height. This means the camera calibration, edge detection and Houghline extraction where accurately processed and our method is consistent.

Although the building appears to have similar building heights, the middle part of the building is smaller then the front and back part of the building, see Figure 26 and 28. This agrees with the updated 3D model in Figure 29.

4.5 Conclusion

Let's answer our research question.

Research question

Is it possible to use a set of (noisy) skyline points together with an aerial 2D model and a 3D point cloud obtained by *FIT3D toolbox[11]* to generate a 3D model of the building?
Yes this is possible, we showed that a Houghline transform is a useful method to discard skyline outliers and find prominent structure in the contour of a building with a flat roof. We introduced a method to extract a 2D model of a building from *Openstreetmap[1]* and aligned it to the scene using the *FIT3D toolbox[11]* Next, we extracted the skyline and transformed this to Houghlines which we paired up with their associated walls. This was used to produce new wall heights which were propagated to the 3D model. Existing and novel AI computer vision techniques were powerfully combined resulting in an reasonable 3D model based on only a few 2D images.

4.6 Future research

4.6.1 Gravity aligned walls assumption

In this project we assumed the walls to align with the gravity. This means the camera must be up right: his parameter *roll* must be exactly zero when capturing the images. In practical use this is not true. We demonstrate this by plotting the 3D point cloud with the 3D model in Figure 30. Although this assumption lets us focus on the important issues, it would be nice to incorporate gravity estimation in future research. Costin Ionita wrote a part of his master thesis about gravity estimation in [13].

4.6.2 Double wall height influence

Sometimes two line segments extracted from one view appear to correspond to the same single wall. This means that they have a double influence on the average wall height, which is unjustified. A simple solution would be to add a normalization pre-process step, so each view has only one wall height vote per wall. A more decent solution would be to merge the two (or more) line segments to a single line segment. Lines that are close and parallel could be merged and averaged. Lines that lie in each others direction could be merged by increasing the Hough transforms *FillGap* parameter. E.g. for the right wall of the building in Figure 27 the *FillGap* parameter needs to be at least as big as the occluding tree.

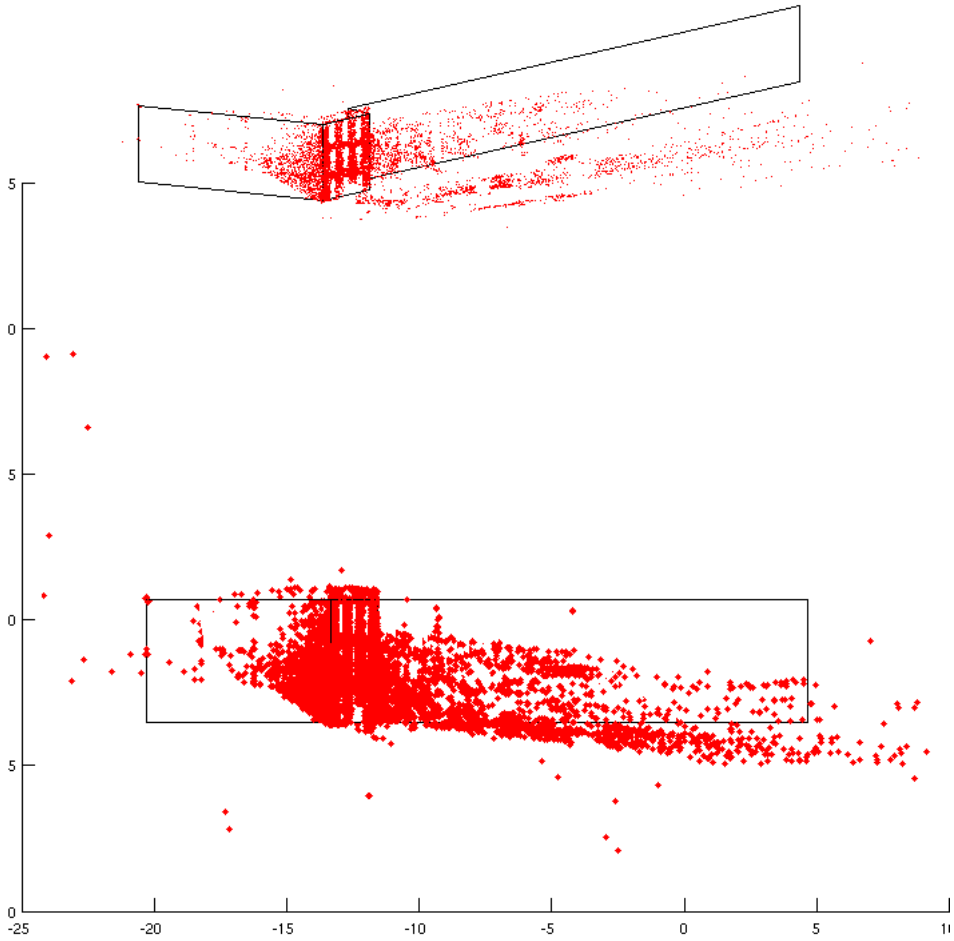

Figure 30: The poincloud plotted in two views of the 3D model. This illustrates that the walls are in practise not aligned to the y-axis.

4.6.3 Complexity

In this thesis little is discussed about the computational costs. Because the computations are done efficiently (e.g. using matrix multiplications in MATLAB[15]) and off line, the calculation are done in reasonable time. However, if we want to make the application real time, the next speedup would be useful.
To determine the best line-wall association the line segments are now projected to every wall and for every wall the amount of line-wall overlap is calculated. This is computational very expensive and looks a bit like an overkill.

It would be a significant speedup to reduce the set of walls to only the walls that contain the middle point of the line segments. To be more concrete the middle point of the line needs to be exposed to an *in polygon* test for every wall. Next the line-wall association algorithm only treats the walls that pass this test.
The downside of this method is that it makes the system less accurate because it will resulting in more false negatives. A line segment that overlaps the wall with only 1/3 could be an important candidate for the height estimation, however the speedup method would discard it. What can be concluded is that there is a trade of in the accurateness of the height estimation and the computational costs.

4.6.4 Alternative roofs

We assumed a flat roof, this doesn't mean that our methods are unusable if we discard this assumption. E.g. without adaptations the existing methods could be used to determine the (maximum) building height. If we discard the flat roof assumption the building is allowed to have any shape. In this situation it should also be possible to extract a full 3D model. We will now consider other roof types and discuss what adaptations the system should require to handle these. In Figure 31, 6 different roof shapes are displayed.

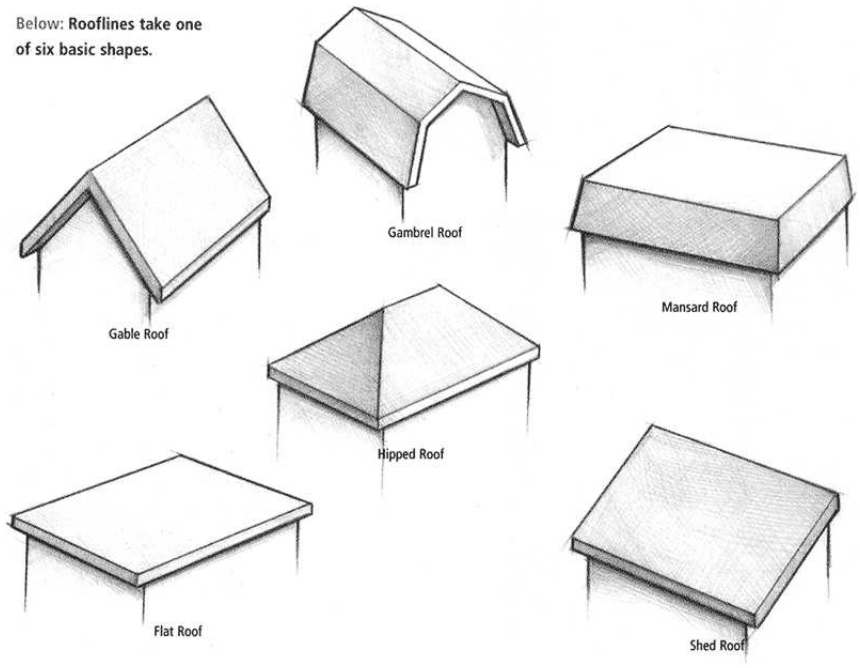

Figure 31: Different types of roofs

Consider the *Gable Roof*, it is a roof consisting of two skew planes This makes

the extraction of the 3D model more complex, but not impossible.
Because we assume that the roof images are taken from the ground, the skyline detector will always detect the top of the building. In case of a flat roof this is also the top of the building walls. In case of an alternative roof, this will be just the top of the roof. The building walls however, could lie a lot lower. Therefore something else needs to be developed to find the wall heights. It would be useful to develop a method that extracts the roof and the walls separately. An idea about this is now proposed:

The 3D model of a building with a non-flat roof type will consist of a cuboid with a custom roof on top of it. First we would generate the custom roof using the shape of the skyline. Next we would determine the height of the cuboid. Finally, we connect both parts and have a full 3D model.

The skyline detector discussed in Chapter 3 could be used to extract the contour of the roof. Next we would extract a basic 3D model (discussed in Chapter 4) which is used to transform the roof contour to a frontal view. Houghline extraction could be used to extract straight lines which correspond to the (frontal viewed) base of the primitive planes of the roof. For example the Gable roof will return two skew lines connected in the middle. Next we extend these lines to primitives, e.g. planes. We would extract a point cloud of the side wall using the *FIT3D toolbox[11]* to determine the alignment of the planes.

Next, a shape is cut out of the plane according to the side view of the building contour. For example the Hipped roof (Figure 31) would imply that we have to cut out a trapezium shape whereas most other roofs require to cut out a rectangular shape. Note the power of combining the frontal and side view to determine the roof planes.

Let us apply this method to the Gable roof: First we rotate the scene to a frontal view, the skyline detector will extract two skew lines. Next we extend them to rectangular surfaces which are connected in the middle and are aligned with the point cloud.

We have determined the roof, next the height of the cuboid needs to be determined. This could be done by detecting the roof-wall separation on the side view of the building.
The roof color often differs from the wall color which produce strong edges. Also some building contain a gutter connected to the roof. The gutter will also produce a long strong edge.

These strong edges can be detected using the Hough transform. Next this can be used to set the height of the cuboid and the lower bound location of the roof. Finally we connect the roof and the cuboid to obtain a full 3D model shaped like Figure 32.

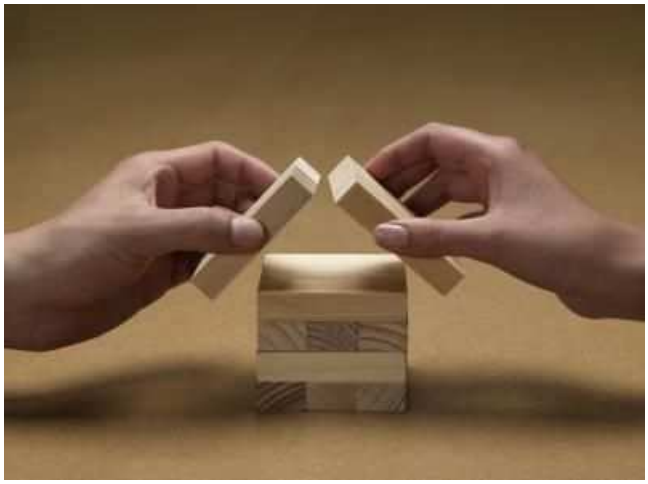

Figure 32: The parts of the 3D model for the Gable roof

To discard roof-wall outliers we would only search for the between the ground plane and the top of the roof. The location of the top of the roof could be obtained by the highest skyline point that is present in both frontal and side view. The ground plane could be estimated using Ionita's [13] method.

5 Window detection

5.1 Introduction

In this chapter we deal with an important aspect of semantic urban scene interpretation: window detection. From the introduction we learned that window detection can play an important role in a variety of domains: semi automatic 3D reconstruction/modelling of city models, documenting historical buildings, analysis of old building deformation, augmented reality, building recognition, etc.

This chapter deals with our developed methods of window detection and it is organized as follows:
We start with our research question. After this we discuss related work and put our work in context. Next we describe our first window detection approach that is invariant to viewing direction. After this we present a facade rectification method. Next the rectification result is used for our second method that assumes orthogonal and aligned windows. Finally we show and discuss our results.

Research question

Is it possible to use edge features to supply an accurate detection of windows in urban scenes?

5.2 Window detection: state of art

A large amount of research is done on semantical interpretation of urban scenes. First we briefly discuss the research that is done on window detection using approaches that differ from our approach. After this, we discuss state of art window detection that lays at the basis of our approach in detail.

5.2.1 Alternative approaches on window detection

Muller et al. [17] detect regularity and symmetry in buildings. The symmetry is detected in the vertical direction (floors) and horizontal direction (window rows). They use shape grammars to divide the building wall in tiles, windows, doors etc. The results are used to derive a 3D model of high 3D visual quality. Although they achieved some interesting results, their method has some disadvantages. As their method is fully based on detecting symmetry, they have to assume repeating and aligned windows. This constrains the variety of scenes the system can handle. Furthermore they match template window objects which they predefine. This constraints the variety of window types that could be matched. At last they use expensive algorithms that make it impossible for the system to run in real time.

(a) original texture (b) reconstruction

Figure 33: Results of Muller et al.

Using a thermal camera, Sirmacek [23] detects heat leakage on building walls as an indicator for doors or windows. Windows are detected with L-shaped features as a set of *steerable filters*. The windows are grouped using *perceptual organization rules*: they search and group intersecting L-shapes to close a window shape. A shape is determined closed if it can separate an inside region from the outside. If the shape is closed it is classifies as a window.

Ali et al. [3] describe the windows with *Haar* like features, the features are combined using a cascading classifier. The cascading classifier (which acts like a decision tree) is learned using the *Ada boost* algorithm. They also applied window detection to determine the region of the facade. Although this method is used a lot in the computer vision domain [19], it is not the most promising approach to window detection because it is supervised. This means that it requires a large dataset in which every window must be accurately annotated. Furthermore, because the cascader uses a fixed swiping window it is sensitive to scale: all windows in the scene must be of about the same size and a size range must be given to the system. Furthermore the system is always overfitted to the learning data , making it hard to generalize (detect windows that are not included in the dataset). A more general descriptor of the window that is size invariant is desirable.

