

[DRAFT] 3D RECONSTRUCTION AND
SEMANTIC ANNOTATION OF URBAN SCENES
Supervisors: Isaac Esteban,
Frans Groen

T. Kosteljik
mailtjerk@gmail.com

May 30, 2012

Contents

1 comments/questions	7
1.1 questions	7
2 Introduction	9
3 Preliminaries on Computer Vision	10
3.1 Edge detection	10
3.2 Hough transform	10
3.2.1 Theory	10
3.2.2 Implementation	11
3.2.3 θ constrained Hough transform	12
3.2.4 Matlab parameters	12
3.2.5 Other shapes	12
3.3 Transform coordinates to a line equation	13
3.4 Coordination systems	13
3.5 Homogenous coordinate	13
3.6 Planes and walls	13
3.7 Getting the camera centers and viewing directions	13
3.8 Calibrating the camera with the Bouguet toolbox	13
3.9 Axis angle representation	13

4 Skyline detection	14
4.1 Introduction	14
4.1.1 Related work	14
Detection of dust devils and clouds on Mars	14
Horizon detection for Unmanned Air Vehicles	15
Planetary Rover localization	15
4.2 Method	16
4.2.1 Requirements situation and assumptions	16
Requirements	16
Situation	16
Definition: skyline in urban scene	16
Top sharp edge assumption	17
4.2.2 Related algorithm	17
4.2.3 Improved algorithm	17
4.3 Results	20
4.4 Edge detection	20
4.5 Skyline detection	24
4.5.1 Datasets	24
4.6 Discussion	27
4.7 Conclusion and Future work	27
5 Improving the 3D building	29
5.1 Introduction	29
5.2 Generating the 3D model	29
5.3 Extracting line segments	30
Assuming a flat roof	31
5.3.1 Hough transform	31
5.4 Project the skyline to the building	32
5.4.1 Camera calibration	32
Extrinsic parameters	32
Intrinsic parameters	32
5.4.2 From image point (2D) to possible points in scene (3D)	33
5.4.3 Intersecting with walls	33
5.4.4 Associating line segments with building walls	34
Assumptions	34
No overlap	35
Partial overlap	36
Full or no overlap	37
Line-wall overlap normalization	38
5.4.5 Improving the 3D model by wall height estimation . .	38

5.5	Results	39
5.6	Discussion	40
5.7	Conclusion	41
5.8	Future work	42
5.8.1	Alternative roofs	42
6	My old method of line-wall association	44
7	Window detection	46
7.1	Introduction	46
7.2	Application examples	46
3d City models	46	
Historical buildings documentation and deformation analysis	46	
Interactive 3d models	47	
Augmented reality	47	
Building recognition and urban planning	48	
7.3	State of art window detection	48
7.3.1	Alternative approaches on window detection	48
7.3.2	Similar approaches	50
7.3.3	Our work in context	52
7.4	Method I: Connected corner approach	52
7.4.1	Situation and assumptions	52
7.4.2	Edge detection and Houghline extraction	52
7.4.3	Connected corners extraction	53
7.4.4	Window area extraction	54
7.4.5	Results	55
7.4.6	Conclusion	58
7.4.7	Future research	58
7.5	Facade rectification	59
7.5.1	Introduction	59
7.5.2	3D plane extraction	59
7.5.3	Efficient Projecting	60
7.5.4	Results	62
7.6	Datasets	65
7.6.1	Anne1	65
7.6.2	Anne2	66
7.6.3	Dirk	67
7.7	Method II: Histogram based approach	68
7.7.1	Introduction	68

	Situation and assumptions	68
7.7.2	Extracting the window alignment	68
	Regular window alignment	68
	Alternative window alignment	70
	Fusing the window alignment methods	70
	Results	72
	Discussion	77
	Conclusion	78
	Future research	79
7.7.3	Basic window classification (based on line amount) . .	81
	Results	84
	Discussion	86
	Future research	86
	Conclusion	86
7.7.4	Improved window classification (based on shape of the histogram function)	87
	Results	88
	Conclusion	93
	Future research	94
7.8	Conclusion	95
8	Conclusion	96
9	Appendices	97
9.1	Figures	97
9.1.1	K-means bad luck	99

Todo list

TODO: , explain 2 coords to an line equation	13
TODO: elaborate	13
TODO: explain how to add extra homogeneous dimension (x,y,1) and why this is	13
TODO: How to span plane by wall coords	13
TODO: Explain here about fit3d toolbox	13
TODO: TODO see ref 15 and fit3d toolbox p4	13
TODO: Axis angle representation, wikipedia	13
TODO: Zorgen dat paragraphs ook 3.5.3.1 nummering krijgt	46
TODO: in state of art een lijn creeren, opbouwen, in context plaatsen, theoretisch afwimplene	48
TODO: research question	50
TODO: 2	53
TODO: 2	55
TODO: Met iemand overleggen wat betere volgorde is	64
TODO: 2	68
TODO: 2	68
TODO: 2	71
TODO: 2	71
TODO: result1 drain pipe result niet erg ivm vlg stap	78
TODO: make diagram: edge- hough -window alignment - window clas- sificatio - window grouping	79
TODO: diagram maken window alignment - ζ peak merging - ζ etc . . .	79
TODO: DIT PLAATJE eruit en mooie staafdiagrammen erin	81
TODO: ref	82
TODO: include results of binary classification	86
TODO: include table of false positives etc, detection rate etc. etc. misch staat dit nog op github	86
TODO: introduce a little bit	87
TODO: say that anne2 dataset is more rectified	87
TODO: make a diagram, histogram - alignment lines - ζ window clas- sification etc.	88
TODO: discuss drain pipe, dat het goed is dat het resultaat mooi is ondanks drainpipe	88
TODO: IMPROVED result on Anne1 dataset, investigate which combi- nations of dataset andmethods i miss	90
TODO: certainty based (white strokes) classification on anne2 dataset, pas als skyline af is, img zijn wel al klaar	93

TODO: remark that we goedkeuren partieel resultaat van windows helemaal rechts en links dirk dataset	93
TODO: Interpretation of the results of the Anne1 dataset	93
TODO: compaire to other classification methods	93
TODO: (perform both classifications on both datasets (2x3 img), visual concatinate images vertically)	93
TODO: especially the horizontal division of the lowest row of windows is amazing, refer to	93
TODO: Meer over lof spreken	93
TODO: put a opbouwende line in future work	94
TODO: speak about video, or sequence of images, detection + tracking etc	94
TODO: compare classification methods and explain differences	95
TODO: We showed that projecting the image to a frontal view is a good preprocessing step of a robust window detector.	95
TODO: Furthermore our algorithms work in real time. 95	
TODO: check if appendix images not already in ch5!!	97

1 comments/questions

1.1 questions

Ik heb er voor gekozen om de resultaten, discussie en conclusie van de window alignment individueel te bespreken
voor window classificatie methode 1 + 2 heb ik dit samengevoegd. is dit een wijze keuze?

Abstract

Here comes the abstract

2 Introduction

3 Preliminaries on Computer Vision

3.1 Edge detection

3.2 Hough transform

3.2.1 Theory

A widely used method for extracting line segments is the Hough transform [5]. In the Hough transform, the main idea is to consider the characteristics of a straight line not as its image points $(x_1, y_1), (x_2, y_2)$, 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 pair 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 straight line in (x, y) space appears as a sinusoidal curve in the Hough parameter (r, θ) space. Let's see an example, the following image is transformed into the space (r, θ) .

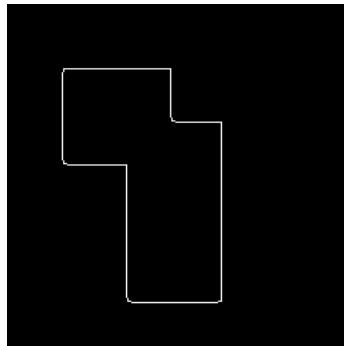
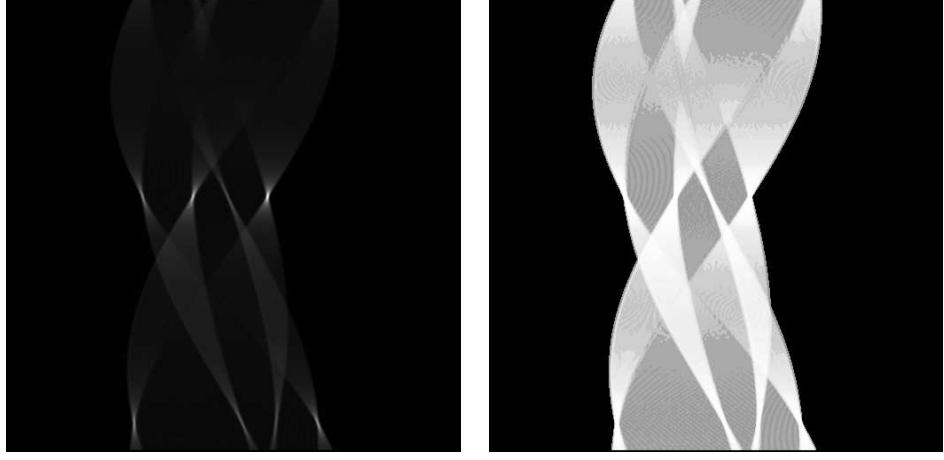


Figure 1: An input image, consisting of eight straight lines, for the Hough transform



(a) (r, θ) values

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

Figure 2: Hough transform

As you can see for every edge point in Figure 1 a curve is generated in (r, θ) space in Figure 2(a). On eight positions (white dots) the number of intersecting sinusoidal curves is high, these position correspond to the eight separate straight line segments in Figure 1 .

This makes the problem of detecting straigth lines in finding peaks in the Hough parameter (r, θ) .

3.2.2 Implementation

The input of a Hough transform is a binary image, in our research it is the output of the skyline detector (4) or in the case of window detection (7) the output of an edge image.

The Houghtransform 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 2(a) are the found straigth lines in the (x, y) space.

These positions are found by looking for local maxima in the accumulator array.

3.2.3 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, furthermore with window detection one 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 of a photograph of a building taken by a user, $\theta = [-10..0..10]$. Although only $\theta = 0$ presents an exact horizontal line we broaden the interval because the user hardly ever holds the camera exactly orthogonal.

3.2.4 Matlab parameters

We used a standard Matlab implementation of the Hough transform for straight lines. This implementation comes with some interesting parameters:

The *minimum length* 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.

3.2.5 Other shapes

The Hough transform isn't bound to finding straight lines, in fact it could be applied to any space which can be represented by a set of parameters. For instance, a circle can be transformed into a set of three parameters, (x, y, r)

representing its center and radius, this makes the Hough space three dimensional. For more complicated shapes, the *Generalised Hough transform* is used. It uses a feature that votes for a particular position, orientation and/or scaling of a certain predefined shape. The shapes are predefined in a look-up table.

3.3 Transform coordinates to a line equation

TODO: , explain 2 coords to an line equation

3.4 Coordination systems

The camera can be described in the same coordinate system as the 3D point we are looking for, we call this the world coordinate system. The camera center is a value in \mathbb{R}^3 that represents the camera's position in the world coordinate system. If the camera rotates, it rotates around this point.

TODO: elaborate

3.5 Homogenous coordinate

TODO: explain how to add extra homogeneous dimension (x,y,1) and why this is

3.6 Planes and walls

TODO: How to span plane by wall coords

3.7 Getting the camera centers and viewing directions

TODO: Explain here about fit3d toolbox

3.8 Calibrating the camera with the Bouguet toolbox

TODO

TODO: TODO see ref 15 and fit3d toolbox p4

3.9 Axis angle representation

TODO: Axis angle representation, wikipedia

4 Skyline detection

4.1 Introduction

If we take a regular image on which both sky and earth 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, guiding micro air vehicles, car localization, etc.

In this project it is used in the process of combining 2D urban images to calculate a 3D model of the scene. The role of skyline detection is to estimate the heights of the facades of the 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.

4.1.1 Related work

Castano et al. [3] present a clear introduction of different skyline detection techniques.

Detection of dust devils and clouds on Mars

In [3], Mars Exploration Rovers are used to detect clouds and dust devils on Mars. Their approach is to first identify the sky and then determine if there are clouds in the region segmented as sky. The sky is detected by an innovative algorithm that consists of three steps. First they place seeds in a sliding window whenever the homogeneity of the window is high. Then they grow these seeds in the direction of edges which are estimated using a Sobel edge detector. Finally each pixel located above the grew seeds is classified as sky.

Of the discussed methods so far, this seed growing method looks like the most sophisticated one, as it is accurate and autonomous. However, 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 this domain [9], scientists detect the horizon to stabilize and control the flight of Unmanned Air Vehicles.

S.M. Ettinger et all [9] 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 work. But it needs to be mentioned that some ideas for section 5.3 are created because the building has walls that have straight lines, an assumption is made about partially straight lines.

Planetary Rover localization

Cozman et al. [4] 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) and align both images. The matching process was first based on feature matching. In an improved version the matching process was done by searching for correspondences among dense structures in the image and on the given map, so called signal based matching.

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. Furthermore it has to be robust and accurate because it provides a basis for the extraction of straight lines which offer an estimation of the buildingwall heights.

We decided to use the Rover method [4] as a basis and build a custom algorithm with higher accuracy on top of that. This is explained in the next section.

4.2 Method

4.2.1 Requirements situation and assumptions

Requirements

For our application the skyline detector must be accurate, robust and must operate without any user interaction. These requirements makes it different from existing skyline techniques (e.g. [3],[9],[4]).

Situation

Before we present the method let's 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 than the color of the building. A color-based edge detector would be an intuitive decision as this produces edges on regions where intens colortransitions 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 intens 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 sharper color transitions. Furthermore it would not be robust to a change in the lightning conditions, influenced heavily by the weather.

To solve this problem we draw an assumption that is based on the idea of [4]. 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.

4.2.2 Related algorithm

As our algorithm is based on a related algorithm presented in [4], this is 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 colortransformation and finally it searches for the highest transformation.

The preprocessing step is used to increase the difference between sharp and vague edges, and to let sharp edges stand out more and vague edges disappear. The preprocessing consist of a Gaussian smoothing operator on the input image. Next the smoothed image is sliced in $\#w$ pixelcolumns each column represents the values of a discretized function. Each column is transformed to the derivative of this column, called the smoothed intensity gradient. The values of this column are high when a big change in color happens (e.g. 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.

4.2.3 Improved algorithm

Although taking the smoothed intensity gradient is a computational cheap way to detect edges it has a big disadvantage as it is not robust to vague edges because they don't survive the threshold. It is not surprising that the algorithm in [4] 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 provide the system some parameters. Furthermore also 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 types 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 [2] 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 includes the weak edges in the output, but only if they are connected to strong edges. In Table 1 we list Matlab's build in edge detectors together with the method explanation.

In the section 4.4 one can find the result of the different edge detection methods.

Table 1: Different edge detectors explained

Edge detecting type	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.
Laplacian	The Laplacian of Gaussian method finds edges by looking for zero crossings after filtering the image with a Laplacian of Gaussian filter.
zero-cross	The zero-cross method finds edges by looking for zero crossings after filtering the image with a filter you specify.
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.

Because the optimal edge detector type can be scene depended, it can be set by the user as a parameter in our functions.

The Canny edge detector outputs a binary image, therefore the column inner 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 blur, this removes a large part of the noise. Note that depending on the edge detector type this could mean that the image is blurred twice.

The system now has several parameters which have to be set manually by the user:

- Contrast,
- Intensity (window size) of Gaussian blur,
- Edge detector threshold.

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

4.3 Results

4.4 Edge detection



Figure 3: Edge detection results. Method: Laplacian of Gaussian



Figure 4: Edge detection results. Method: Prewitt



Figure 5: Edge detection results. Method: Roberts



Figure 6: Edge detection results. Method: Sobel



Figure 7: Edge detection results. Method: Zerocross



Figure 8: Edge detection results. Method: Canny

4.5 Skyline detection

4.5.1 Datasets

The skyline detection algorithm was applied on three datasets.

Table 2: Dataset properties

Name	Resolution	Source	Location
Floriande	1728x1152px	Fit3d [6]	Unknown
Bram	3072x2304px	Author	Amsterdam, 'Postjesweg'
Anne	3072x2304px	Author	Amsterdam, 'Van spilbergenstraat'

The output of the edge detector and skyline detector on the *Floriande* dataset [6] can be seen in Figure 9. We emphasize the effect of different thresholds of the edge detector on the *Bram* dataset (Figure 10) and the *Anne* dataset (Figure 11).



Figure 9: The output of the skyline detector. The skyline elements are marked red.



(a) Because of the hanging streetlight, a large part of the building is not detected. Threshold=0.3



(b) The desired part of the building is detected. Threshold=0.7

Figure 10: The skyline detector on two different thresholds



Figure 11: The output of the skyline detector with a too high threshold (0.70)

4.6 Discussion

Consider Figure 9, the largest part of the building edge is detected. This is a desired result, given the algorithm operates without any user interaction. The system assumes that the first sharp edge (seen from top to bottom) is always the building contour. This is why not every skyline element is placed on the building contour but placed on, for example, a streetlight or a tree. We define them as outliers. Other sharp edged objects that appear above the building, for example an aircraft, will also turn into outliers. This is a disadvantage of the column based method.

However, not every object above the building becomes an outlier, as can be seen in Figure 10 a change in the threshold parameter of the edge detector can erase tough outliers. In Figure 11 the risk of using a too high threshold is shown. Although increasing the threshold to 0.7 removed the streetlight outliers in, the results on this scene are very bad.

We decided to keep the threshold low and developed a separate module to remove the outliers. It is not realistic to assume full absence of sharp objects above the building, therefore we don't. Furthermore this is the part where we can add some artificial intelligence. This is described in the next section.

4.7 Conclusion and Future work

A detailed research on related work research was done. We introduced a novel application of skyline detection: the extraction of a building contour. We build an algorithm on top of a successful existing algorithm. The algorithm doesn't depend on human intervention and is robust and accurate enough to provide a base for the next module in the system.

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

Although the outlier removal procedure is done in a separate module, it would be interesting (future work) to develop a skyline detector which is more robust to outliers. Most of the related work is based on detecting parts that are classified as sky and parts that are classified as ground. The idea of detecting the sky and ground could be replaced by detecting the sky and a building. The distinctive textures of the buildings (repeating bricks) could be of great use for the classification. After that, a rough building contour could be estimated by using the highest building pixel for every

column. In this neighborhood, detailed edge detection could be done. In this way the outliers (e.g. the streetlight and the tree) are filtered and no secondary outlier removal procedure is needed.

5 Improving the 3D building

5.1 Introduction

In the previous chapter we extracted the building contour with the skyline detector. The output was a set of 2D points and we collected this set for every view of the building. The aim of this chapter is to use this set of points to improve a basic 3D model.

The point cloud from the skyline detector included a lot of noise caused mostly by occluding objects like trees. How do we detect those outliers? And if we have an outlier free point cloud how can we use this information to improve a basic 3D model? And how do we know which point is associated to which part of the building? These questions are addressed in this chapter.

We present a stepwise solution. First *Openstreetmap* is used to generate a basic 3D model of a building. Secondly the set of points returned by the skyline detector is transferred to a set of lines. Then each line segment is assigned to a wall of the building. After this the lines are projected to these assigned walls in the 3D model. The projections are used to estimate new height values of the building walls. The 3D model is finally improved by updating the walls according these heights.

We will now elaborate on each step.

5.2 Generating the 3D model

The 3D model is generated using a basis (groundplane) which is manually extended. The basis (viewed from top) of the generated 3D model is originated from *Openstreetmap*.

Openstreetmap, see Figure 12, 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. We take a snapshot of one 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 walls.

Because the map is based on aerial images, it is in 2D and contains no information regarding the height of each wall.

The final 3D model is generated by starting with the the 2D building contour as its basis. We estimate the wall heights by hand and overestimate this height. After this we use the height to extend the 2D basis in the opposite gravity direction, resulting in a 3D model.



Figure 12: Openstreetmap with annotated buildings

Gravity aligned walls assumption

*The walls of the building are aligned in the opposite direction of the gravity which is orthogonal to the 2D basis from **Openstreetmap**.*

An example of the 3D model can be seen in Figure 14.

5.3 Extracting line segments

The skyline detector returned a set of points. If some of these points lie on the same line, they form a straight line. Straight lines are likely to come from the building contour. If we have a method that extracts these straight line segments, we can use these line segments to find parts of the building contour and finally use this to improve the 3D model.

Unfortunately a problem arises because some points are outliers. To discard these outliers we detect the inliers and consider the remainder as outliers. In this section we draw an assumption and we explain how straight line segments are extracted and how outliers are discarded at the same time.

Assuming a flat roof

Many urban areas contain buildings with a flat roof. This means that the contour of the building is mostly formed by straight lines. We use this fact to simplify our problem:

Flat roof assumption

We assume each building has a flat roof, implicating that each building wall has a straight upper contour. The walls may have different heights but the roof should be flat.

This assumption is very useful as it let us focus on finding the height of the building walls from the building contour without having to concern for (complex) rooftypes. This doesn't mean that the method described in this thesis is unusable for buildings that contain roofs. E.g. with a small adjustment the method could be used to at least gain the building height which is a useful application.

Ideas about how to handle rooftypes explicitly can be found in the Future work section.

5.3.1 Hough transform

A widely used method for extracting line segments is the Hough transform [5]. We regard this as a suitable method because as it is used a lot for this kind of problems. This is probably because it is unique in its low complexity (compared to other (iterative) methods like *RANSAC*).

For a detailed explanation of the Hough transform please read 3.2. The input of a Hough transform is a binary image, in our case the output of the skyline detector (chapter 4).

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 is most likely that it returns parts of the skyline and therefore the building contour.

The Hough transform is implemented in *Matlab* which has some useful extra functions. The algorithm can optionally return the start and endpoint of

the found lines which is very useful as it helps us to associate which part of the building is described by the line.

Furthermore it has 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.

Results of the Hough transform on the output of the skyline detector are displayed and evaluated in the Result section.

5.4 Project the skyline to the building

The Hough transform of the previous section returned a set of 2D line segments which present parts of the skyline. If we can find a way to project these lines to the building we can improve our basic 3D model. This process involves three steps, the calibration of the camera, the projection to the building and a way to associate a houghline with a specific buildingpart.

5.4.1 Camera calibration

The Camera calibration exist in finding the extrinsic and intrinsic parameters.

Extrinsic parameters

The extrinsic parameters are the centers (positions) and viewing directions of the camera that took the pictures. These are unique values for every image. In some systems these are recorded by measuring the cameras position and orientation at the scene. In other systems this is calculated afterwards from the images. We calculated the values also afterwards and used the *Fit3d toolbox* [6] for this. See preliminaries 3.7 for detail.

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 dataset of the Fit3d toolbox [6] comes with K. For the *Anne* dataset we retrieved these values by a method of Bouguet. The calibration matrix is retrieved by

making images of a chessboard in different positions and orientations and using the Bouguet toolbox. More on this in the preliminaries 3.8.

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

Now lets see what we can do with the calibrated data. 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. The value of the 3rd dimension represents the distance from the 3D point to the camera that took the picture. Unfortunately this value is unknown, however because we calibrated the camera we can reduce the possible points in 3D space to a line.

We know:

- The 2D location of the pixel
- The camera's internal parameters (K)
- The translation of the camera and therefore the location of the camera center
- The rotation of the camera

We can now define the line of possible points in 3D space by two coordinates:

- C , the location of the center of the camera, and
- $K'p$, the point that lies on the retina of the camera, where K is the Calibration matrix and p is the homogeneous pixel coordinate.