To investigate this, we developed two methods of which one method does not require repeating windows nor aligned windows. One of the main targets of our research is to have small requirements on the input data. First our system doesn't need a large annotated dataset. Furthermore we extract the windows from the image space only which makes us independent of additional expensive data like heat or laser range images.
This doesn't mean the previous work on window detection using laser or heat images isn't of good use. Instead we learned a lot from the previous research as they have to match the laser or heat data to the real image space. This

matching process involves a description (semantical annotation) of the facade. Let us explain a method of that kind and discuss approaches that are more similar to our approach.

5.2.2 Similar approaches

Pu and Vosselman [20] combine laser range images with ground images to reconstruct facade details. They solve inconsistency between laser and image data and improve the alignment of a 3D model with a matching algorithm. In one of the matching strategies they compare the edges of a 3D model to extracted Hough lines of both ground and laser range images. They match the lines by comparing the angle, location and length differences. These criteria are also used in our approach.

They also detect windows and use them to provide a significant better alignment of the 3D model. As windows have a high reflection, they form hole like shapes in the laser range images. These holes are directly used to extract the windows. Unfortunately due to bad laser range data, the results where far from accurate.

The work of Pu and Vosselman [20] provides an useful practical application of window detection. It amplifies the need for a robust window detection technique that is independent of laser range data.

Recky et al. [21] developed a window detector that is build on the primary work of Lee and Nevatia [14] (which is discussed next). In order to make clear orthogonal projections they rectify the facade. To determine the alignment of the windows, the edges are projected into their orthogonal direction. For example the horizontal edges are projected in the vertical direction to establish the vertical division of the windows.
A function is developed that counts the amount of projected edges on each location, this is called the *Projection profile*, see Figure 34.
A threshold is applied on the projection profile to indicate the window boundaries that are used for the window alignment.

In the next step they use color to disambiguate the window areas from non-window areas. To be more precise, they convert the image to CIE-Lab color space and use k-means to classify the windows. Although this method is robust, both color transformation and k-means clustering are very computational expensive. Furthermore the classification based on color is sensitive to change in illumination conditions.

Similar to the work of Recky et al. [21], Lee et al. [14] perform orthogonal edge projection to find the window alignment. As different shapes of windows can exist in the same column/row, they only use the window alignment as a hypothesis. Then, using this hypothesis, they perform a refinement for each

Figure 34: Projection profiles of Lee and Nevatia's work

window independently. Although this comes with accurate results, the iterative refinement is a computational expensive procedure. As we want to run our system in real time this method is not suitable for our application.

5.2.3 Our work in context

The state of art window alignment procedure in [21] and [14] is very robust. Therefore we have decided to use this method as a basis and improved the alignment algorithm. Furthermore we have build a different window classification method.
Our improvement on the alignment procedure is as follows. In the previous work [21] and [14] a single projection profile for each direction is used. We improved this process by fusing two (more advanced) projection profiles for each direction. E.g. for the determination of the horizontal division of the windows we fuse both horizontal and vertical projection profiles.

Furthermore we have build two alternative window classification procedures which are based on the shape interpretation of these projection profiles. As the classification is based on the projection profiles (edge information) we don't require expensive color transformations and we only apply (rectification) transformations on line segment endpoints. This makes our algorithm invariant to change in illumination and we expect it to perform in real-time.

5.3 Method I: Connected corner approach

5.3.1 Situation and assumptions

A window often consists of a complex structure that optionally contain several sub windows enclosed in rectangular window frames. As the color of the window frames differ from the glass, the amount of horizontal and vertical edges is large at these locations. Some horizontal and vertical edges come from the same window frame, therefore they often share a corner of the window. We developed an approach where we pair up these horizontal and vertical lines to determine *connected corners*. The connected corners give a good indication of the position of the windows. In this approach the viewing direction is not required to be frontal. The windows could be arbitrarily located and they don't need to be aligned to each other neither to the X and Y axis of the image. As such the windows are detected individually.
First we detect the edges and transform them to straight lines.

5.3.2 Edge detection and Houghline extraction

Edge detection is done using the Canny edge detector motivated earlier in section (3.3.3). From the edge images two groups of Hough lines are extracted. The groups fall in the two window directions: horizontal and vertical. This is done by controlling the allowed angles, θ bin ranges, in the Hough transform.
The horizontal group has a range of $\theta = [-30..0..30)$ degrees, where $\theta = 0$ presents a horizontal line. The vertical group has a range of $\theta = [80..90..100)$ degrees.
We use ranges because 1) the user hardly ever holds the camera exactly upright and 2) we work with unrectified facades: the window sides do not apear orthogonal due te perspective distortion. To be more concrete, if the user takes a photo (Figure 35) with a certain Yaw not equal to zero, the horizontal lines become skew. The range of the vertical group is smaller then the horizontal group as the user often takes photos which vary in yaw but contain a low pitch.

The results of the edge detection and the Hough transform of two images can be seen in Figure 37 and 38.

5.3.3 Connected corners extraction

How do we detect the connected corners? Often a connected corner contains a small gap or an extension which we tolerate, these cases are illustrated in Figure 36 in the top row: a horizontal gap, a vertical and horizontal gap and a vertical elongation. The modified connected corners are given in the bottom row.
When the horizontal and vertical lines intersect, the gap distance is $D = 0$. When the lines do not intersect, the distance D between the intersection point P_i and the endpoint P_e of the line is measured $D = ||P_i - P_e||$, this is illustrated as dotted lines in Figure 36. Next, D is compared to a *maximum intersection distance* threshold $midT$. And if $D <= midT$, the intersection is close enough

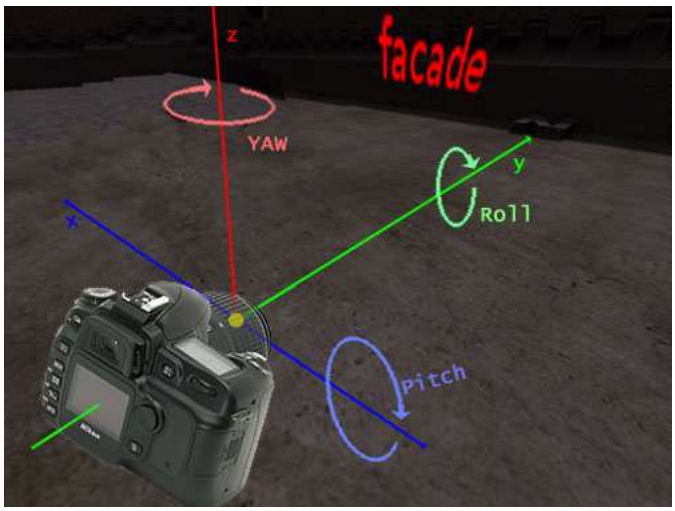

Figure 35: Pitch roll and yaw of the camera

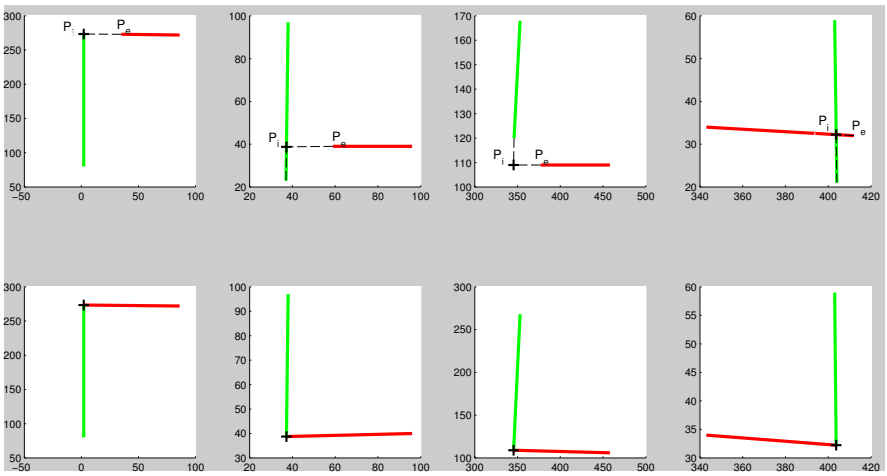

Figure 36: First row: different type of connected corner candidates. Second row: the result the clean connected corner

to form a connected corner.

After two Hough lines are classified as a connected corner, they are extended or trimmed, depending on the situation. The results are shown in the second row in Figure 36. In Figure 36(I) the horizontal line is extended. Figure 36(II) shows that the vertical line is trimmed. In Figure 36(III) both lines are extended. At last, Figure 36(IV) shows how both lines are trimmed.

5.3.4 Window area extraction

To retrieve the actual windows, each connected corner is mirrored along its diagonal through the endpoints. The connected corner now contains four sides which form a quadrangle window area. All quadrangles are filled and displayed in Figure 40, this result is discussed in section 5.3.5.

5.3.5 Results

.

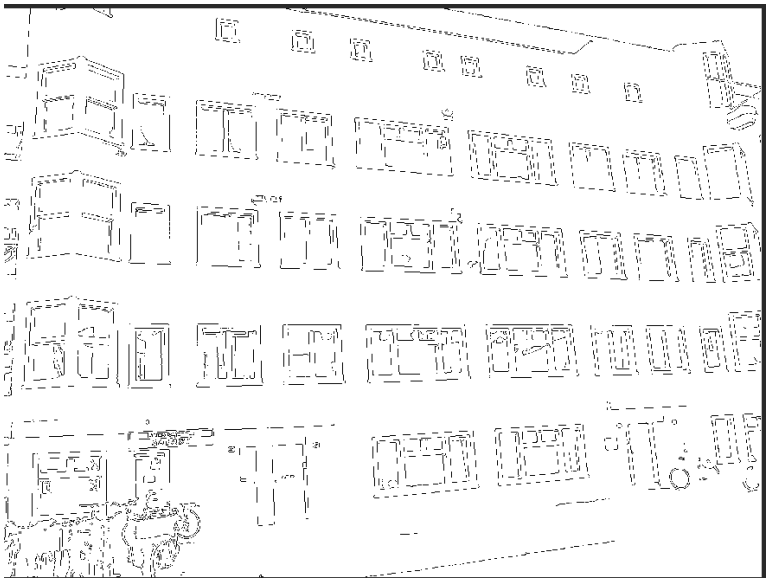

Figure 37: Edge detection

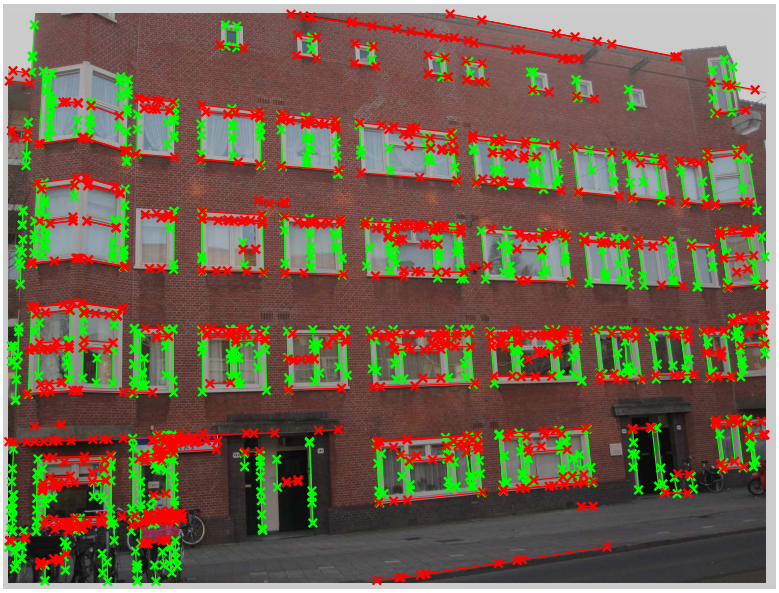

Figure 38: Result of θ constrained Hough transform

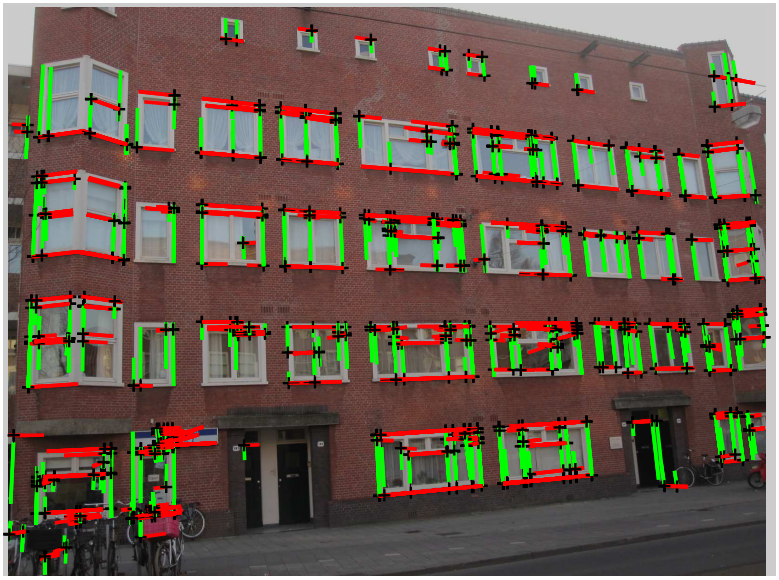

Figure 39: Found connected corners

Figure 40: Window regions

Figure 40 displays the result of the connected corner approach on an unrectified scene. It contains 110 windows of which are 109 detected, this is 99%. Furthermore there are some false positive areas, this is about 3 %. Sometimes a window is not detected, for example the window on the right top isn't detected, this is because its smaller then the minimum length of the Hough line.

5.3.6 Future research

We only developed L-shaped connected corners. In future research we could connect more parts of the window. E.g. to form U shaped connected corners or even complete rectangular shapes. The latter is difficult because the edges are often incomplete due to for example occlusion.

The next step in this study would be an analysis of the substructure of the windows. The big window that contains sub windows could be found by calculating the convex hull of the red areas in Figure 40. The sub windows could be found by grouping connected corners that correspond to the same sub window. This could be done by clustering the connected corners at their location. We could extend the parameters of the cluster space with the length and position of the connected corners' horizontal and vertical line parts. It would be useful to assume the number of sub windows as this can be used to determine the maximum inter-cluster distance. The inter cluster distance and the number of grouped connected corners could form a good source for the certainty of the sub window.

5.4 Facade rectification

5.4.1 Introduction

In order to apply our second method of window detection (5.6), we need the windows on the facade to be orthogonal and aligned. Therefore we rectify the facade, this can be achieved in a manual or in an automatic way.

The manual rectification method uses point to point correspondences. This requires annotation of the corner points of the facade that are mapped with the corners of a rectangle. This mapping is used to calculate a transformation matrix. The downside of this method is that it is not very accurate as it doesn't take the width and height ratio of the facade into account. It also does not take the camera lens distortion into account. Another downside is that it requires (manual) annotation of the corner points. As we want our rectification method to be accurate and fully automatic this method is not suitable.

The second method involves the extraction of a 3D plane of the facade. This method is complex but gives more accurate results. Extracting a 3D plane from

a series of images is a comprehensive process and a large amount of research is done in this area. As we want to focus on the annotational part of facade interpretation, we used existing software to apply the rectification.

I. Esteban's *FIT3D toolbox* [11] comes with an add-on which extracts a 3D model from a series of frames. This add-on involves a process that calculates the motion between a series of frames in order to extract a point cloud of matching features. This point cloud is used to extract a plane for every unique wall. More details of this process are explained in (2.4).

5.4.2 3D plane based rectification

The next challenge is to use the extracted 3D plane in order to rectify the facade. It would be straight forward to rectify the full image. However this is computational very expensive as each pixel needs to be projected. To keep the computational cost to a minimum we project only the necessary data. Since we are using Hough lines we project only the coordinates of the endpoints of the found Hough lines. This is allowed because the projective transformation we apply preserves the straightness of the lines [6]. Note that this means we apply the edge detection and Houghline extraction on the unrectified image.

This effective way of projecting saves computational costs. If h is the number of Hough lines, the number of projections is $2h$. When we rectify the full image the number of projection is w x h, where w,h are the width and height of the image. To give an indication, for the *Anne1* dataset this means we apply 600 projections in stead of 1572864: a factor of almost 3k faster.

The Houghline endpoints are projected to the 3D plane that was extracted in the same way as we explained in Chapter 3. We calculate lines from the camera center trough the Hough line endpoints and calculated the intersection with the 3D plane. The result is a 3D point cloud where each point is labeled to its corresponding Houghline.

The next step is to transform the facade (and therefore the Hough lines) to a frontal view. Instead of transforming the facade we rotate and translate the camera. This means the heading (z-axis) of the camera needs to be equivalent to the normal of the facade plane. We determine the rotation matrix R by calculating the angle between the camera's heading an the normal of the facade. This process involves calculating 1) an orthogonal rotation vector $\vec{n}$ and 2) an angle θ, see Figure 41.
Next, R is applied to the 3D point cloud resulting in a set of rectified 3D points that are grouped to their Hough lines.

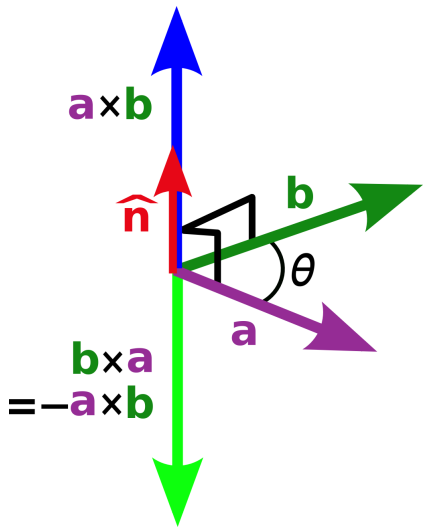

Figure 41: $\vec{a}$ is the camera's heading and $\vec{b}$ is the normal of the facade plane, $\vec{n}$ and θ are used to generate rotation matrix A [24]

5.4.3 Results

We show the result of two datasets, Anne1 and Dirk. Note that we also rectified the image pixels itself, this is only for the purpose of displaying the projected Hough lines in their context.

Figure 42: Dataset: Anne1, Original, (unrectified) image

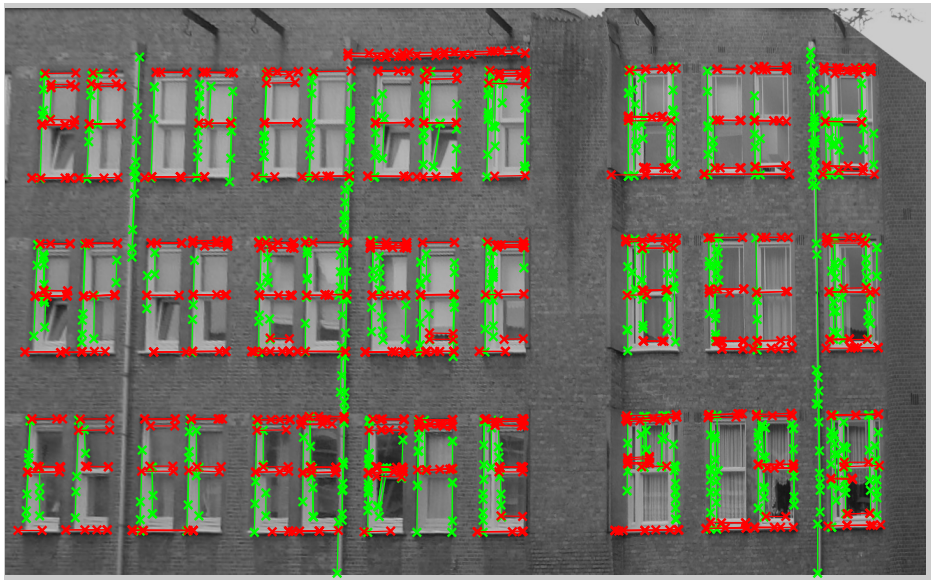

Figure 43: Dataset: Anne1, (Projected) Hough lines on rectified image

Figure 44: Dataset: Dirk, Original, (unrectified) image,

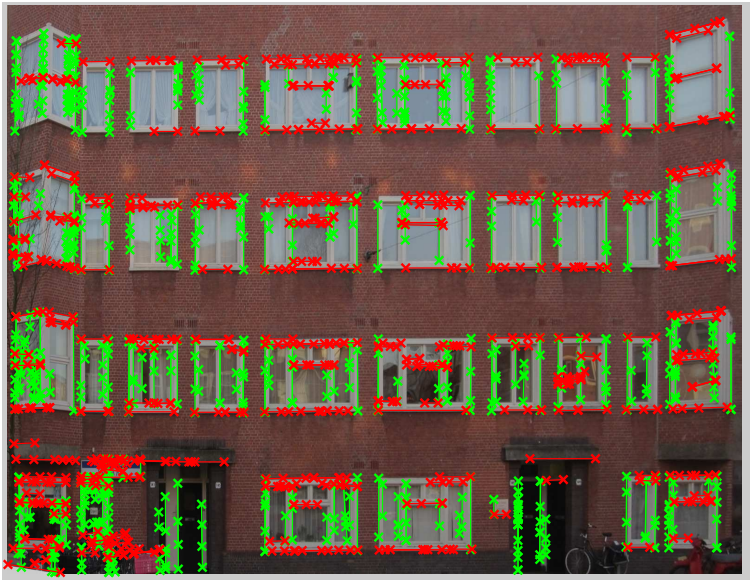

Figure 45: Dataset: Dirk, (Projected) Hough lines on rectified image

5.5 Datasets

To avoid over fitting we used multiple datasets that contain different urban scenes. Furthermore every dataset has a challenge/handicap. We used three datasets that are recorded in the suburban area 'de Baarsjes' of Amsterdam. All images have an original resolution of 3072x2304 pixels and are downscaled (using cubic interpolation) to 1280x1024 pixels.

5.5.1 Anne1

The challenge of this dataset is that it suffers from rectification errors. This makes the window alignment a challenging task as it assumes the windows to be aligned with the x-axis and y-axis of the image. The rectification error can be seen as the skew window alignment (and skew drainpipe) on the left of the image. Furthermore the yaw value of the camera (horizontal viewing angle) was (relative to the other datasets) high, making parts of the windows occluded. The height of the facade on the right side of the image is 698 pixels, at the left side this is 320 pixels: a resolution of more then 2 times smaller which makes it hard to detect the left windows. Trilinear interpolation is used to minimize this loss. To reduce the number of handicaps (and to focus on the rectification and occlusion error) we cut off the bottom of the image which included cars, unaligned doors and windows.

Figure 46: Dataset: Anne1, Rectified image

5.5.2 Anne2

This dataset contains images of the same scene as Anne1. It has zero rectification error: the windows are perfectly aligned, although the resolution on the left is as in the Anne1 dataset, two times smaller. The challenges of this dataset are the occlusion artefacts and the bottom area of the image (cars, unaligned doors and windows).

Figure 47: Dataset: Anne2

5.5.3 Dirk

The Dirk dataset represents an everyday scene as it contains light spots on the facade, bicycles and an occluding tree. It contains zero rectification error but the windows are partially aligned. The windows are very close to each other (making it hard to detect non-window areas between them), furthermore they differ in shape, size and in type. The yaw of the camera is relatively low, implicating little or no occlusion artefacts.

Figure 48: Dataset: Dirk

5.6 Method II: Histogram based approach

5.6.1 Introduction

In the previous section we saw that from a series of images, a 3D model of a building can be extracted. Furthermore we saw that using this 3D model the scene could be converted to a frontal view of a building, where a building wall appears orthogonal. This frontal view enables us to assume orthogonality and alignment of the windows. We exploit these properties to build a robust window detector as follows: first we rectify the image as described in the previous section. Then the alignment of the windows is determined. This is based on a histogram of the Hough lines. We use this alignment to divide the image in window and not window regions. Finally these regions are classified and combined which gives us the windows. We present a regular and alternative window alignment method followed by two different kind of window classifications.

Situation and assumptions

To be more precise in our assumptions, we assume the windows have orthogonal sides. Furthermore we assume that the windows are aligned. This means that a row of windows share the same height and y position. For a column of windows the width and x position has to be equal. Note that this doesn't mean that all windows have the share the same size.

5.6.2 Extracting the window alignment

Regular window alignment

We introduce the concept alignment line. We define this as a horizontal or vertical line that aligns multiple windows. In Figure 49 we show the alignment lines as two groups, horizontal (red) and vertical (green) alignment lines. The combination of both groups give a grid of blocks that we classify as window or non-window areas.

How do we determine these alignment lines? We make use of the fact that among a horizontal alignment line a large amount of horizontal Hough lines is present, see Figure 43. For the vertical alignment lines the number of vertical Hough lines is high, see the green lines.

We begin by extracting the pixel coordinates of Hough transformed line segments. We store them in two groups: horizontal and vertical. We discard the dimension that is least informative by projecting the coordinates to the axis that is orthogonal to its group. This means that for each horizontal Houghline the coordinates on the line are projected to the X axis and for each vertical Houghline the coordinates are projected to the Y axis. We have now transformed the data in two groups of 1 dimensional coordinates which represent the projected position of the Hough lines.

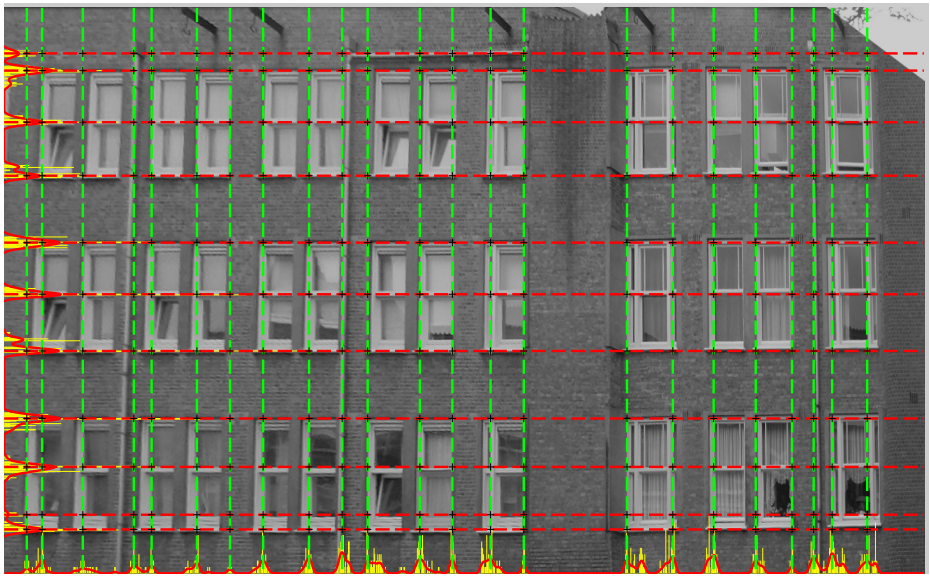

Figure 49: Dataset: Anne1, Regular window alignment (parallel projection): Based on a smoothed histogram (red line) that displays the amount of overlapping Hough lines, for the column division the horizontal Hough lines are counted (at each y position), for the row division the vertical Hough lines are counted (at each x position)

Next we calculate two histograms H(orizontal) and V(ertical), containing respectively w and h bins, where w x h is the dimension of the image. The histograms are presented as small yellow bars in Figure 49.
The peaks are located at the positions where an increased number of Hough lines start or end. These are the interesting positions as they are highly correlated to the alignment lines of the windows.
It is easy to see that the number of peaks is far more then the desired number of alignment lines. Therefore we smooth the values using a moving average filter.

The result, red lines in Figure 49 , is a smooth *projection profile* which contains the right number of peaks. The peaks are located at the average positions of the window edges.