Now we have the two required coordinates, we can set up an equation of the line of possible points in 3D. This is done as described in preliminaries 3.3.

5.4.3 Intersecting with walls

The 3D building model is divided into different walls. These walls are described by two ground coordinates and a direction which is the y-direction as assumed by the *Gravity aligned walls assumption*.

As you can read in preliminaries 3.6 the walls are transformed into planes. This is done for two reasons: first this transformation is required to calculate

the intersection properly. Secondly, because the 3D model is an estimate, the walls maybe just to small which could result in no intersection which is not what we want.

Now we have the building divided up into planes we can intersect lines with these planes.

For every line segment returned by the Hough transform we have two endpoints. The endpoints are transformed into lines (which represent the possible points in 3D) and these lines are intersected with the planes of the building walls. This results in $2 \times l \times w$ points in 3D, grouped by the line segments, where l is the number of lines and w is the number of walls.

The next challenge is to reduce the number of intersection for every line segment to one. In other words, to determine the wall that is responsible for the line segment. This is later on used to update the height of the wall of the 3D model.

5.4.4 Associating line segments with building walls

Assumptions

We consider each building consisting of separate walls and associate each line segment with a wall of the building that is most likely responsible for that line segment.

Unique wall assumption:

We assume that the output of the Hough transform are line segments that each represent a single wall of the building, e.g. if the Hough transform finds 3 line segments there are 3 walls present.

Building wall appearance assumption:

We assume that every line segment in the output of the Hough transform represent (a part of) the upper side of a specific wall of the building.

If a line segment is assumed to represent a single wall then the projection to that wall should have a large overlap with this wall. To be more precise, let's define l in \mathbb{R}^2 as a line segment that is generated by the Hough transform. If we project l to the plane spanned by a wall W we get a line l_{proj_W} in \mathbb{R}^3 . If we assume l to come from contour of wall W , then l_{proj_W} should have a large intersection with this wall W (as we overestimated the height of the building). We call this the line-wall overlap value, lwo . A mathematical

definition of lwo follows. Note that the projection of l with the other walls should have a small lwo value.

Largest line-wall overlap assumption:

A line segment describes the contour of the wall where its projection has the largest overlap with.

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. First, different types of overlap are explained. Secondly the algorithm determines the *overlap type*, then the overlap amount is determined and finally the amount of overlap is normalized.

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

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 3 represents the types of overlap with the corresponding number of points that pass the *in polygon* test and their possible line-wall overlap value.

Table 3: 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

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,

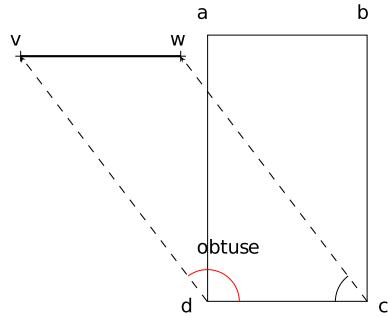


Figure 10a

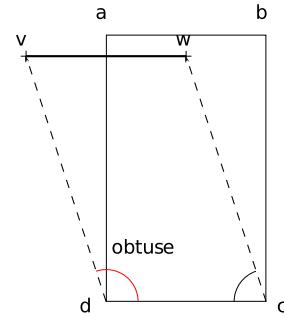


Figure 10b

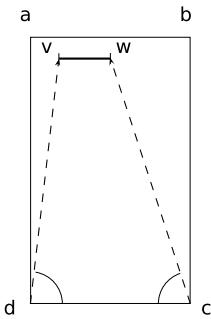


Figure 10c

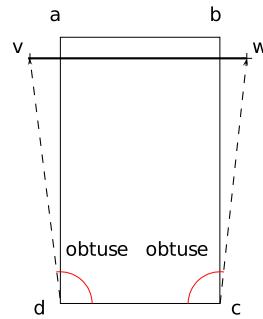


Figure 10d

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 calculate the amount of line-wall overlap, the line segment is cropped to the part that overlaps the wall and the length is measured.

The cropped line has two coordinates, first of course the point that passed the *in polygon* test and secondly the intersection of the line segment with one of the vertical wall sides (*da* or *cb* from Figure 10b).

We assume the walls to be of infinite height, therefore the partial overlapping line segment always intersects one of the vertical wall sides.

To determine which of the two vertical wall sides is crossed, we determine on which side the point that doesn't lie in the polygon (v) is on. This is done 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 $\angle d$ or $\angle c$ is obtuse, in this case $\angle d$ is obtuse. (Note that this holds because the walls are orthogonal to the basis which we assumed in the *Gravity aligned walls assumption*

To be more precise:

- 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. The advantage of this method is that it's simple and has low computational costs.

Line-wall overlap calculation

The amount of line-wall overlap is calculated by cutting off 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 with no points in the polygon, like in Figure 10a, the whole line segment lies outside the wall: an overlap value of zero is returned.

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 the same as line dc , but it's easy to see that this is not the case when

vw is not parallel to dc .

Line-wall overlap normalization

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

$$\alpha_l = \frac{lwo}{|l|} \quad (1)$$

Where α_l is the normalized line-wall overlap, lwo is the unnormalized amount of line-wall overlap, and $(|l|)$ is the total length of the line segment (uncut). 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, both amounts of overlap and -missing overlap are taken into account.

The maximum of the normalized line-wall overlap is used to associate a line segment with its most likely wall. To summarize, the overlap type is defined by calculating the numbers of in polygon points and evaluating two dotproducts. Next the line segment is cut off depending on the overlap type and the line is normalized.

Now we have determined the normalized line-wall overlap, we use this 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.

5.4.5 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 which will eventually be used to update the 3D model in the next section.

Now all line segments are associated with a certain wall, we re-project the line segment from the different views on their associated wall. The re-projection is done by intersecting both endpoints of the line segment to the plane that is spanned by the associated wall.

Next the 3D intersection points are collected and averaged, this gives us an average of the midpoints of the projected line segments. We do this for every wall separately, returning the average height of the line segments. These

averages are then used as the new heights of the walls of the building. Note that this is only permitted in presence of the *flat roof assumption*.

The new individual heights are used in the 3D model by adjusting the location 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 permitted by the *Gravity aligned walls assumption*.

5.5 Results

Let's return to the output of the skyline detector in Figure ??.

Figure 13 shows the top 3 longest Houghlines, the endpoints are denoted



Figure 13: Three best ranked lines of the Hough transform on the skyline detector

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. The left and middle line segment are connected. The right line segment covers the wall until the tree occludes. Figure 14 3 displays the line segments (originated from different views) pro-

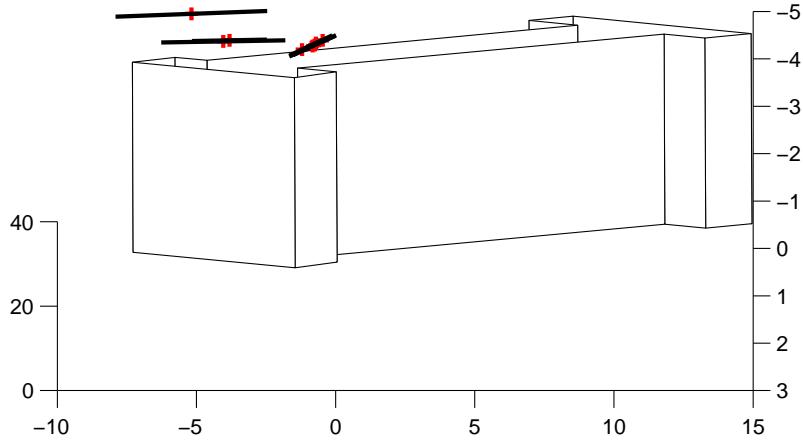


Figure 14: Houghlines projected on the most likely wall

jected on to their associated walls. For a clear view we've only selected the lines that were associated with two specific walls of the building. The red cross in the middle of the line represents the average of its endpoints. Figure 15 displays the updated 3D model. The corner points of the walls are adjusted according the calculated wall heights. The green plane displays the augmented wall. The left and middle wall are extended and the right wall is shortened.

5.6 Discussion

As can be seen in Figure 13, the top three Houghlines correspond the the three most prominent building walls which is a good result. What also can be seen is that the left line segment doesn't cover the whole building wall. This is caused by the use of strict parameters in the Hough transform (like a small line thickness parameter). If some ascending skyline pixels fall just outside the Houghlines, a gap is created and the line segment is cut down at that point. This is however not a big problem because the lines are long

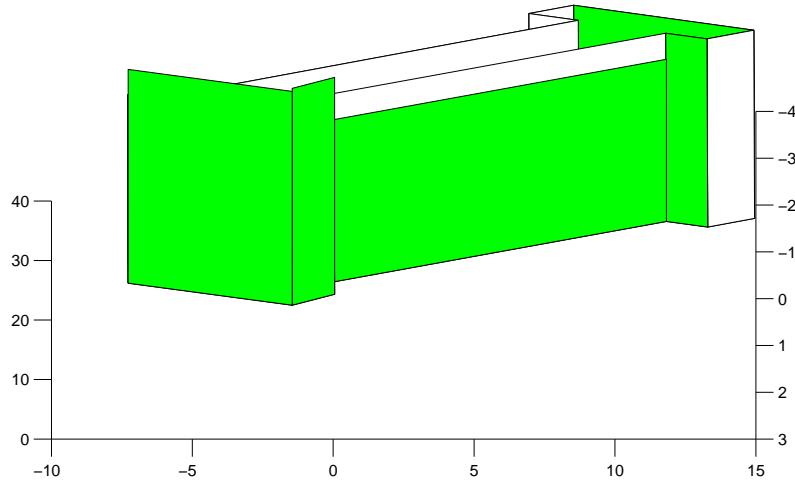


Figure 15: Improved 3D model

enough to produce a good wall height estimate. Furthermore there are 5 other lines (originated from different views) that support the estimate for this wall.

In Figure 14 the lines at the left side differ to much in height, at the time of writing this looks like a projection error that will be fixed. I think it is best to add more discussion when the other datasets and results are ready

5.7 Conclusion

To conclude, we showed that a Houghline transform is a useful method to detect outliers and find prominent structure in the contour of a building with a flat roof. We introduced a method to pair up line segments 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 significant improvement of a 3D model based on only a few calibrated 2D images.

5.8 Future work

Sometimes two line segments appear on 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. This could be achieved with an iterative Hough transform where the *Fill-Gap* parameter is increased in each iteration. E.g. for the right wall of the building in Figure *this figure will present at the next update* two iterations would be enough, the *FillGap* parameter needs to be at least as big as the occluding tree in the second iteration.

In this thesis little is discussed about the computational costs. This is because the computations are done efficiently (e.g. using matrix multiplications in Matlab) and off line, making the calculation process accomplishable in reasonable time. 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 needs to be calculated by averaging the line segments endpoints, this middlepoint is used in the *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 will be inaccurate, resulting in more false negatives: a linesegment that overlaps the wall with only 1/3 could be of use in the height estimation but instead it is discarded. What can be concluded is that there is a trade off in the accurateness of the height estimation and the computational costs.

5.8.1 Alternative roofs

To make the algorithm more generic, the flat roof assumption could be stretched or even discarded. We'll now consider other roof types and discuss what adaptations the system should require to handle these. In Figure 16, 6 different roof shapes are displayed.

Below: Rooflines take one of six basic shapes.