Next step is to calculate the peak areas and after this, the peak positions. Before we find the peak positions we extract the peak *areas* by thresholding the function. To prevent an overfitted threshold value, a relative threshold is used. We set the threshold to $0.5 \cdot \max$ Peak. This value works for most datasets but is a parameter that can be changed.

Next we create a binary list of peaks P. P returns 1 for positions that are con-

tained in a peak, i.e. are above the threshold, and 0 otherwise. We detect the peak areas by searching for the positions where P = 1 (where the function passes the threshold line). If we loop through the values of P we detect a peak-start on position s if $P(s-1), P(s) = 0, 1$ and a peak-end on e if $P(e-1), P(e) = 1, 0$. I.e. if P = 0011000011100, then two peaks are present. The first peak covers positions $(3, 4)$, the second peak covers $(9, 10, 11)$.

Having segmented the peak areas, the next step is to extract the peak positions. Each peak area has only one peak and since we used an average smoothing filter, the shape of the peaks are often concave. Therefore we extract the peaks by locating the max of each peak area. These locations are used to draw the window alignment lines, they can be seen as dotted red lines and dotted green lines in Figure 49.

Alternative window alignment

As you can see in Figure 50 a few window alignment lines are not found and a few lines are found at wrong locations. The right side of the window frame of the first 4 columns of windows is not found. This means we have to find another way to detect the window alignment on these positions.

For the vertical alignment we only took vertical lines into account. In this method we examine the projection profile of the *horizontal* Hough lines projected on the X axis, X_h, Figure 51. On the positions of the desired vertical alignment lines there appears to be a big decrease or increase of X_h at the window frame. This is because on these positions a window (containing a large amount of horizontal lines) starts or end.

We detect these big decreases or increases by creating a new pseudo peak profile D that takes the absolute of the derivative of X_h, Figure 51.

$$D = abs(X_h')$$

Next we extract the locations of the peaks as the previous method.

Fusing the window alignment methods

We have presented two window alignment methods, next we fuse the methods to gain a robust window alignment. The aim is to have as few as possible false positives while detecting all alignment locations.

We fuse the methods by plotting the two sets of alignment lines in the same image. If the same window alignment position is found by both methods, the peaks are often located very close to each other, see Figure 52. If this is the case we want to fuse the results by grouping the peaks. Most of the times the peaks indicate the same window alignment but have some disparity. This is often the case when horizontal lines stop at the *inside* of a window frame while

the vertical edges are located at the *outer* side of the window frame (this is supported by the fact that the disparity is often the size of the window frame part). However, in other cases, close peaks indicate different windows that are just happen to be located closely. To apply a proper grouping of the peaks the challenge is to distinguish these two cases.

First we decrease the total number of found window alignment locations by increasing the individual thresholds (from 0.5*max peak to 0.7*max peak). Note that this has a positive side effect that the peaks that are found are more certain. After this we group the peaks as follows:
First we calculate the average of the maximum window frame part and the minimum window distance and call this the maximum peak group distance G. Next we compare all peaks and if the distance between two peaks is lower than G, we discard the peak with the least evidence (lowest peak). The result, a set of unique peaks, can be seen in Figure 58.

The advanced peak grouping is only required for the vertical alignment of the windows: the horizontal inter window distance is often big enough to not be mistaken by a window frame part.

Results

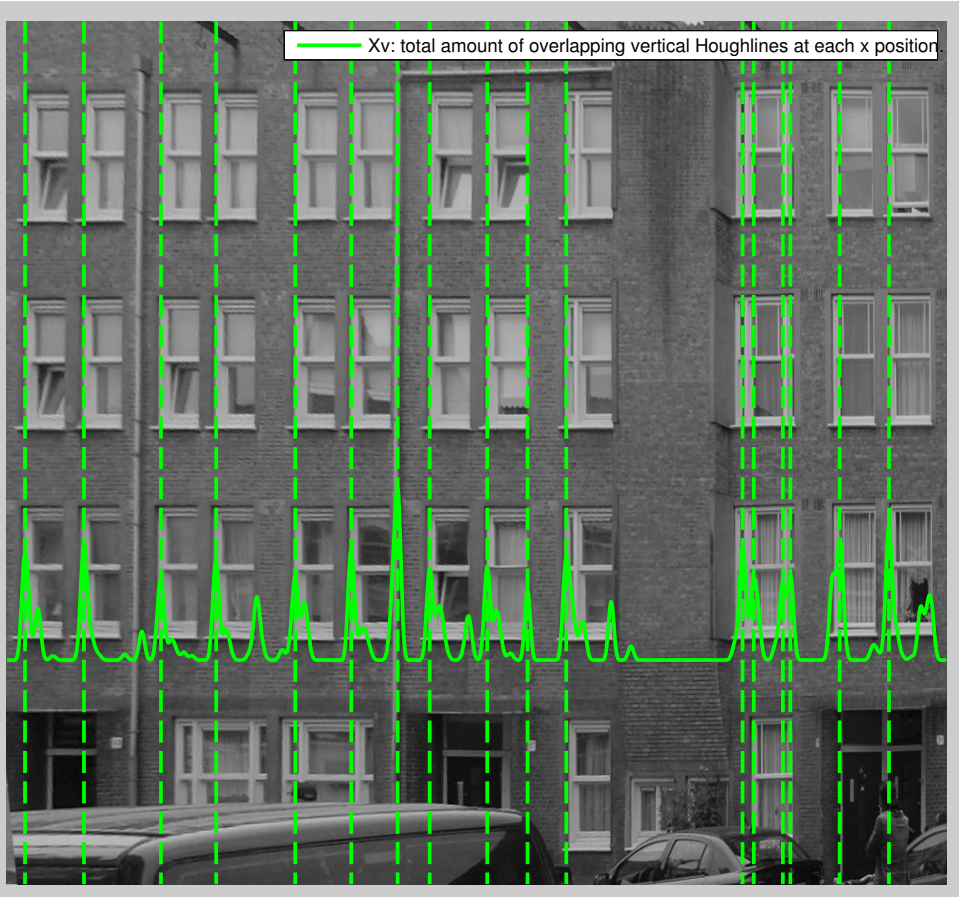

Figure 50: Dataset: Anne2, Regular window alignment: based on a histogram that displays amount of overlapping vertical Hough lines at each x position

Figure 51: Dataset: Anne2, Alternative window alignment (orthogonal projection): Based on the shape of the smoothed histogram function. For the column division, the number of *horizontal* Hough lines is counted (Note that this is the orthogonal opposite of the regular window alignment method). Peaks (that represent a big decrease or increase of the histogram function) are used for the alignment.

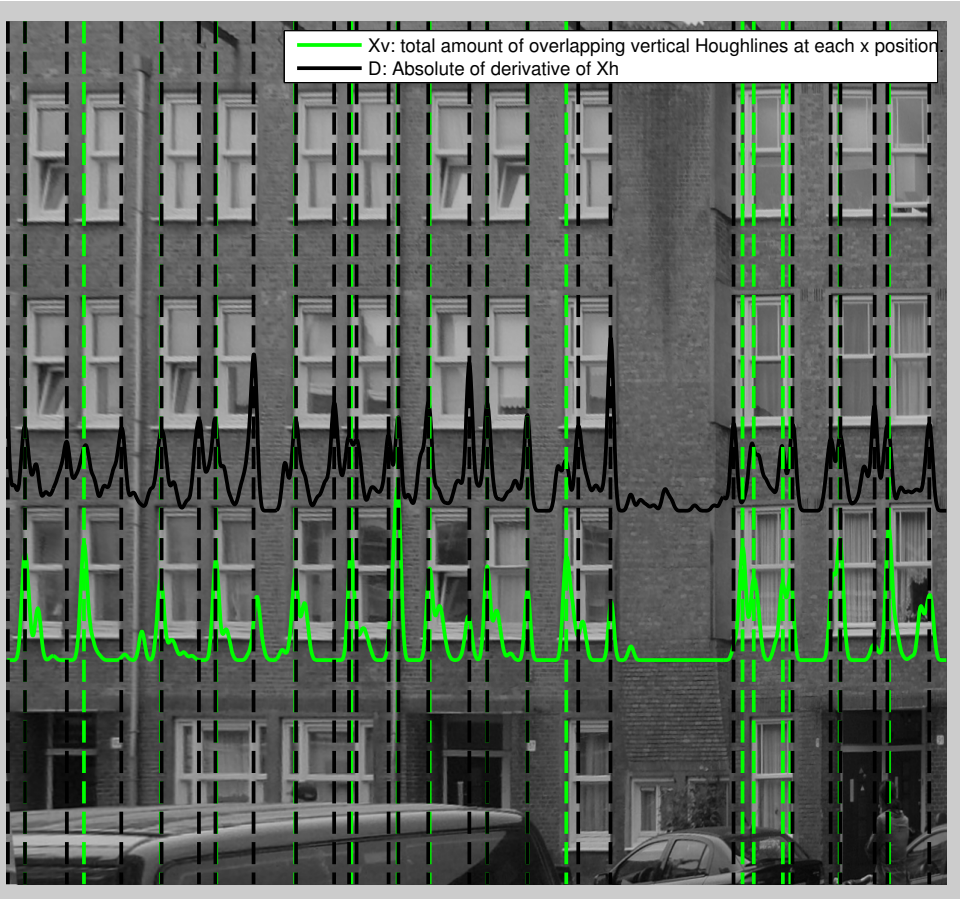

Figure 52: Dataset: Anne2, Regular and alternative window alignment combined

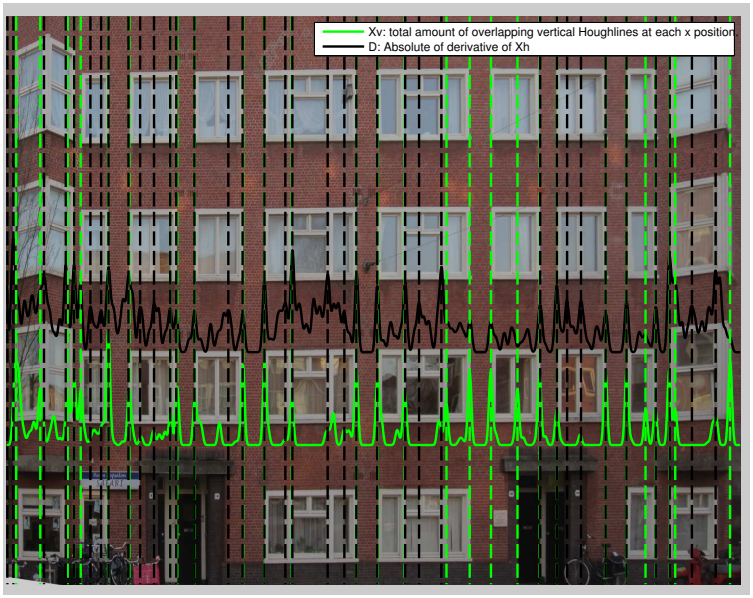

Figure 53: Dataset: Dirk, Regular and alternative window alignment combined

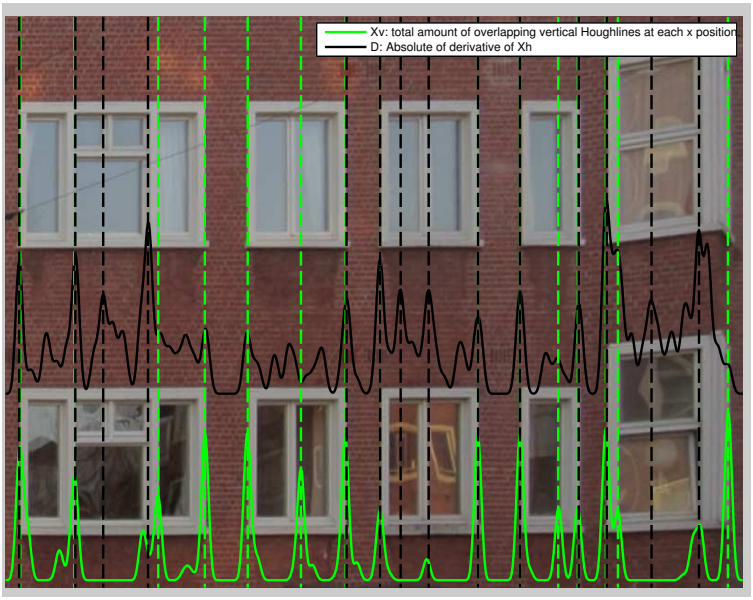

Figure 54: Dataset: Dirk, Regular and alternative window alignment combined, close-up

Discussion

The results are promising, as for both datasets 100 % of the alignment lines are positioned either at the boundary of a window area or inside a window area.

Discussion:Many alignment lines

In Figure 53, too many alignment lines are detected and not every alignment line is placed correctly. This is mainly because the windows are partially aligned: the two front doors contain many vertical edges that are not aligned with the upper windows, therefore a double alignment is found (doors and windows). Another cause of this artefact is that the bicycles on the left (see Figure 75) cause a large amount of found Hough lines which implies a (false) alignment line. Although the causes are clear, these additional window alignment lines cause that lie inside a window area cause no problem because adjacent window areas are grouped by the classification module.
Fortunately no alignment lines are found at non-window areas.

Discussion:Fusing the methods

If we evaluate the independent window detection results on the Anne2 dataset we see (Figure 51, 50) that in both datasets neither the regular window alignment method nor the improved window alignment method detect 100% of the window alignment lines. The effect of fusing the methods is high as after fusing, 100% of the window alignment is found on both datasets: each window alignment line is detected by at least one method. This is basically explained by the fact that both methods are based on a different Houghline direction. We now discuss the interesting cases, where only one of the methods succeeds in detecting the window alignment.

If we take a look at the regular window alignment in Figure 50 we see that for the first 4 window columns the right side of the window is not detected. This is because the original (unrectified) image (Figure 42) is not a frontal image. This makes building wall extensions (middle of Figure 42) or drainpipes occlude parts of the windows. In this case the windows are countersunked into the wall, making the building wall itself occluding the right window frame (Figure 50). The color of the reflection of the window is very similar to the bricks and because the windows and walls are not separated by the window frame, the edge detector doesn't find a strong edge. This means that on all positions where the window frame is missing, no vertical Hough lines will be detected and (as the regular window alignment is based on the amount of vertical Hough lines) no window alignment will be found. This artefact can be studied in 1) the edge image Figure 73 (few or no edges are present), and 2) at the low height of the peaks at these positions (Figure 50).

However, the alternative window alignment does find a window alignment on these positions. This is because the method is based on the opposite Houghline

direction: for the vertical window alignment the horizontal Houghline direction is taken into account. This occlusion artefact has no effect on the horizontal Hough lines, which makes the alternative window alignment method a strong alternative for the alignment of (partially) occluded windows.

In general, the alternative window alignment performs better then the regular window alignment method. This is because this method takes the horizontal window frame parts into account which is a priori stronger as there are more horizontal window parts then vertical window parts present. E.g. in Figure 43 every window has (as it has two vertical sub windows) 3 horizontal window frame parts but only 2 vertical window parts. Furthermore the method is more robust because it relies on a higher level of histogram interpretation (by using the derivative). However, in a few cases the alternative window alignment method is outperformed by the regular window alignment. For example, in Figure 54, the left side of the second window column is not detected. This is because this window type has no horizontal subdivisions. Only the top and bottom of the window frame produce an edge, therefore the derivative of the amount of Hough lines will return a small peak on this position (which is too small to survive the threshold). The threshold is a priori hard to survive because it has a high value as it is determined by taking a fraction of the maximum peak which is located at the windows that do have horizontal subdivisions (for example the most left or most right window column).

Future research

Future research:Determine window alignment of different window types

A solution of the missing window alignment lines on a scene with different window types (Figure 54) would be to decrease the threshold if multiple window types are found. One could design a measure of variety of the window types. This could be done by taking the variation of the derivative of the amount of Hough lines. This amount of variation will determine the amount of decrease in threshold. Let's explain this by two examples: If there is just a few variation the maximum peak is very representative for a window so the threshold could be for example 0.7*max peak. However if there is a large variation is found (which means multiple window types are present), the threshold should be lowered to 0.3*max peak to detect the hard window types (with few horizontal divisions).

Another method would be to cluster the amount of Hough lines in $n+1$ values where n is the number of window types (the 1 is for the non-window area). Areas that transcend from a window area cluster to a non-window area cluster (or vice versa) are determined as the window alignment locations.

For both methods the challenge is again to detect the window alignment with unknown window types but keep the number of false positives (e.g. a drain

pipe) zero. It is also possible to discard the false positives by an additional (e.g. local refinement) method.

Future research:Peak grouping

We fused the result of the horizontal and vertical Hough lines and discarded peaks that where close. A more accurate result would be achieved if close peaks where averaged, the height of the peak could be used as a weight. We used a manual maximum peak group distance (G), this value could also be automatically derived from the image by for example taking a percentage of the window, or by detecting the size of the window frame parts.
It is challenging to handle multiple close peaks, e.g. if 4 peaks are close, then peak 1 could indicate the same window as peak 2 but indicate a different window then peak 4. The location of the close peaks: inside, at the border, or outside the window could add important evidence, some methods of the window classification could be used to discriminate these cases.

Future research:Window alignment refinement

To get more accurate result or to handle scenes with poor window alignment, a refinement procedure could be applied. As mentioned in the related work, Lee et all [14] applied window refinement. Although this comes with accurate results, this approach to iterative refinement comes with a computational expensive procedure.

It would be nice to have a dynamic system that is aware of this accuracy and computational time trade of. A system that only refines the results when the recourses are available. For example if a car is driving and uses window detection for building recognition the refinement is disabled. But if the car is lowering speed, the refinement procedure could be activated. Resulting in accurate building recognition which opens the door for augmented reality.

Both window refinement and window alignment steps could use some additional evidence which could be provided by feature based methods. For example a *multi scale Harris corner detector* could help an accurate alignment or refinement of the windows.

A computational cheaper method would be to apply a 2-stage window alignment procedure. This means we have to extend our algorithm. Let us explain this with an example.
The first stage detects the global refinement using the alignment methods discussed in this thesis. For example for the horizontal window division, we extract only the global division (e.g. the building levels). Next, the second stage determines an accurate window alignment for each floor independently. In this way the other floors don't influence the result which will result in more accurate alignments. The next step window classification will be according to the

approach we discussed.

Note that above concept will also open the door for scenes where the windows are aligned within a certain area but are unaligned with respect to other areas. E.g. a building that contains doors on the ground floor, windows on the first floor and different types of windows on the second floor.

5.6.3 Basic window classification (based on line amount)

The image is now divided in a new grid of blocks based on these alignment. The next challenge is to classify the blocks as window and non-window areas: the window classification. We developed two different methods for this.
Instead of classifying each block independently, we classify full rows and columns of blocks as window or non-window areas. This approach results in an accurate classification as it combines a full blockrow and blockcolumn as evidence for a singular window.
The method exploits the fact that the windows are assumed to be aligned. A blockrow that contains windows will have a high amount of vertical Hough lines caused by the windowframe, Figure 43 (green). For the blockcolumns the number of horizontal Hough lines (red) is high at window areas. We use this property to classify the blockrows/blockcolumns.
For each blockrow the pixels of all vertical Hough lines that overlap with that blockrow are summed up. We refer to this sum as the Hough line overlap. (Remark that with this method we take both the length of the Hough lines and amount of Hough lines implicitly into account.)
To prevent the effect that the size of the blockrow influences the outcome, this total value is normalized by the size of the blockrow.

$$\forall Ri \in \{1..numRows\} : R_i = \frac{HoughlinePxCount}{R_i^{width} \cdot R_i^{height}}$$

Leaving us with $||R||$ (number of blockrows) scalar values that give a rank of a blockrow begin a window area or not. This is also done for each blockcolumn (using the normalized horizontal amount of Hough lines pixels) which is stored in C.
If we examine the distribution of R and C, we see two clusters appear: one with high values (the blockrows/blockcolumns that contain windows) and one with low values (non-window blockrows/blockcolumns). For a specific example we displayed the values of R in Figure 55. Note that the high values, row (4,5), (7,8), (10,11) apear in pairs and correspond to the three rows of windows in Figure 56 and 57.
How do we determine which value is classified as high? A straight forward approach would be to apply a threshold, for example 0.5 would work fine. However, as the variation of the values depend on (unknown) properties like the ratio of window and non-window areas, the amount of subwindows a window contains, etc., this method is not robust for other scenes.

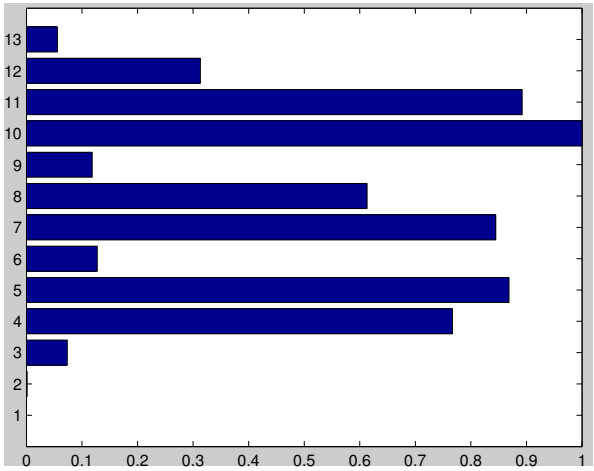

Figure 55: Classification values for window block rows representing the normalized vertical Houghline pixel count of (R)

As the threshold method is classified useless we developed an other method. We use the fact that a blockrow is either filled with windows or not, hence there should always apear two clusters.
Hence, we use *k-means* clustering (with $k = 2$) as the classification procedure. This results in a set of Rows and Columns that are classified as window an non-window areas.

The next step is to determine the actual windows W. A block $w \in W$ that is crossed by R_j and C_k is classified as a window iff *k-means* classified both R_j and C_k as window areas. These are displayed in Figure 57 as green rectangles. The last step is to group a set of windows that belong to each other. This is done by grouping adjacent positively classified blocks. These are displayed as red rectangles in Figure 57.
As the figure gives a binary representation of the windows it is not possible to see how certain a block is classified. To get insight about this, we developed a measure of certainty function.

$$P(R_i) = \frac{R_i}{max(R)}$$

$$P(C_i) = \frac{C_i}{max(C)}$$

$$C(w) = \frac{P(w^{R_i}) + P(w^{C_i})}{2}$$

As you can see C is normalized, this is to ensure the value of the maximum certainty is exactly 1. The results can now be relatively interpreted, e.g. if the

rectangle's $P = 0.5$ then the system nows for 50 % sure it is a window, compared to its best window ($P = 1$). And, as the normalization implies this, there is at least one window with $P = 1$.
The visualization of the measure of certainty is shown in Figure 56, the whiter the area the higher the measure of certainty.

Results

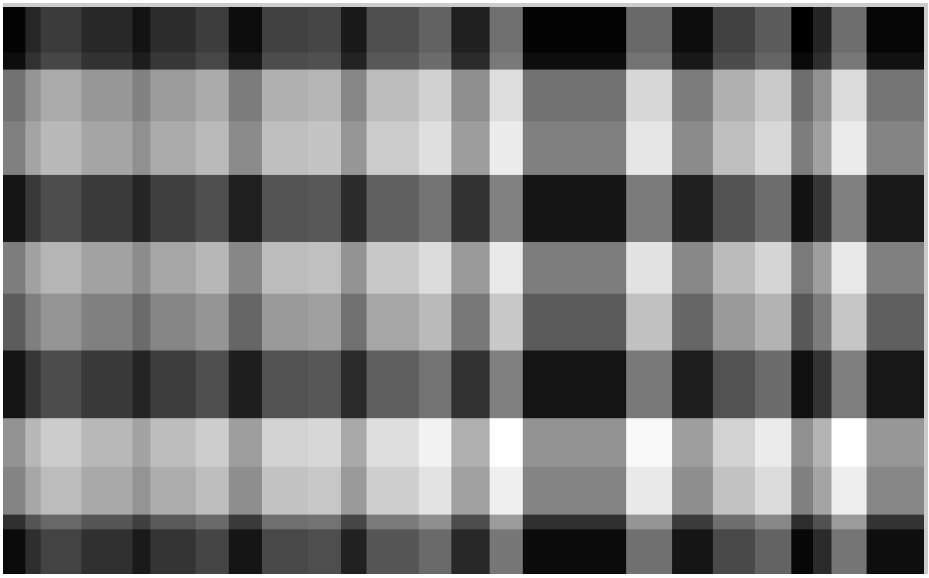

Figure 56: Dataset: Anne1, Basic window classification method:Measure of certainty, the whiter the area the higher the measure of certainty.

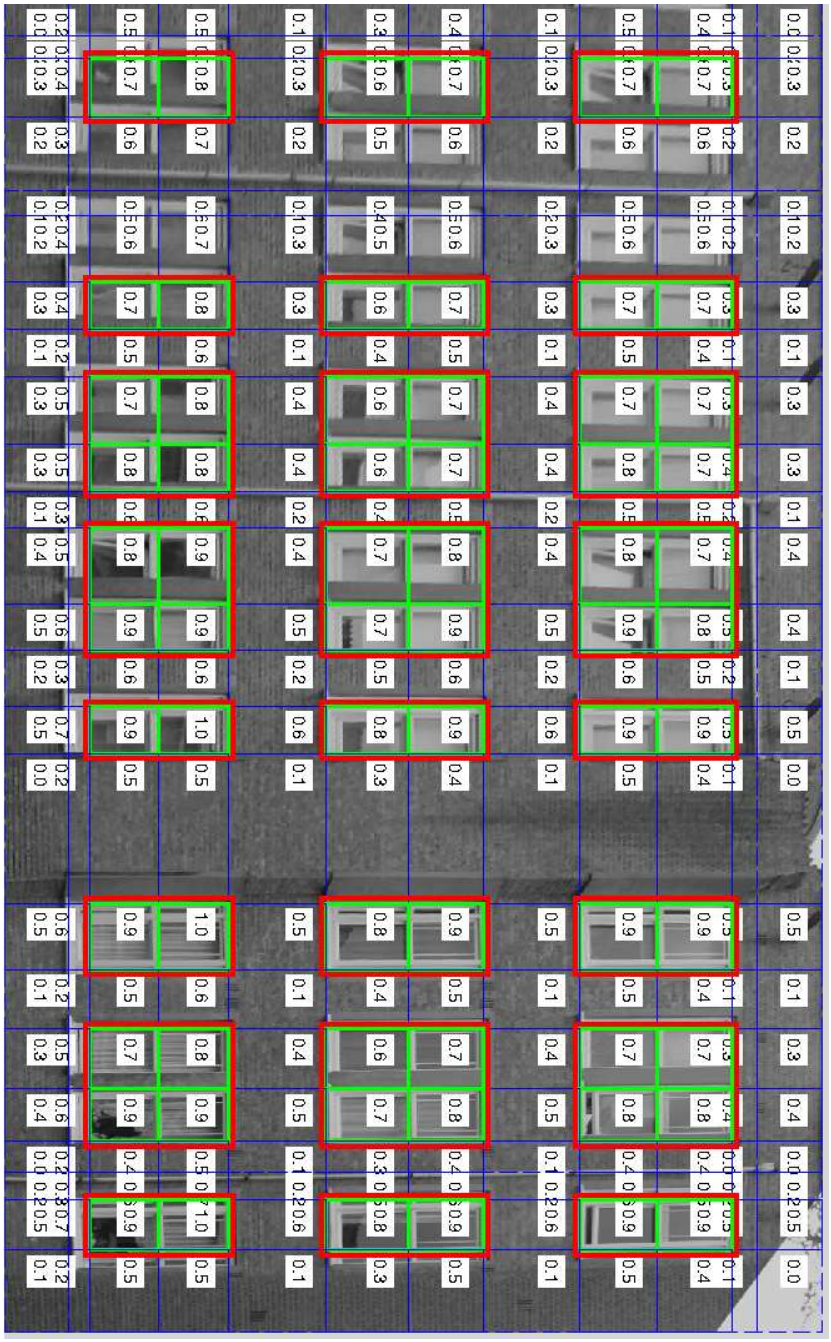

Figure 57: Dataset: Anne1, Basic window classification method: The extracted windows, red:the grouping

The bright rows and columns in Figure 56, indicate window blockrows and blockcolumns, whereas the dark rows and columns indicate non-window areas. The stripe patterns support that the classification process exist of individual row and column classification. The area of window positions is particularly white, as it intersects a bright (positively classified) row and column. One can compare this result to the windows in the original image (Figure 46).