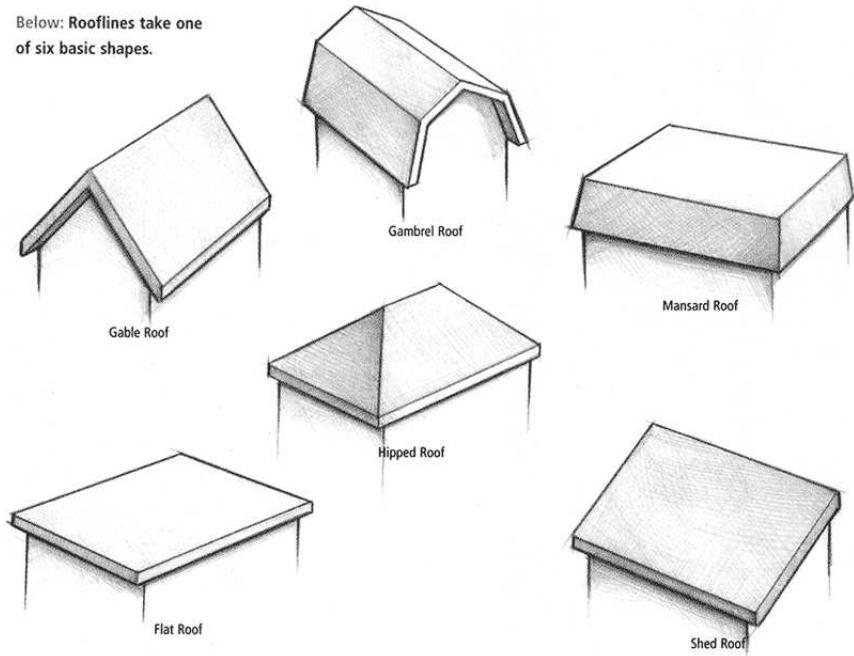


Figure 16: Different types of roofs

Consider the *Gable Roof*, it is a roof consisting of two planes which are not parallel with the facade of the building. This makes the problem of extracting the 3D model more complex, but not infeasible.

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 building. 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 can detect which roof type we are dealing with, what the wall heights are, and finally generate an entire 3D model.

Some ideas about this are now proposed:

- Use an object detector to detect doors, windows and dormers so the number of floors, the location of the wall-roof separation and the exclusion of some roof types (e.g. a dormer is never located on a flat roof) could be determined.

- Use the Hough transform to search for horizontal lines to detect the wall-roof separation, and use the ground plane and the top roof line to guide the search. Some buildings have a gutter, because of this the number of horizontal lines on the wall-roof separation will be larger which could be of great use.
- Use geographic information (a database of roof types) with GPS location to classify the roof type.
- The skyline detector detects the building height, if we could use pre-defined information about the ratio between the wall height and total height of the building, the wall heights could be estimated.

Assuming we determined the roof type, the building height and wall heights, the 3D model could easily be generated. For the *Gable* roof for example this will involve connecting two surfaces from the upper side of the walls with the high roof line (returned by the skyline detector). For the other roof types, the building height and wall height together with a template structure of the roof could be used to generate the 3D model.

6 My old method of line-wall association

We take the line segments endpoints and project it onto the building walls. The wall with the shortest distance to the camera center will be assigned to the line segment. Points that lie outside the polygon are punished.

And to update the specific wall we first need to know with a high probability of being correct which wall the line segment presents. But this method introduces a problem: some of the line segments have endpoints that lie at the corner of the building. These line segments could easily be associated with the neighboring wall. Because the 3D model is a rough estimate this could lead to bad results. In the corner case it is not clear to which wall the line segment belongs because both endpoints do not agree on the same wall. To solve this problem some heuristic methods are developed and tested. The following heuristic is both simple and effective. The heuristic uses the importance of the middle point of the line segment. This middle point has a low chance of being on a building corner and the on average biggest chance of being on the wall we are looking for. Therefor we discard the endpoints

and use the middle point the endpoints to determine the right wall. This middle point is intersected with all planes spanned by the walls. The line segment is stored to the wall with the shortest distance. The output of this part of the algorithm is for every wall a bunch of associated line segments originated from different views.

in section Results was this text:

The left and middle line segment of Figure 13 are a good example of the corner problem. Both endpoints that lie on the corner could easily be associated with the wrong wall (even if the rough 3D model is very accurate). Fortunately we use the middle point of the line segment to determine the correct wall. This works well as its 100% accurate (for this dataset).

TODO: Zorgen dat paragraphs ook 3.5.3.1 nummering krijgt

7 Window detection

7.1 Introduction

Semantic interpretation of urban scenes is used in a wide range of applications. We deal with an important aspect of semantic urban scene interpretation, Window detection.

This chapter is organised as follows, we start with a variety applications that use semantic interpretation of urban scenes. Then 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 this method is used for our second method that assumes orthogonal and aligned windows. Finally we show and discuss results.

7.2 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 detected (doors and) windows are mapped to the model to increase the level of detail. Some other applications can automatically extract a CAD-like model of the building surface.

Historical buildings documentation and deformation analysis

In some field of research, Historical buildings are documented. The complex structures that are contained in the facades are recorded and reconstructed. Window detection plays a central role in this. Another field of research is the analysis of building deformation in areas containing old buildings. 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 who require 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.



Figure 17: Simulation environment

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 using large image datasets where 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 doesn't 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 window detection plays an important role in the interpretation of urban scenes and is applied in a wide range of domains.

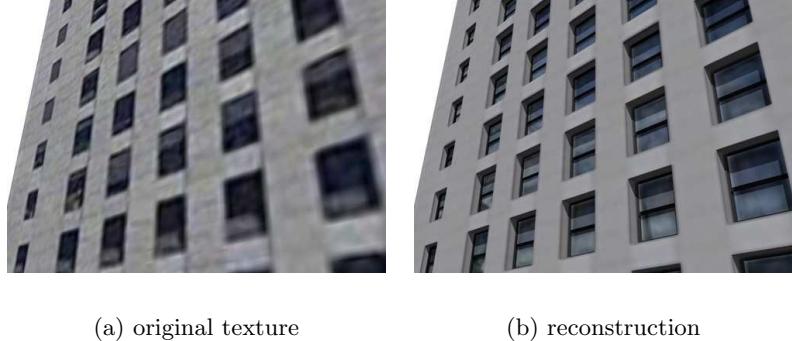
TODO: in state of art een lijn creeren, opbouwen, in context plaatsen, theoretisch afwimpen

7.3 State of art window detection

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 the art window detection that has a big overlap with our approach in detail.

7.3.1 Alternative approaches on window detection

Muller et all [8] detect regularity and symmetry in the building. The symmetry is detected in the vertical (floors) and horizontal (window rows) direction. 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 constraints the variety



(a) original texture

(b) reconstruction

Figure 18: Results of Muller et all

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.

Using a thermal camera, Sirmacek [12] detects heat leakage on building walls as an indicator for doors or windows. Windows are detected with L-shaped features as 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, i.e. determine if it is a window.

Ali et all [1] 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 use window detection to determine the region of the facade.

Although this method is used a lot in computer vision, 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. As the cascader uses a fixed swiping window it is sensitive to scale: all window sizes must be equal and a size range must be given to the system. Furthermore the system is always overfitted to the learning data, making it hard to generalise (detect windows that are not included in the dataset). A more general descriptor of the window that is size invariant is desirable.

TODO: research question

To investigate this, we developed two methods on which one method doesn't require repeating windows nor aligned windows. One of the main targets of our research is a low requirements on the input data. First our system doesn't need a large annotated dataset. Furthermore we took the images with a mobile phone and decided to extract the windows from the image space only, this makes us independent of additional expensive data like heat or laser range images. It 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's explain a method of that kind and discuss other approaches that are more similar to our approach.

7.3.2 Similar approaches

Pu and Vosselman [10] 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 the results were far from accurate.

The work of Pu and Vosselman [10] provides a useful practical application of window detection and it amplifies the need for a robust window detection technique that is independent of laser range data.

Recky et all [11] developed a window detector that is build on the primary work of Lee and Nevatia [7] (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 to

establish the vertical division of the windows.

A function is developed that counts the amount of projected edges on each



Figure 19: Projection profiles of Lee and Nevatias work

location, this is the *Projection profile*, see Figure 19. A threshold is applied on the projection profile to indicate the window boundaries that is 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.

As in the work of Recky et all [11] Lee et all [7] 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 window independently. Although this comes with accurate results, the iterative refinement is a computational expensive procedure. As we want to run our system in realtime this method is not suitable for our application.

7.3.3 Our work in context

The state of the art window alignment procedure in [11] and [7] is very robust. Therefore we decided to use this method as a basis and improved the alignment algorithm, furthermore we build a different window classification method.

Our improvement on the alignment procedure is as follows. In the previous work [11] and [7] they use a single projection profile for each direction. 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 build two alternative window classification procedure which are based on a higher level of 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 perform in real-time.

7.4 Method I: Connected corner approach

7.4.1 Situation and assumptions

A window consists of a complex structure involving a lot of connected horizontal and vertical lines, we use this property to detect the windows. We introduce the concept *connected corner*, this is a corner that is connected to a horizontal and vertical line. The search for these connected corners is based on edge information. 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.

7.4.2 Edge detection and Houghline extraction

Edge detection is done as it is described in section 3.1. From the edge images two groups of Houghlines 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 these ranges because 1) the user hardly ever holds the camera exactly orthogonal. 2) we work with unrectified facades, meaning we deal with perspective distortion. To be more concrete, if the user takes a photo (Figure 20 with a certain Yaw $\bar{\theta}$, the horizontal lines become skew. The range of the vertical group is smaller than the horizontal group as the user often takes photo with a low pitch value and a high yaw.

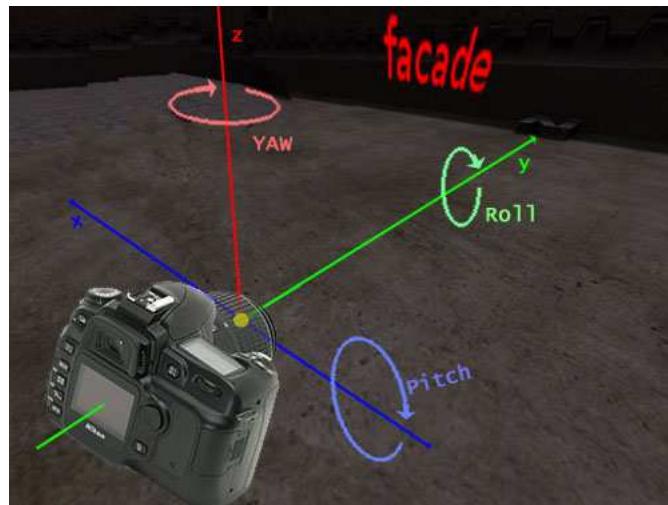


Figure 20: Pitch roll and yaw of the camera

The results of the edge detection and the Hough transform of two images can be seen in Figure 22 and 23.

TODO: 2

maak analogie met L en H patronen hoe we die herkennen in de hersenen

7.4.3 Connected corners extraction

As windows contain complex structures the amount of horizontal and vertical Houghlines is large at these locations. A horizontal and a vertical line are often connected in a corner of a window. In this approach we pair up these horizontal and vertical lines to determine *connected corners* that indicate a window.

Often a connected corner contains a small gap or an extension which we tolerate, these cases are illustrated in Figure 21 in the top row. A horizontal

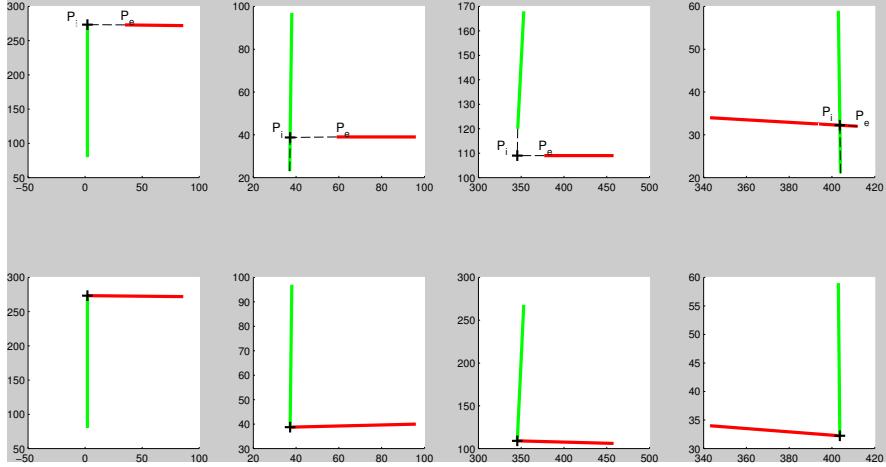


Figure 21: First row: different type of connected corner candidates. Second row: the result the clean connected corner

gap, a vertical and horizontal gap and a vertical elongation. The cleaned up 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 21. Next, D is compared to a *maximum intersection distance* threshold $midT$. And if $D \leq midT$, the intersection is close enough to form a connected corner.