Discussion

The outcome of this method is non-deterministic, as it depends on to the random initialization of the cluster centers. This means that our results could be correct by coincidence. To exclude this artefact, we ran the cluster algorithm 10 times on all datasets. Unfortunately for the Dirk dataset, 2 of 10 times it resulted in a bad result. This result can be found in Appendix Figures 64, 63, 65 and 66. and can be explained as follows. The Dirk dataset contains window types in the middle of that contain sub windows that are seperated by a horizontal subdivision. The other windows don't share this property therefore jhe windows in the middle will cause k-means to drag the cluster center to a value that is too high. This results in windows areas that are classified as non-window areas.

Future research

A solution to the bad luck on the initialization of the cluster centers is to increase the number of cluster centers to $n+1$, where n is the number of window types (the 1 is for the non-window area).

5.6.4 Improved window classification (based on shape of the histogram function)

If we take a look at Figure 58 we see that X_h (the amount of horizontal Hough lines) has two shapes that repeat: At the location where a window is present X_h is concave whereas at non-window areas X_h is convex. This is because a window contains framed sub windows that create a large amount of edges. The number of edges is large at the center of the window. Wheres the number of edges at positions that lie between the windows (the non-window areas) is small. The concave and convex shape of X_h also supports that these values increase towards the window center and decrease towards non-window area centers. This artefact is used for our our second window classifier.

This shape type of X_h is detected as follows: We took a similar approach as in the alternative window alignment (5.6.2). First we examine the derivative of X_h, see the blue line in Figure 58. Next, we investigate the positions where X'_h changes from sign, these are the peaks and valleys of X_h. X_h is concave at the sign changes from positive to negative (+,-) and X_h is convex if the sign changes (-,+).

We expect one sign change per block, however it is possible that multiple sign changes occur. In this case we smooth X_h again and repeat the algorithm until for each block a maximum of one sign change is found.

Now we have detected the shape type (concave or convex), we can directly classify the blockrows and blockcolumns as window areas and non-window areas. The windows are determined as the previous classifier by combining the (positively classified) blockrows and blockcolumns.

For the purpose of clear illustration, only blockcolumns are drawn. The method for the blockrows is almost the same: the projection profiles are projected to the Y-axis.

Results

Results:Anne2 dataset

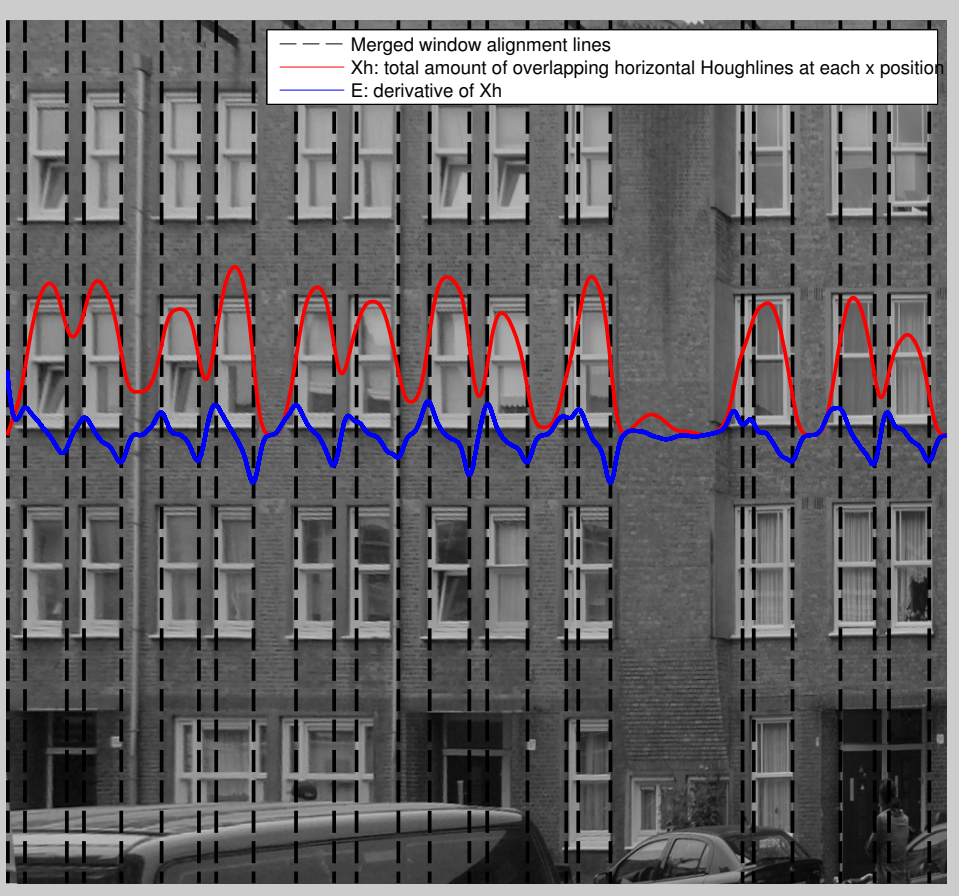

Figure 58: Dataset: Anne2, Improved window classification method: The red line shows concave shapes at window locations and convex shapes at non-window locations

Although the scene contains a lot occlusion artefacts and suffers resolution loss, especially on the left side of the image, the results for the Anne2 dataset, a 100% detection rate and a 100% true positive rate. This means that all windows are detected and that all areas that are classified as windows are indeed window areas.

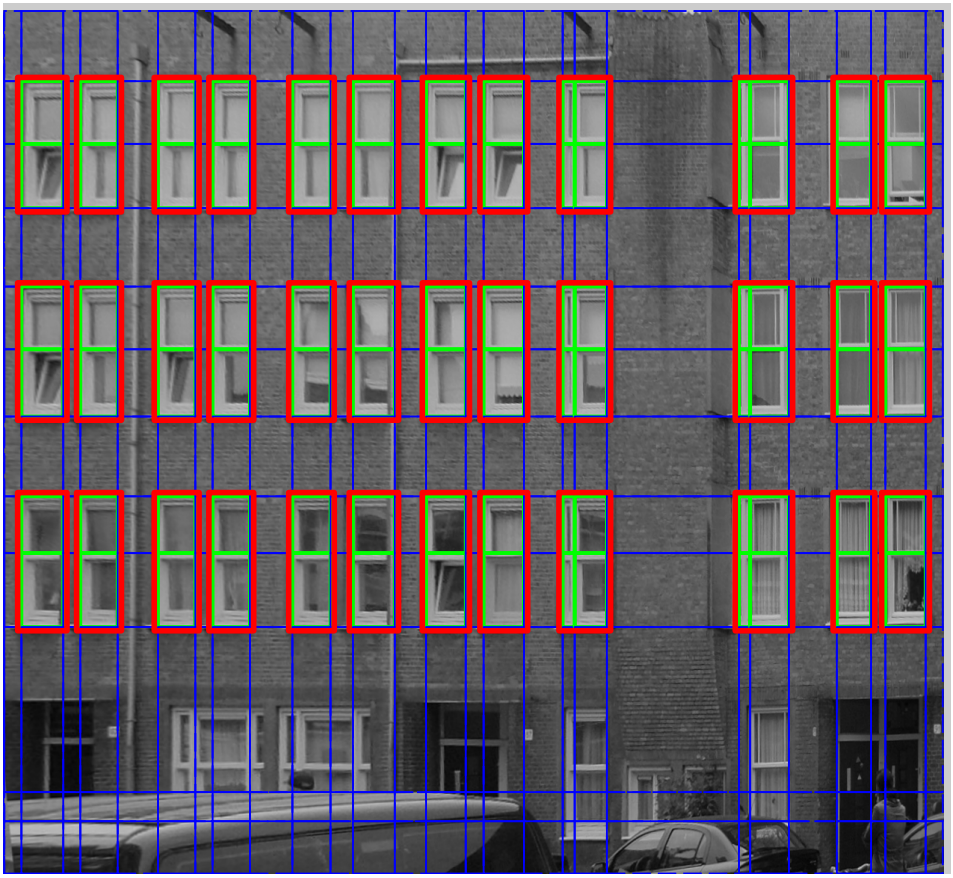

Figure 59: Dataset: Anne2, Improved window classification method: The extracted windows, red:the grouping

Results:Dirk dataset

Figure 60: Dataset: Dirk, Rectified image of a realistic scene which is realistic as it contains light spots, bicycles. Note that the windows are partially aligned and the differ in size and type

The results on the Dirk dataset are also promising given we are dealing with a scene which violates the aligned window assumption and also contains different window types. All windows that are present are detected (100 % Detection rate). 88 % of the detected windows are actually window areas. 12 % from the classified windows are not window areas: these are the doors at the lowest row. 9 % of the areas that are classified as non-window areas should be window areas, this is caused by the unaligned parts of the windows in the first and last column.

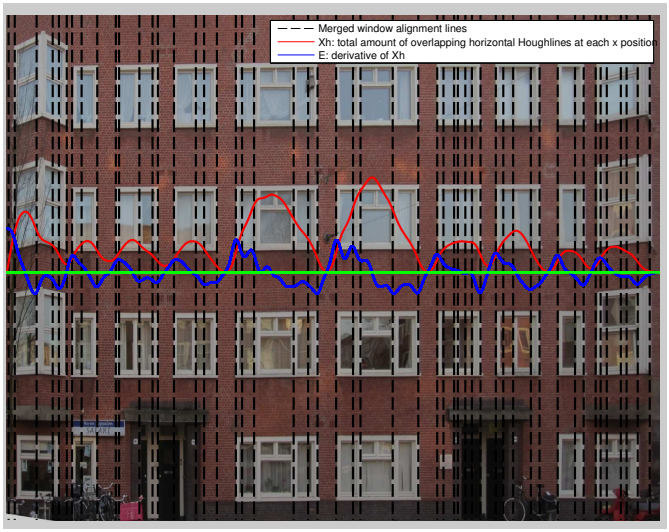

Figure 61: Dataset: Dirk, Improved window classification method: The red line shows concave shapes at window locations and convex shapes at non-window locations

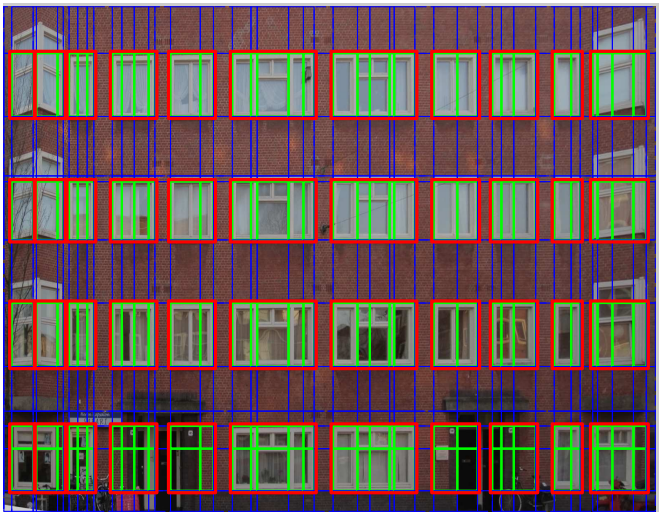

Figure 62: Dataset: Dirk, Improved window classification method: The extracted windows, red:the grouping

Table 5: Improved window classification results on Anne2 and Dirk dataset

Type	Anne2 dataset (Fig 59)	Dirk dataset (Fig 62)
Detection rate	100 %	100 %
True positive rate	100 %	87 %
False positive rate	0 %	11 %
False negative rate	0 %	9 %

Table 6: Grouping results on Anne2 and Dirk dataset

Type	Anne2 dataset (Fig 59)	Dirk dataset (Fig 62)
Grouped correct	100 %	90%
Grouped incorrect	0 %	10%

We discussed in the section about window alignment that the classification module would handle the redundant window alignment lines. This can be seen in Figure 62 where the classification took advantage of the fact that all alignment lines are located at a boundary of a window or in a window. The peak fusing module discarded many close alignment lines and the residual redundant alignment lines create small green adjacent sub windows which where grouped to clear red big windows. This grouping result is promising as 90% of the grouping went well. It is a quite good result given the scene contains a large variety in window shape, window size and window type. This result is mostly due an almost perfect window alignment and classification (as the grouping only groups adjacent positively classified sub windows). The first two window rows in Figure 62 however are classified as two groups but this should be one. Normally the peak merging step would handle this problem, but with this dataset we could not use a large *maximum peak distance* value because the windows in the image (especially the first and last columns) are very close. This brings is to Future research.

Future research

Future work:Grouping error minimization

Although the grouping module gave promising results, it can be optimized. The grouping module now groups adjacent positively classified window areas. Some window in Figure areas are false negatively classified, see the left side of Figure 62. This is caused by a small area between the windows that is classified as a non-window area. This could be solved by adding a minimum size constraint for non-window areas. In this way small negatively classified areas cannot interrupt the adjacent windows.

Future work:Extensive evaluation methods

As the classification worked very well it would be interesting to find out on which point it will fail? This could be investigated by using low quality images (which are taken for example with a cell phone), furthermore we could down-scale the images and add Gaussian noise to the data.

The results of the classification module depend heavily on the result of the window alignment. It would be nice future research to make an independent evaluation of the classification modules. This could be done with a random window alignment generator. Some evaluation method should be developed and it would help if the windows are annotated.

Performance measure for extreme viewing angles

It would be nice to investigate the effect of the occlusion and to examine the robustness of the window detector under extreme viewing angles. For example the viewing angle could be plotted against the percentage of correct detected windows.

5.7 Conclusion

Research question

Is it possible to use edge features to supply an accurate detection of windows in urban scenes?

We discussed two novel window detection methods based on edge detection. The first method is used for scenes where no calibrated input data is available. Despite of the lack of this data it performed really well as it detected 99% of the windows. However the assignment of the window areas could be more accurate. Our second method which uses a calibrated scene. It is based on Histograms of Houghlines and scored a 100% detection rate of the windows.
We now discuss the method independent conclusions.

5.7.1 Method I:Connected corner approach

As can be concluded from our results, the connected corner method is suitable for, and robust to, scenes with a variation in window sizes and types. This makes the connected corner approach suitable for a wide range of window scenes where no or few prior information about the windows is known.
The system has small requirements on the input data because it does not require camera calibration nor image rectification. Furthermore no information regarding the building is required.

5.7.2 Method II: histogram based approach

Histogram based window alignment

We conclude that interpreting the amount of Hough lines is a strong approach towards determination of the window alignment: the results in this work and in previous work are very promising.

We proposed two window detection methods and can conclude that the alternative approach performs better because it is based on a horizontal window division which is more robust to occlusion. Furthermore the alternative approach uses a higher order (the derivative) interpretation of the histogram function.

Occlusion

We showed that window alignment becomes a challenging task if the windows are partially occluded (due to for example a non frontal viewing angle). One of the main solutions we proposed is to use multiple window alignment approaches. To have a 100% detection rate one needs to be certain that if one approach fails at least one other approach must succeed.
We proposed the strong combination of two methods where the first method filled the gap of the second method and the other way around. For example we took care of the window parts that suffer vertical occlusion by a method that detects horizontal window parts. We can conclude that a method that fills the gap of the occluded alignment locations must generally rely on Hough lines that lie in the orthogonal direction of the occlusion.

Relative thresholding fails at differing window types

Furthermore we discussed the implication of the use of different window types in the same scene. We conclude that applying a threshold that is relative to a maximum peak doesn't work well on window types that differ (in for example number of sub windows). This is caused by the relative high maximum peak which is determined by the window with the largest amount of sub windows (the sub windows produce a large amount of edges). The window types with a

small amount of sub windows sub don't survive this threshold which results in missing alignment lines (Figure 54).
This means that if we want to stick with the relative thresholding method, we have to assume a certain equality of the window types. Otherwise an alternative thresholding method should be developed (e.g. a threshold that is altered depending on the window type).

Window classification

We discussed two different window classification methods.

Certainty based window classification

The certainty based window classification works quite good on the dataset we used. However, it requires a very well alignment of the windows which is a disadvantage. As it is based on the number and length of the found Hough lines, the errors in the window alignment will propagate to the classification. This makes this classification method inappropriate for a system where one cannot fully rely on the window alignment.
This conclusion amplified the need for a robust classification method that is less dependent to window alignment errors.

Histogram based window classification

Whereas the certainty based window classification only took the number of Houghlines into account, this approach used the fact that the number of edges, caused by framed sub windows, increases towards the center of the window. We interpreted the Histogram function on a derivative level: and searches for a strong increase or decrease in the number of Houghlines.
The method appears to be very robust as it classified 100% of the windows correct. This is mainly because the increase and decrease pattern is very consistent: it holds for all window and non-window areas in all datasets. It is also robust because the amount of increase and decrease is a relative measure. In contrast to methods that use absolute measures, this approach is robust to a variation of window types. Furthermore we expect it to be robust to change in illumination conditions as it uses relative measures.

We can conclude that this method, based on the shape of the histogram function, outperformed the certainty based window classification.

6 Conclusion

In this thesis we annotated urban scenes using skyline detection and window detection.

We proposed a skyline detection algorithm that is simple has a low complexity and works without any user interaction. Under the assumption that the skyline is the upper edge, the method works well. A set of upper edges should be seen as potential hypothesis which are evaluated using additional features like color and height variation.

Furthermore we proposed two window detection methods that both detect at least 99% of the windows. We conclude that for uncalibrated scenes the connected corner approach performed best as it is robust to variation in window type and viewing angle. For calibrated scenes we conclude that interpreting the amount of Hough lines is a strong approach towards determination of the window alignment. We proposed several alignment methods and conclude that they perform best if we combine them. Furthermore we developed two classification method. The winner is based on the derivative of the Houghlines Histogram function.

In this project we retrieved important semantics of an urban scene: a full 3D construction of the building and the extraction of his windows. Although this is very little in comparison to what humans perceive, we made a valuable start towards human-like interpretation of urban scenes.

The applications grow in the number of correct derived semantics. With just the extracted 3D model and the extracted windows we can recognize buildings, build 3D city models, monitor building deformation and add augmented reality to scenes. Even more application would rise if we add more semantics like house numbers, doors, trees, cars or even (your stolen?) bicycle.

The skyline is the limit.

A Appendices

A.1 K-means bad luck

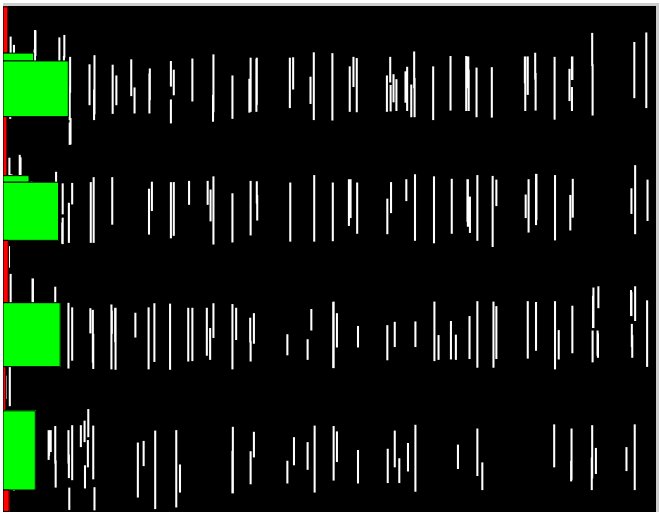

Figure 63: Dataset: Dirk, Vertical Houghlines

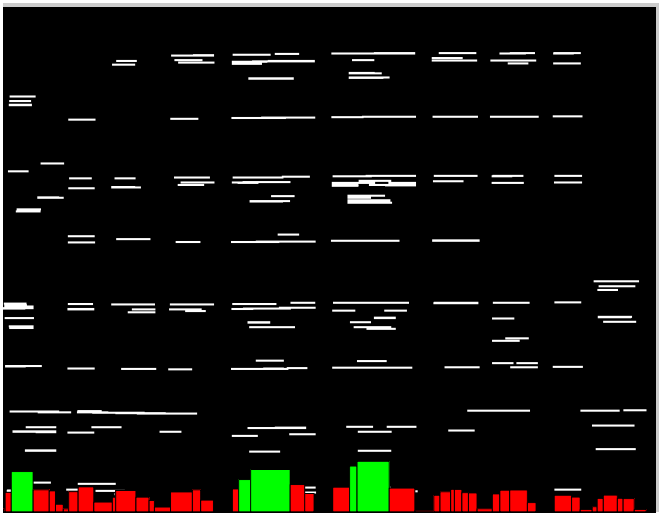

Figure 64: Dataset: Dirk, Horizontal Houghlines

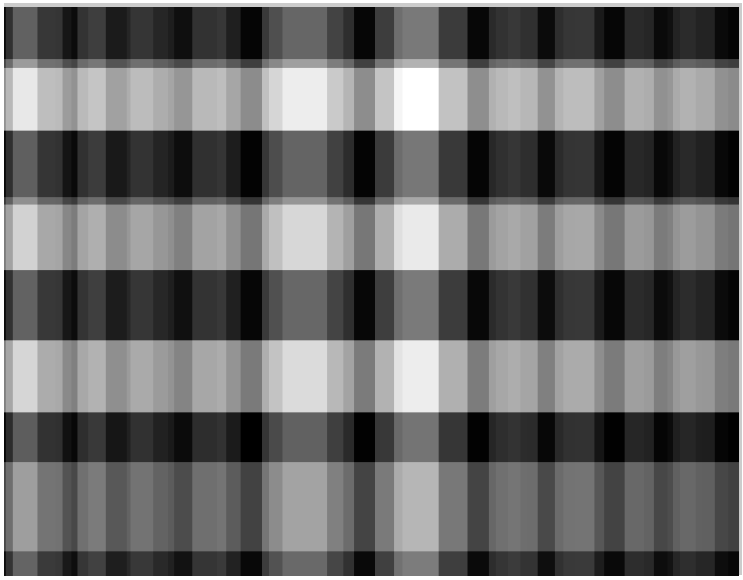

Figure 65: Dataset: Dirk, Basic window classification method:Measure of certainty, the whiter the area the higher the measure of certainty. The values of the window areas in the middle are relative large, this is due the large amount of sub windows.