After two Houghlines 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 21. In Figure 21(I) the horizontal line is extended. Figure 21(II) shows that the vertical line is trimmed. In Figure 21(III) both lines are extended. At last, Figure 21(IV) shows how both lines are trimmed.

7.4.4 Window area extraction

To retrieve the actual windows, each connected corner is mirrored along its diagonal. The connected corner now contains four sides which form a quadrangle window area. All quadrangles are filled and displayed in Figure 25, this result is discussed in section 7.4.5.

7.4.5 Results

· TODO: 2

more datasets..

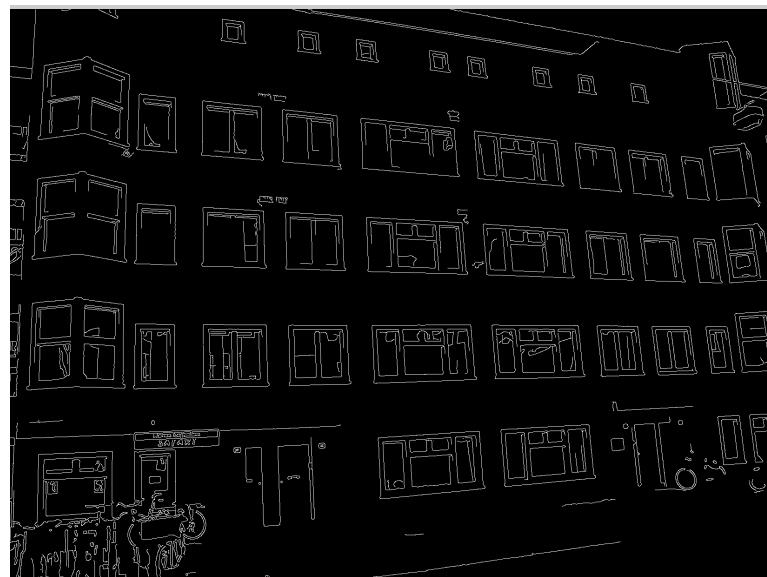


Figure 22: Edge detection

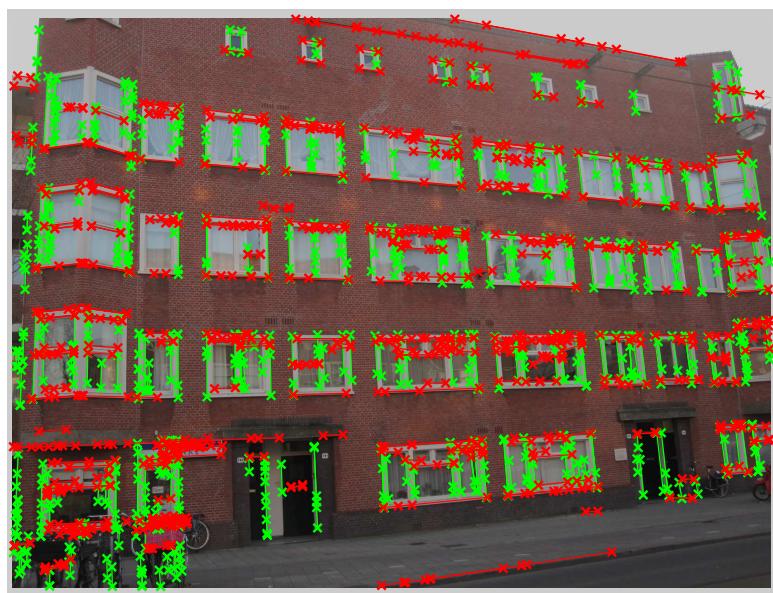


Figure 23: Result of θ constrained Hough transform



Figure 24: Found connected corners



Figure 25: Window regions

Figure 25 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 than the minimum window width.

7.4.6 Conclusion

As can be concluded from the 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. Furthermore as no image rectification is required, the system has a low input requirement as it is independent of calibrated input data and 3d information about the building.

7.4.7 Future research

We only developed L-shaped connected corners, it would be nice to connect more parts of the window to form U shaped connected corners or even complete rectangles. The later is difficult because the edges are often incomplete due to for example occlusion.

Furthermore the next step in this study would be to group the connected corners to windows and sub windows. The big window that contains sub windows could be found by calculating the convex hull of the red areas in Figure 25. The sub windows could be found using a clustering algorithm that groups the connected corners to a window. For this method it would be useful to assume the window size as this correlates directly to the inter-cluster distance. It would also be nice to incorporate not only the center of the connected corner as a parameter of the cluster space but also the length and position of the connected corners' horizontal and vertical line parts. The inter cluster distance and the number of grouped connected corner could form a good source for the certainty of the sub window.

7.5 Facade rectification

7.5.1 Introduction

In order to apply our second method of window detection (7.7), we need the windows on the facade to be orthogonal and aligned. Therefore we rectify the facade, which can be achieved in a manual or automatic way.

The simple 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 isn't very accurate as it doesn't take the width and height ration of the facade into account. It only uses four point to point correspondences and doesn't take the camera lens distortion into account. Another downside is that it requires (manual) annotation of the corner points. A more accurate and fully automatic method is desired .

The second method involves the extraction of a 3D plane of the facade. This method is more complex and gives more accurate results. It involves a comprehensive process and a large amount of research is done in this area. Given the related work, and our focus on the annotational part of facade interpretation, we used existing software to apply the window rectification. I. Estebans *FIT3D toolbox* [6] comes with an addon which extracts a 3D model from a series of frames. One of the planes of the 3D model was used to rectify the facade, making it ready for robust window detection.

Using this addon of the *FIT3D toolbox* [6] we calculated 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. which is used to rectify the facade by a projection process. We now explain the steps in more detail.

7.5.2 3D plane extraction

We started by taking a series of 7 consecutive images (steady zoom, lighting, etc. parameters) of a scene. The images don't need to be chronological but need to have sufficient overlap.

We calculated the motion of the camera between the frames in a few steps First we extract about 25k SIFT features of each frame. Then we use SIFT descriptors to describe and match the features within the consecutive frames. Not all features will overlap or match in the frames therefor RANSAC is used to robustly remove the outliers. After this an *8-point algorithm* together

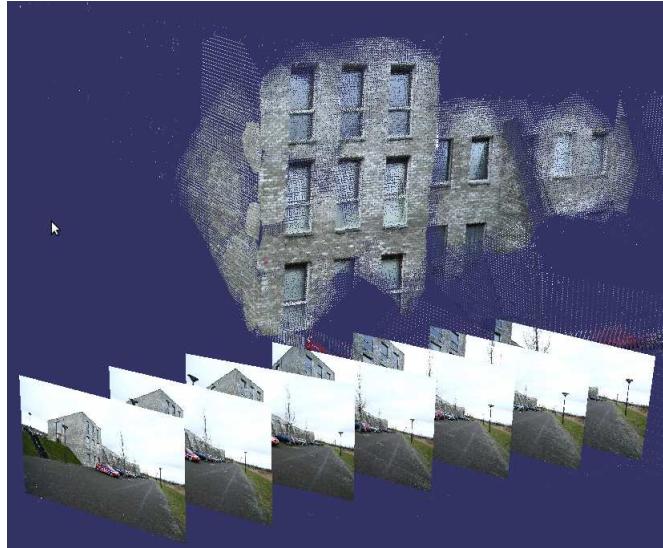


Figure 26: Example series of 7 consecutive frames, (dataset: FIT3D toolbox[6])

with a voting mechanism is used to extract the camera motion. For details please read [6].

The frames were matched one by one which returns an estimation of the camera motion that is not accurate enough. Therefore a 3-frame match is done which gives more accurate results. Unfortunately this result comes with a certain amount of reprojection error, this error is minimized using a numerical iterative method called *bundle adjustment*. The final result is a very accurate estimation of the camera motion.

The next step is to use this camera motion to obtain a set of 3D points (corresponding to the matching image features). This is achieved using *linear triangulation* method.

Next a RANSAC based plane fitter is used to accurately fit a plane through the 3D points. *todo example figure*

7.5.3 Efficient Projecting

Now we extracted the 3D plane of the facade, the next challenge is to use this in order to rectify the facade.

It would be straight forward to rectify the full image. However this is com-

putational 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 Houghlines we project only the coordinates of the endpoints of the found Houghlines. (This is allowed because the projective transformation we apply preserves the straightness of the lines. Note that this means we apply the edge detection and Houghline extraction on the unrectified image.)

If h is the number of Houghlines, the number of projections is $2h$. When we rectify the full image the number of projection is $w \times h$, 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 we extracted in the same way as we explained in chapter ???. To wrap up, we send rays from the camera center through the houghline 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 Houghlines) to a frontal view. Instead of transforming the facade we rotate and translate the camera. This means the viewing direction (z-axis) of the camera needs to be equivalent to the normal of the facade plane. Lets denote the unit vector spanning the original viewing direction as z and the unit vector spanning the desired viewing direction/normal of the facade z' . We calculated a rotation matrix from the axis-angle representation.

This presentation uses a unit vector u indicating the direction of a directed axis, and an angle describing the magnitude of the rotation about this axis, see 3.9. This axis is orthogonal to z and z' and can therefore be calculated by $u = \text{cross}(z, z')$ where cross defines the cross product of two vectors. The magnitude of the rotation α is equivalent to the angle z between z' given u . E.g. if the images are taken from a(n almost) frontal view (the facade is almost rectified) α is low. α and u are used to create a rotation matrix R which is applied to the 3D point cloud. The result is a set of rectified 3D points that are grouped to their Houghlines.

7.5.4 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 Houghlines in context.



Figure 27: Dataset: Anne1, Original, (unrectified) image

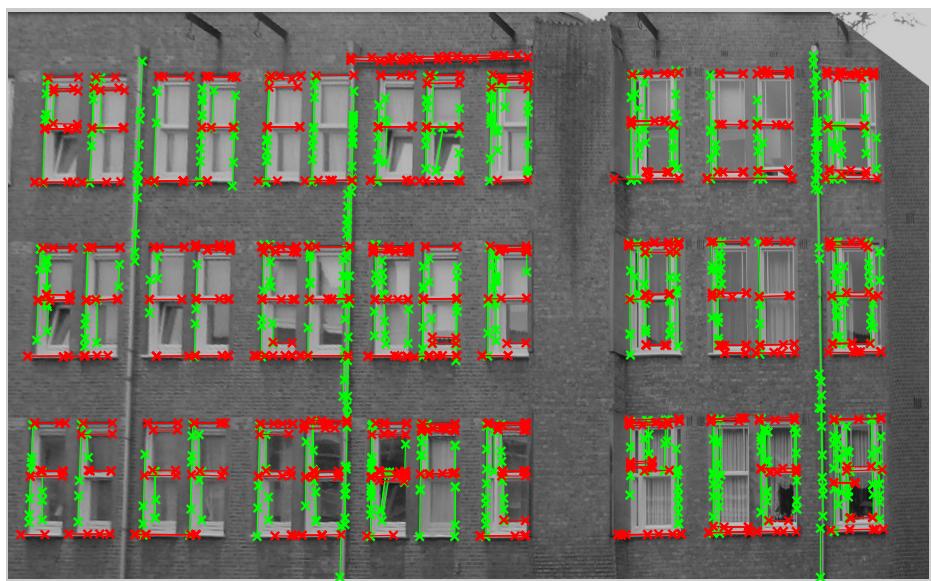


Figure 28: Dataset: Anne1, Projected Houghlines on rectified image



Figure 29: Dataset: Dirk, Original, (unrectified) image,

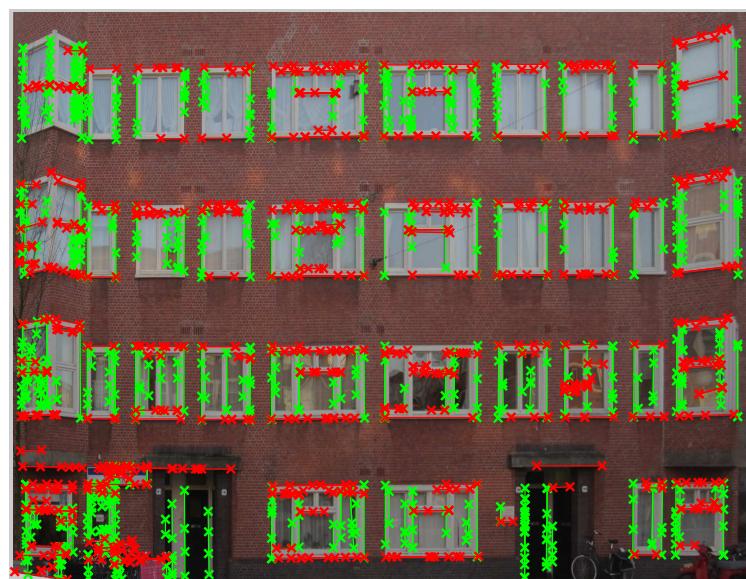


Figure 30: Dataset: Dirk, Projected Houghlines on rectified image

TODO: Met iemand overleggen wat betere volgorde is

7.6 Datasets

To avoid overfitting we used multiple datasets that have a large property variety. 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 a original resolution of 3072x2304 px and are downscaled (using trilinear interpolation) to 1280x1024 px.

7.6.1 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 windows on the right side of the image is 698 px, at the left side this is 320 px: a resolution of more than 2 times smaller making 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 31: Dataset: Anne1, Rectified image

7.6.2 Anne2

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



Figure 32: Dataset: Anne2

7.6.3 Dirk

The Dirk dataset represents an everyday scene as it contains light spots on the facade, bicycles, an an occluding tree. It contains zero rectification error but the windows are partially aligned. The windows are very closely located (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 relative low, implicating little or no occlusion artefacts.



Figure 33: Dataset: Dirk

TODO: 2

tabel van datasets en handicaps!

7.7 Method II: Histogram based approach

7.7.1 Introduction

From 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 this 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 Houghlines'. We use this alignment to divide the image in window or 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.

7.7.2 Extracting the window alignment

Regular window alignment

TODO: 2

Schematje We introduce the concept alignment line. We define this as a horizontal or vertical line that aligns multiple windows. In Figure 34 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 this alignment lines? We make use of the fact that among a horizontal alignment line a lot of horizontal Houghlines are present,

see Figure 28. For the vertical alignment lines the number of vertical Hough-lines is high, see green lines in Figure 28.

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 Houghlines.

Next we calculate two histograms $H(\text{horizontal})$ and $V(\text{vertical})$, containing respectively w and h bins where $w \times h$ is the dimension of the image. The histograms are presented as small yellow bars in Figure 34.

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 than the desired number of alignment lines. Therefore we smooth the values using a moving average filter. The result, red lines in Figure 34, 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 make the threshold invariant to the values, we set the threshold to $0.5 \cdot \max \text{Peak}$. (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 contained 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 34.

Alternative window alignment

As you can see in Figure 35 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* Houghlines projected on the X axis, X_h , Figure 36. 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 36.

$$D = \text{abs}(X'_h)$$

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 target is to have as few as possible false positives while detecting all alignment locations.

If a window alignment position is found by both methods, often the peaks are located very close to each other, see Figure 37. 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 windowframe part). Yet, 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 $0.3 * \text{max peak}$ to $0.5 * \text{max peak}$). 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 43.

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.

TODO: 2

peak merging result misses for 1 dataset?

TODO: 2

peak merging result plaatje met en zonder merging onder elkaar

Results

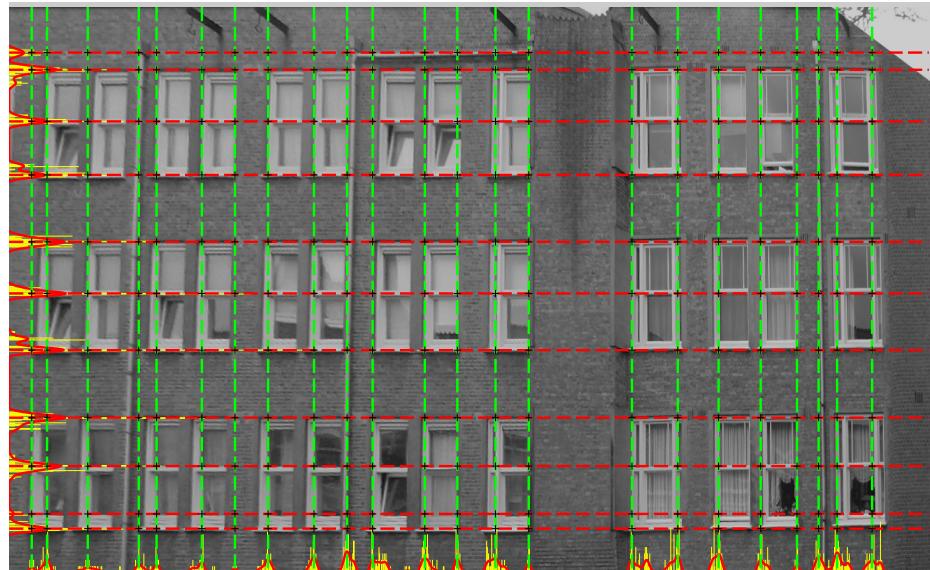


Figure 34: Dataset: Anne1, Regular window alignment (parallel projection): Based on a smoothed histogram (red line) that displays the amount of overlapping Houghlines, for the column division the horizontal houghlines are counted (at each y position), for the row division the vertical houghlines are counted (at each x position)

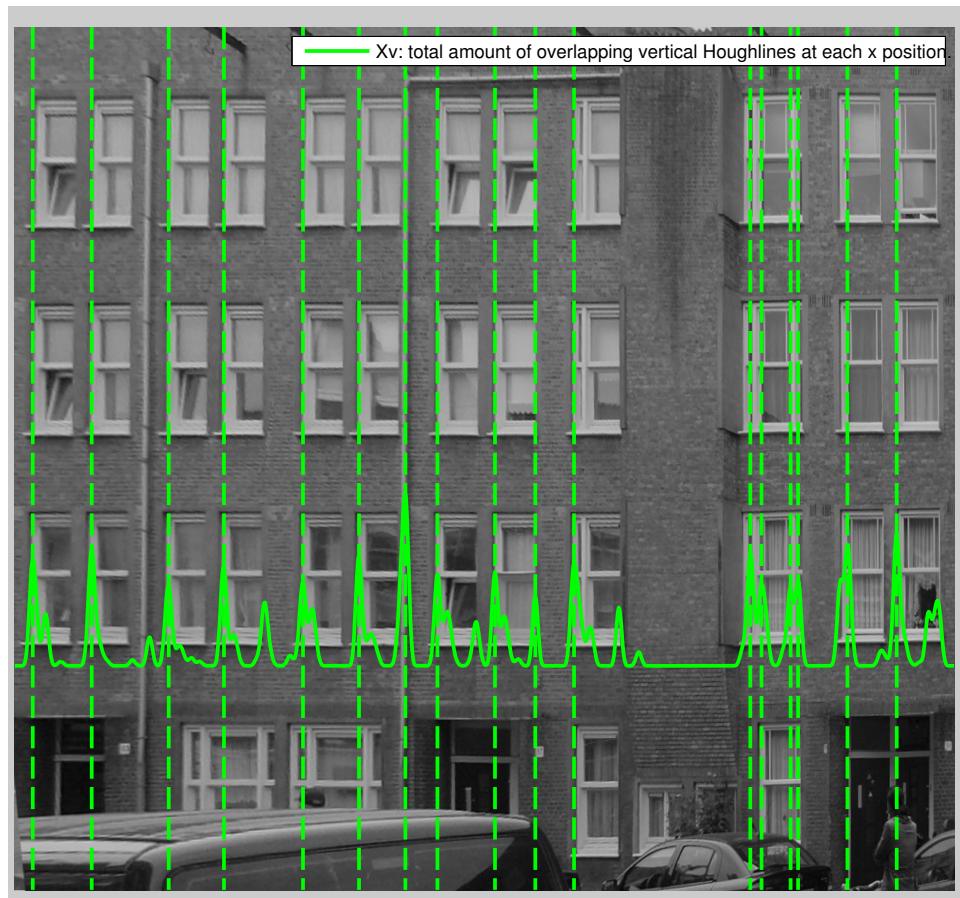


Figure 35: Dataset: Anne2, Regular window alignment: Based on a histogram that displays amount of overlapping vertical Houghlines at each x position



Figure 36: Dataset: Anne2, 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 (Note that this is the orthogonal oposite of the regular window alignment method). Peaks (that represent a big decrease or increase of the histogram function) are used for the alignment.