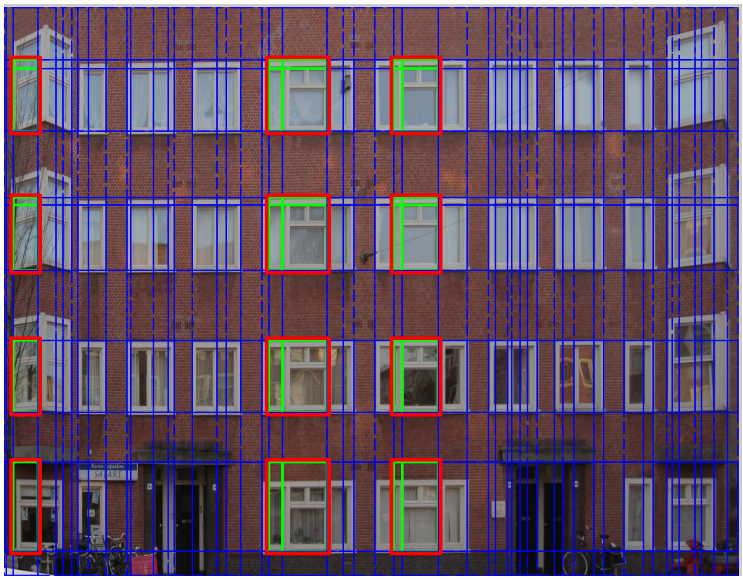

Figure 66: Dataset: Dirk, Extracted windows

A.2 Edge detection results

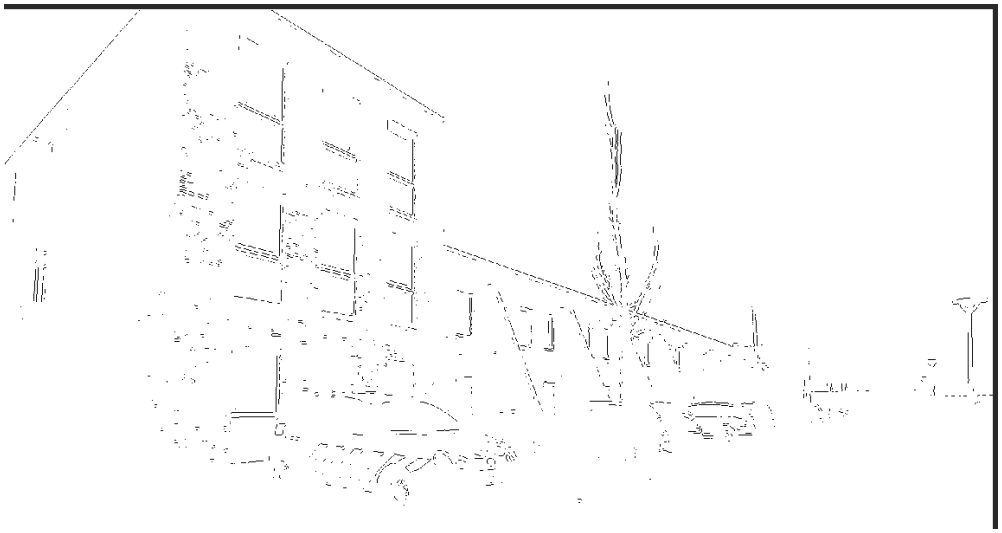

Figure 67: Edge detection results. Method: Laplacian of Gaussian

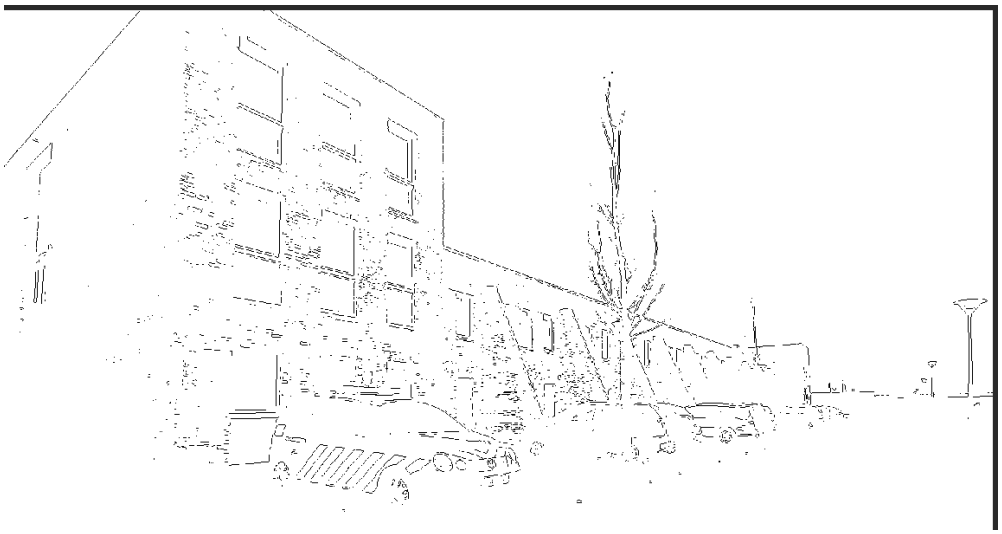

Figure 68: Edge detection results. Method: Prewitt

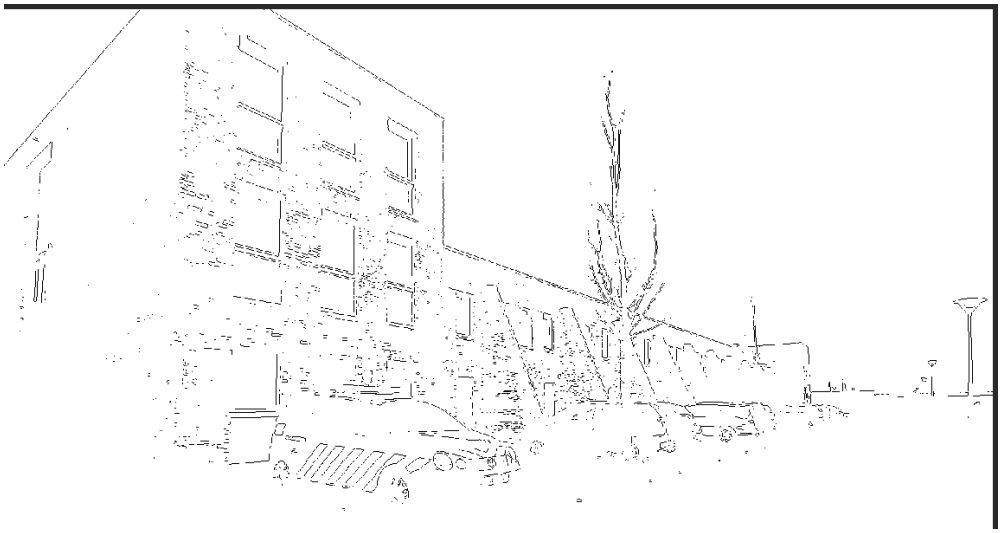

Figure 69: Edge detection results. Method: Roberts

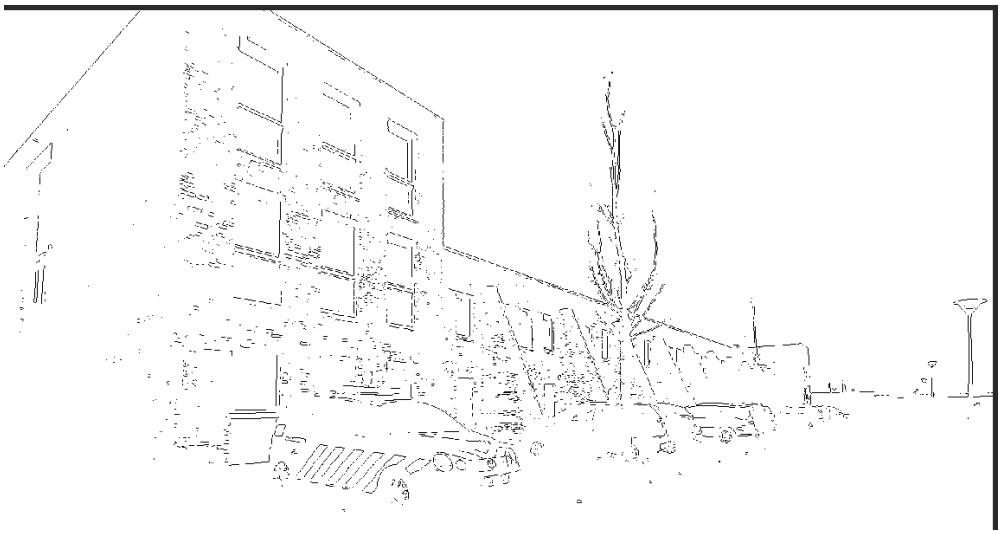

Figure 70: Edge detection results. Method: Sobel

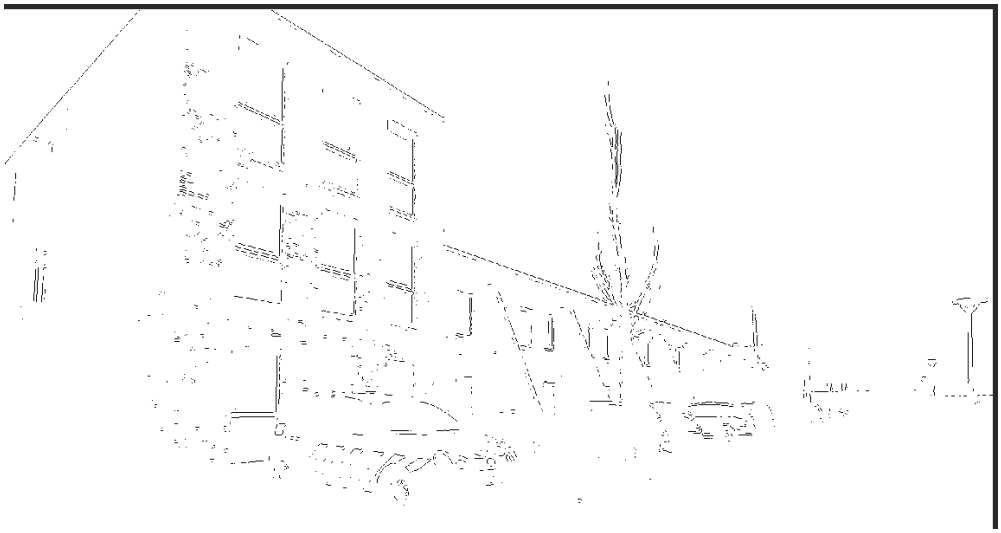

Figure 71: Edge detection results. Method: Zerocross

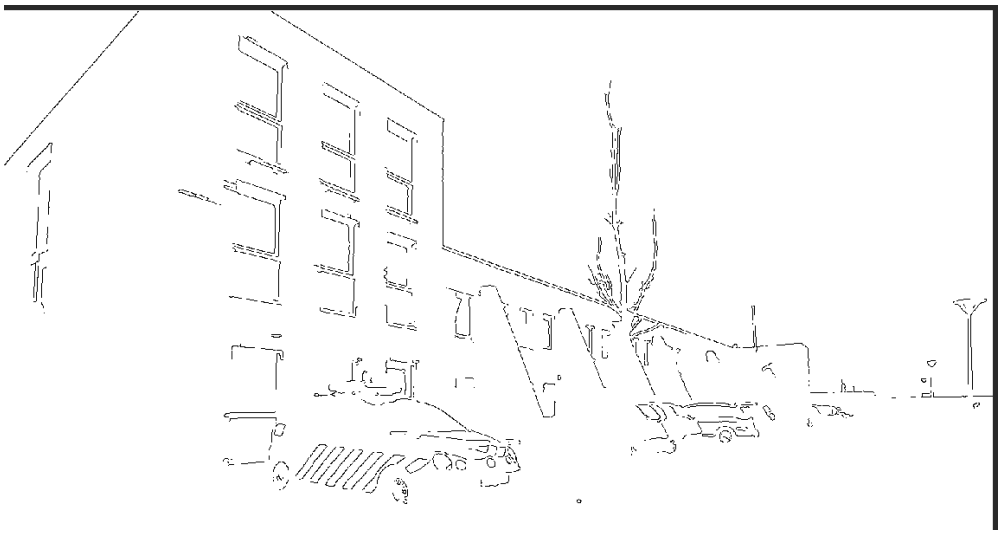

Figure 72: Edge detection results. Method: Canny

A.3 Detailed window detection images

Figure 73: Dataset: Anne1, Result edge detection

Figure 74: Dataset Dirk, Edge detection result

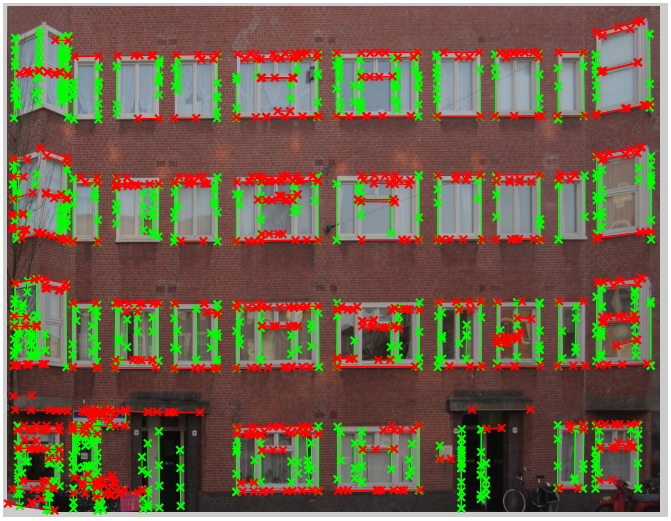

Figure 75: θ-constrained Hough transform

Figure 76: Dataset: Dirk, Regular window alignment: Based on a histogram that displays amount of overlapping vertical Houghlines at each x position

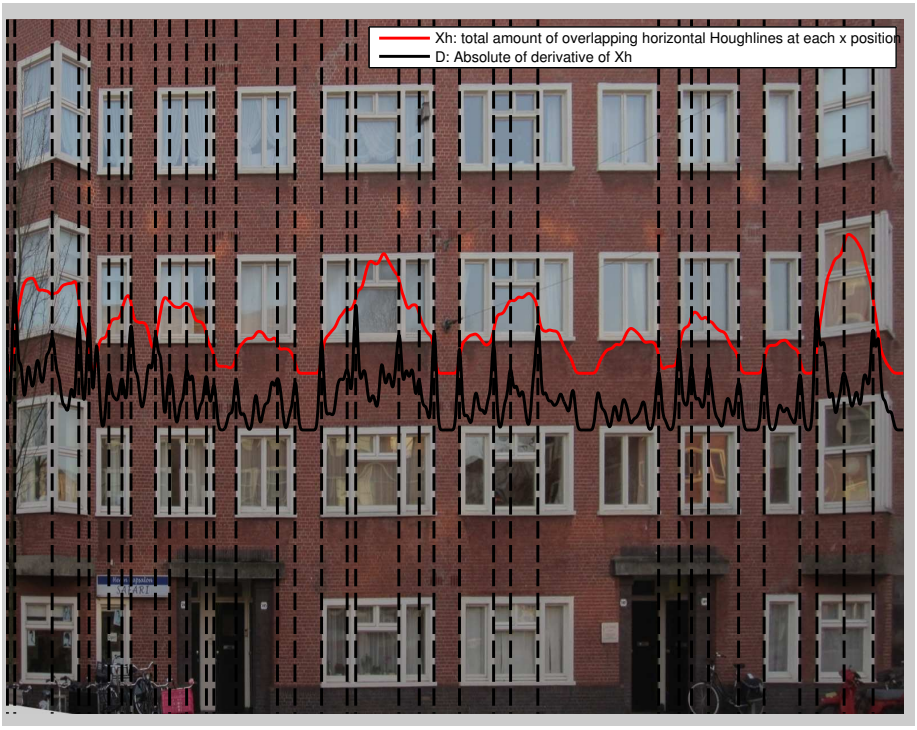

Figure 77: Dataset: Dirk, Alternative window alignment (orthogonal projection): Based on the shape of the smoothed histogram function. For the column division, the number of *horizontal* Houghlines is counted. Peaks (that represent a big decrease or increase of the histogram function) are used for the alignment.

References

[1] Openstreetmap. http://www.openstreetmap.com.

[2] P. d. kovesi. matlab and octave functions for computer vision and image processing. school of computer science & software engineering, the university of western australia. available from:. http://www.csse.uwa.edu.au/~pk/research/matlabfns/.

[3] H. Ali, C. Seifert, N. Jindal, L. Paletta, and G. Paar. Window detection in facades. In *Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on*, pages 837–842, 2007.

[4] John R. Anderson. *Learning and Memory, An integrated approach, 2nd edition.*

[5] Jean-Yves Bouguet. Camera calibration toolbox. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[6] Otto Bretscher. *Linear Algebra with Applications.*

[7] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[8] Andres Castano, Alex Fukunaga, Jeffrey Biesiadecki, Lynn Neakrase, Patrick Whelley, Ronald Greeley, Mark Lemmon, Rebecca Castano, and Steve Chien. Automatic detection of dust devils and clouds on mars. *Mach. Vision Appl.*, 19:467–482, September 2008.

[9] Fabio Cozman, Eric Krotkov, and Carlos Guestrin. Outdoor visual position estimation for planetary rovers. *Auton. Robots*, 9:135–150, September 2000.

[10] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15:11–15, January 1972.

[11] I. Esteban, J. Dijk, and F.C.A. Groen. Fit3d toolbox: multiple view geometry and 3d reconstruction for matlab. In *International Symposium on Security and Defence Europe (SPIE)*, 2010.

[12] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521623049, 2000.

[13] Costin Ionita. 3d models enhancement using gis data.

[14] Sung Chun Lee and Ram Nevatia. Extraction and integration of window in a 3d building model from ground view images. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:113–120, 2004.

[15] MATLAB. *version 7.10.0 (R2010a).* The MathWorks Inc., Natick, Massachusetts, 2010.

[16] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, July 2006.

[17] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM Trans. Graph.*, 26(3), July 2007.

[18] Michael C. Nechyba, Peter G. Ifju, and Martin Waszak. Vision-guided flight stability and control for micro air vehicles. In *IEEE/RSJ Int Conf on Robots and Systems*, pages 2134–2140, 2002.

[19] Michael J. Jones Paul Viola. Robust real-time face detection.

[20] Shi Pu and George Vosselman. Refining building facade models with images.

[21] Michal Recky and Franz Leberl. Windows detection using k-means in cie-lab color space. In *Proceedings of the 2010 20th International Conference on Pattern Recognition*, ICPR '10, pages 356–359, Washington, DC, USA, 2010. IEEE Computer Society.

[22] A. Reiterer. A semi-automatic image-based measurement system. In *In Proceedings of Image Engineering and Vision Metrology, Dredsden, Germany, 2006.*

[23] B. Sirmacek, L. Hoegner, and Stilla. Detection of windows and doors from thermal images by grouping geometrical features. In *Proc. Joint Urban Remote Sensing Event (JURSE)*, pages 133–136, 2011.

[24] Sara Stancin and Saso Toma. Angle estimation of simultaneous orthogonal rotations from 3d gyroscope measurements. *Sensors*, 11(9):8536–8549, 2011.

[25] Robert J. Sternberg. *Cognitive Psychology, Fourth Edition, ch 4 perception.*

[26] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, pages 298–372, London, UK, UK, 2000. Springer-Verlag.