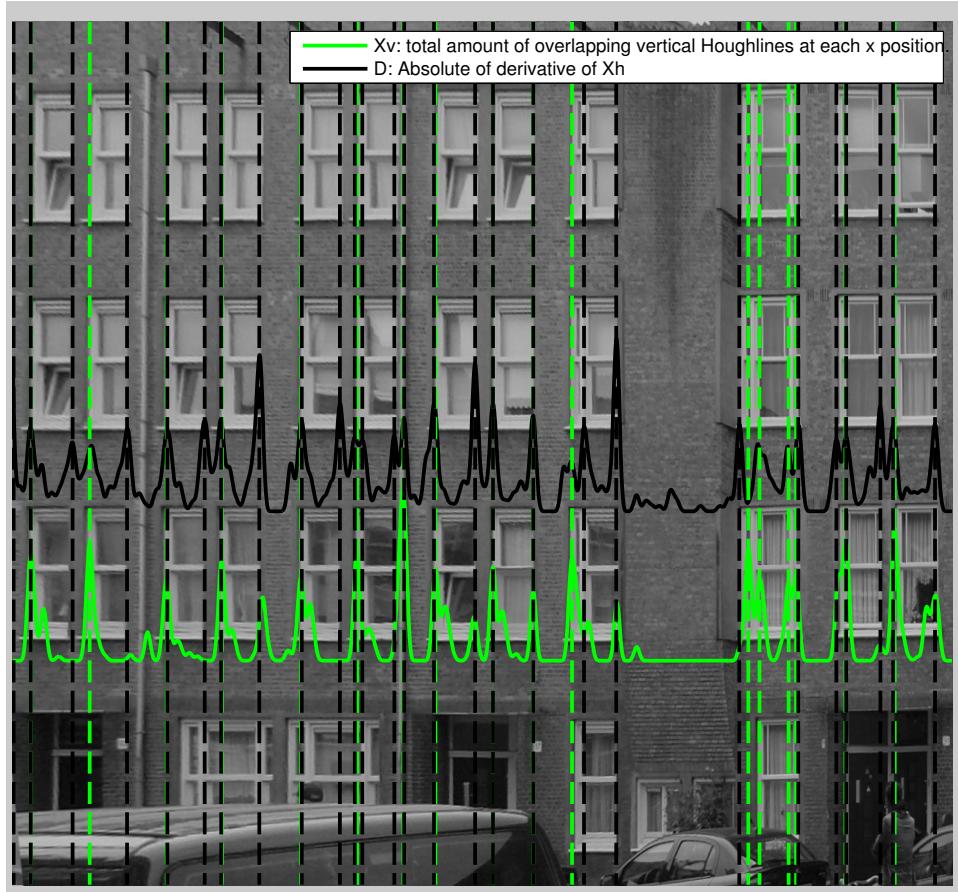


Figure 37: Dataset: Anne2, Regular+Alternative window alignment combined

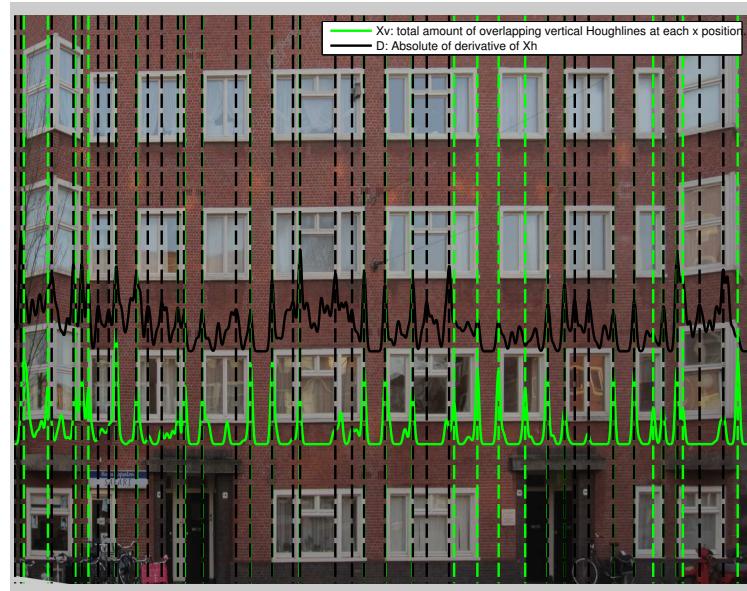


Figure 38: Dataset: Dirk, Regular+Alternative window alignment combined

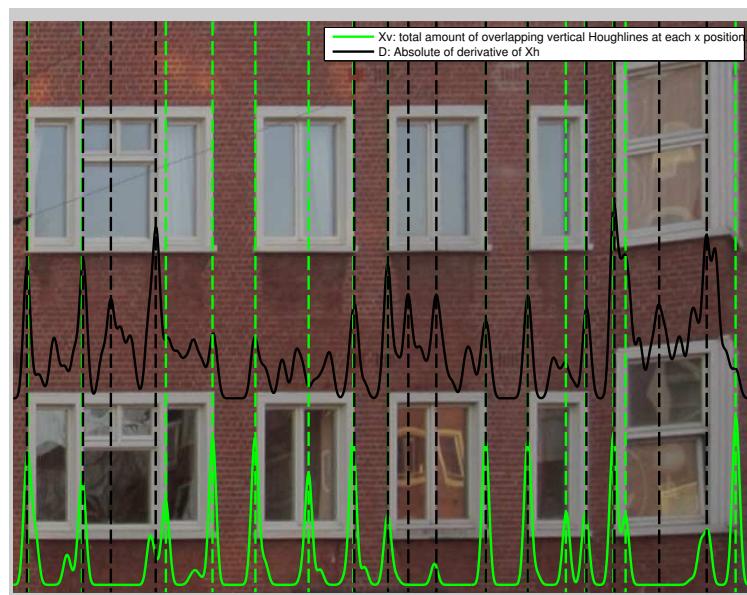


Figure 39: Dataset: Dirk, Regular+Alternative window alignment combined, zoomed

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 38, too many alignment lines 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 49) cause a large amount of found Hough lines which implies a (false) alignment line. Although the causes are clear, we don't have to worry about this additional window alignment lines that lie inside a window area as 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 36, 35) that in both datasets neither the regular window alignment method nor the improved window alignment 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 method succeeds in detecting the window alignment.

If we take a look at the regular window alignment in Figure 35 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 27) is not a frontal image. This makes building wall extensions (middle of Figure 27) or drainpipes occlude parts of the window. In this case the windows are countersunked into the wall making the building wall itself occluding the right window frame (Figure 35). The color of the reflection of the window is very similar to the bricks and because the window and wall 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 Houghlines will

be detected and (as the regular window alignment is based on the amount of vertical Houghlines) no window alignment will be found. This artefact can be studied in the edge image Figure 48: few or no edges are present, and at the low height of the peaks in Figure 35 at these positions.

However, the alternative window alignment does find a window alignment on this 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 Houghlines 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. Figure 28 every window has (as it has two vertical subwindows) 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 alternative window alignment method is outperformed by the regular window alignment . For example, in Figure 39, the left side of the second window column is not detected. This is because this window type has no horizontal divisions. Only the top and bottom of the windowframe produce an edge, therefore the derivative of the amount of Houghlines 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 divisions (for example the most left or most right window column)).

TODO: result1 drain pipe result niet erg ivm vlg stap

Conclusion

We can conclude that developing histogram functions of the amount of Houghlines is a strong approach towards determination of the window alignment as the results in this work and in previous work a 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 and because it uses a higher level interpretation of the histogram function.

We showed that window alignment becomes a challenging task if the windows are partially occluded. One of the main solutions we proposed is to use multiple window detection 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.

We can conclude that a method that fills the gap of the occluded alignment locations must rely on Houghlines that lie in the orthogonal direction of the occlusion. In our case the occluded vertical window parts are supported by a method that detects horizontal window parts.

Conclusion:Relative thresholding

Furthermore we discussed the implication of the use of different window types. Applying a threshold that is relative to a maximum peak doesn't work well on this window types that differ (in for example horizontal divisions) as window alignment lines are missing (Figure 39). This means that if we want to stick with the relative thresholding method, we have to assume a certain equality of the window types.

TODO: make diagram: edge- hough -window alignment - window classification - window grouping

TODO: diagram maken window alignment -*i*, peak merging -*i*, etc

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 39) 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 Houghlines. 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 $0.7 * \text{max peak}$. However

if there is a large variation is found (which means multiple window types are present), the threshold should be lowered to $0.3 * \text{max peak}$ to detect the hard window types (with few horizontal divisions). Another method would be to cluster the amount of Houghlines in $n+1$ values where n is the number of window types (the 1 is for the non-window area). Areas that transcent 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.

Future research:Peak grouping

We fused the result of the horizontal and vertical Houghlines 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 detect these values.

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 [7] applied window refinement. Although this comes with accurate results, the iterative refinement is a computational expensive procedure. It would be nice to have a dynamic system that is aware of this accuracy and computational time trade off. A system that only refines the results when the resources 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 align-

ment or refinement of the windows.

7.7.3 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 a 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 Houghlines, Figure 28 (green). For the blockcolumns the number of horizontal Houghlines (red) is high at window areas. We use this property to classify the blockrows/blockcolumns.

For each blockrow the overlap of all vertical Houghlines are summed up. (Remark that with this method we take both the length of the Houghlines and amount of Houghlines 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.

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

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 Houghlines pixels) which leaves us with C .

TODO: DIT PLAATJE eruit en mooie staafdiagrammen erin

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 40. Its easy to see that the high values, blockrow 4,5,7,8,10 and 11, correspond to the six window blockrows in Figure 42.

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 number of windows, window types etc., the threshold maybe classify

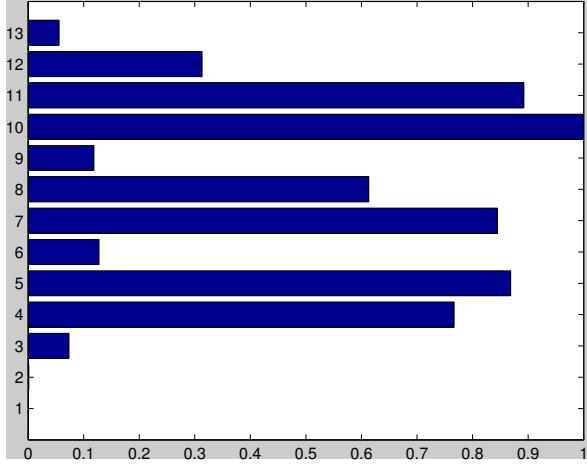


Figure 40: Classification values for window block rows representing the normalized vertical Houghline pixel count of (R)

insufficient in another scene. Hence working with the threshold wouldn't be robust.

Instead we use the fact that a blockrow is either filled with windows or not, hence there should always be two clusters. We use *k-means* clustering (with $k = 2$) as the classification procedure.

TODO: ref

This results in a set of Rows and Columns that are classified as window and 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 42 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 42.

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 knows 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 41, the whiter the area the higher the measure of certainty.

Results

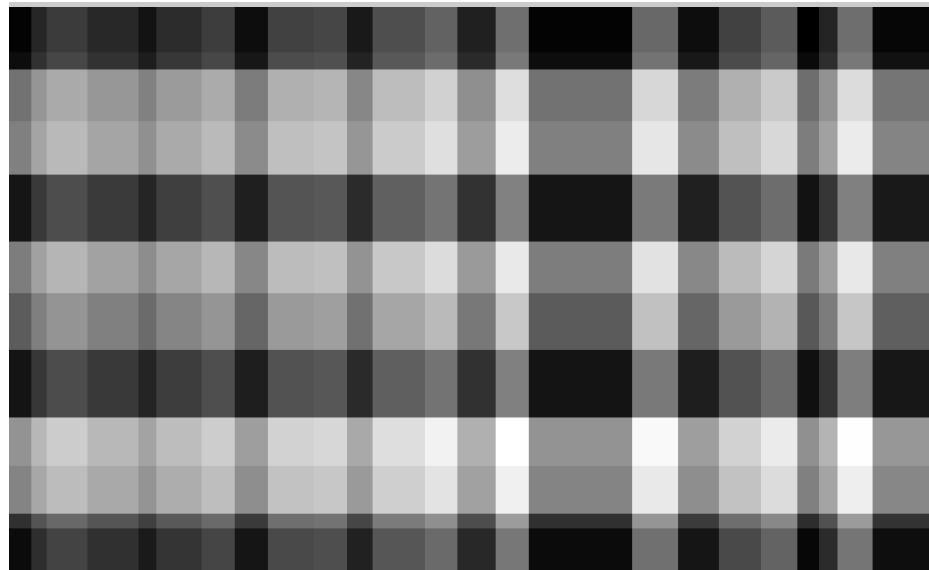


Figure 41: Dataset: Anne1, Basic window classification method:Measure of certainty, the whiter the area the higher the measure of certainty.

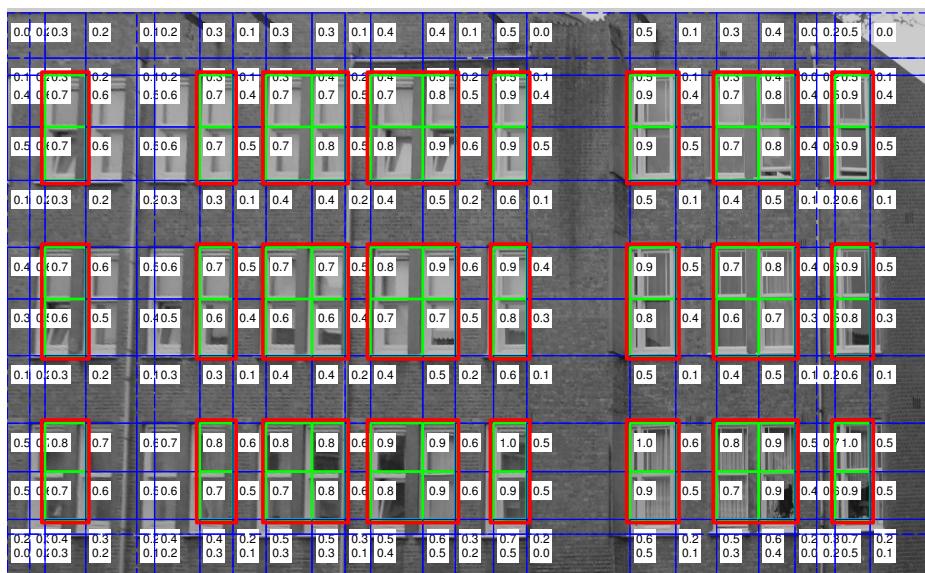


Figure 42: Dataset: Annel1, Basic window classification method: The extracted windows, red:the grouping

TODO: include results of binary classification

The bright rows and columns in Figure 41, 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 31).

TODO: include table of false positives etc, detection rate etc. etc. misch staat dit nog op github

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 50, 51, 52 and 53. This result can be explained as follows. The Dirk dataset has window types in the middle of that contain a horizontal subdivision while the others don't. The windows in the middle will cause k-means to drag the cluster center to a value that is too high making windows 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 clustercenters to $n+1$, where n is the number of window types (the 1 is for the non-window area).

Conclusion

We can conclude that this certainty based classification works quite good on this dataset. However, it requires a very well alignment of the windows which is a disadvantage. As it is based on the number of found Houghlines, 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 amplifies the need for a robust classification method (that is independent, (or at least less sensitive)) to window alignment errors.

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

TODO: introduce a little bit

TODO: say that anne2 dataset is more rectified

If we take a look at Figure 43 we see that X_h (the amount of horizontal Houghlines) 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 lots of horizontal lines are found at certain window positions (the window centers) and few are found at positions that are far away from these certain positions, which are the centers of non-window areas (that lie between windows). The shape type of X_h is used as a cue for our second window classifier.

This shape type of X_h is detected as follows: As in **improved window alignment**, the first step is to examine the derivative of X_h , blue line in Figure 43. We investigate the positions where $D = X'_h$ changes from sign, these are the peaks or 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 sake of representation only blockcolumns are presented, the method for the blockrows is the same, the projection profiles are projected to the Y-axis.

Results

TODO: make a diagram, histogram - alignment lines - ζ window classification etc.

TODO: discuss drain pipe, dat het goed is dat het resultaat mooi is ondanks drainpipe

Results: Dirk dataset

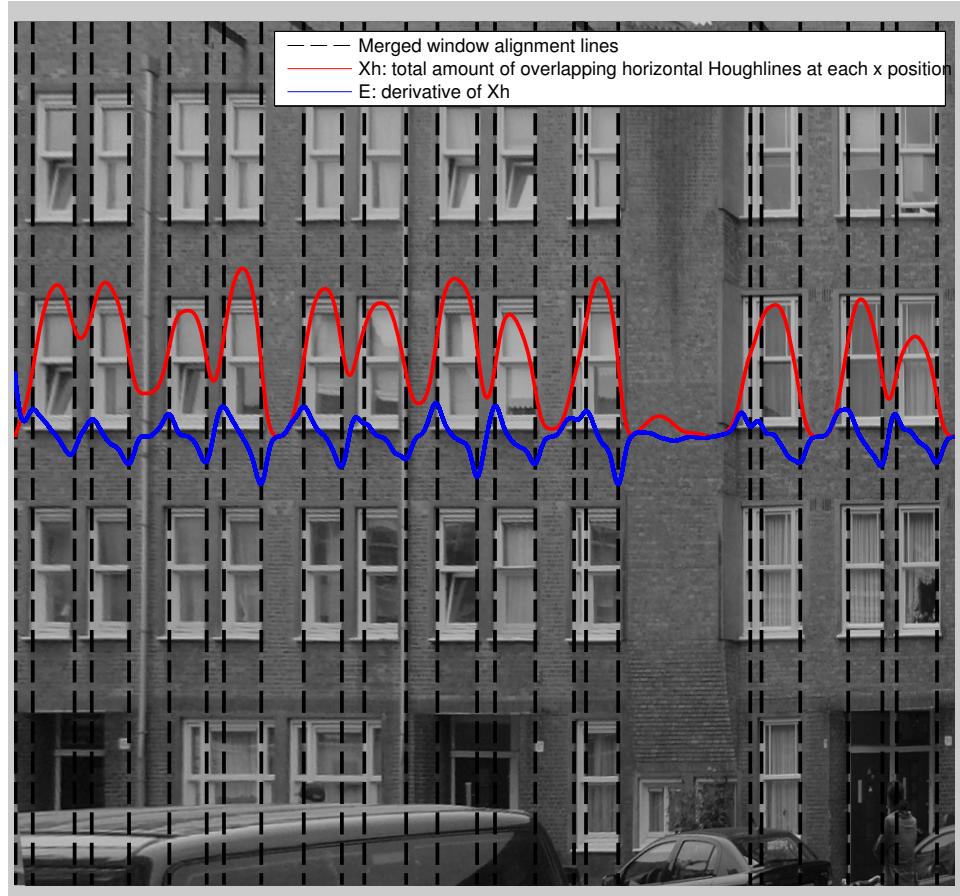


Figure 43: Dataset: Anne2, Improved window classification method: The red line shows concave shapes at window locations and convex shapes at nonwindow locations

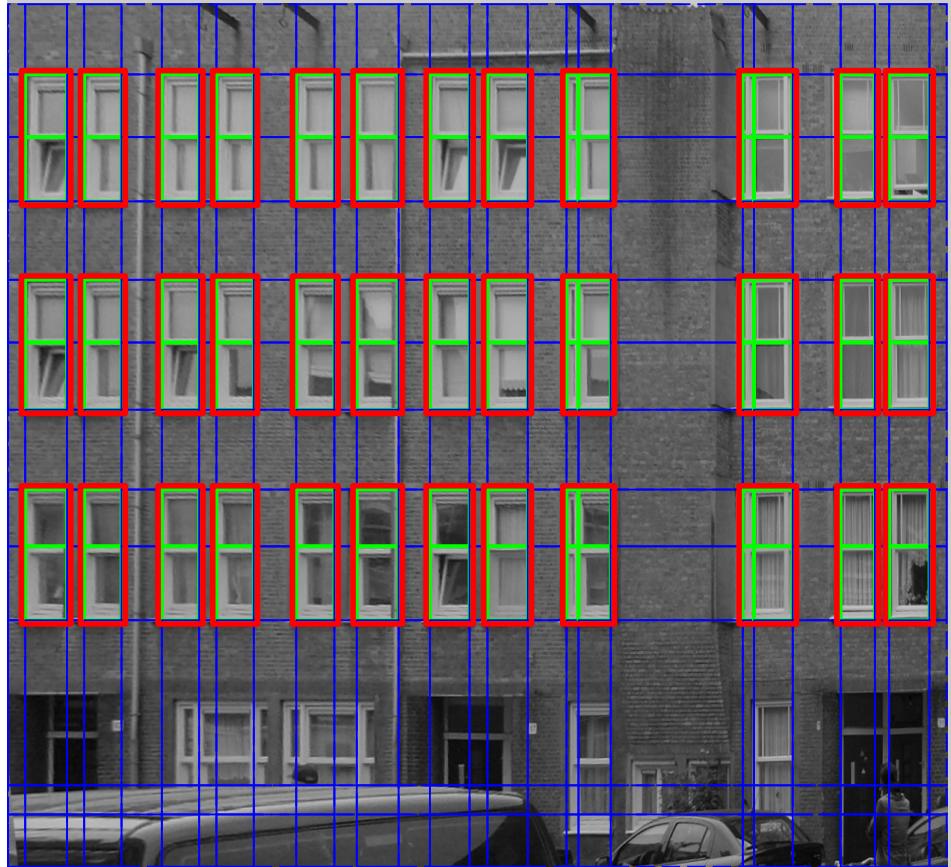


Figure 44: Dataset: Anne2, Improved window classification method: The extracted windows, red:the grouping

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 and a 100% true positive rate are very well. This is mainly due the high interpretation level of the Histogram function. We used the fact that the histogram function has a very consistent pattern, at every window area its shape is convex and every non-window its shape is concave.

Results: Anne2 dataset



Figure 45: 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

TODO: IMPROVED result on Anne1 dataset, investigate which combinations of dataset and methods i miss

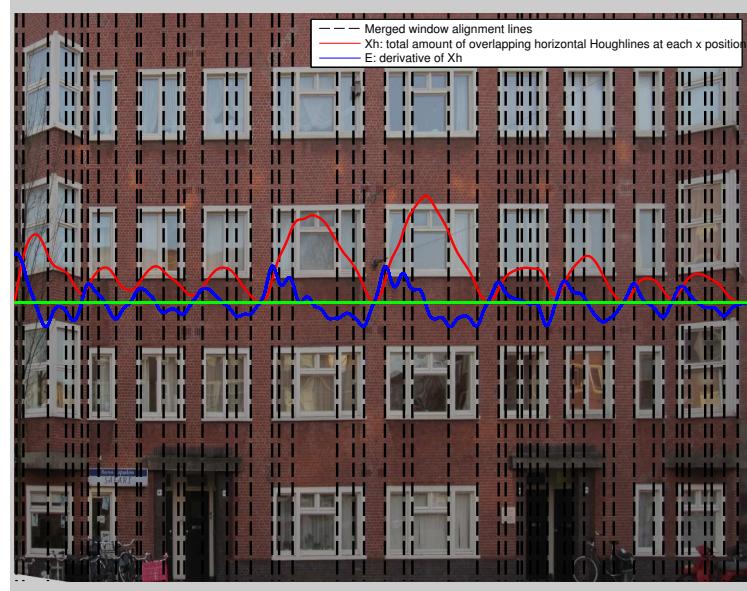


Figure 46: Dataset: Dirk, Improved window classification method: The red line shows concave shapes at window locations and convex shapes at nonwindow locations

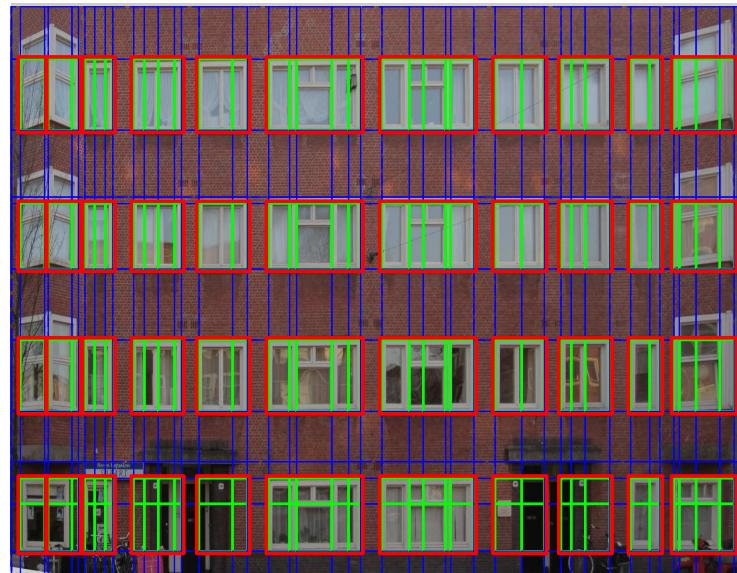


Figure 47: Dataset: Dirk, Improved window classification method: The extracted windows, red:the grouping

Table 4: Improved window classification results on Anne2 and Dirk dataset

Type	Anne2 dataset (Fig 44)	Dirk dataset (Fig 47)
Detection rate	100 %	100 %
True positive rate	100 %	87 %
False positive rate	0 %	11 %
False negative rate	0 %	9 %

Table 5: Grouping results on Anne2 and Dirk dataset

Type	Anne2 dataset (Fig 44)	Dirk dataset (Fig 47)
Grouped correct	100 %	90%
Grouped incorrect	0 %	10%

TODO: certainty based (white strokes) classification on anne2 dataset,
pas als skyline af is, img zijn wel al klaar

TODO: remark that we goedkeuren partieel resultaat van windows helemaal rechts en links dirk dataset

TODO: Interpretation of the results of the Anne1 dataset

TODO: compaire to other classification methods

TODO: (perform both classifications on both datasets (2x3 img), visual concatinate images vertically)

TODO: especially the horizontal division of the lowest row of windows is amazing, refer to

TODO: Meer over lof spreken

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 47 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 subwindows which were grouped to clear red big windows.

This grouping result is promising as 90% of the grouping went well. This 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 subwindows). The first two window rows in Figure 47 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. More on this in *Future research on window classification (7.7.4)*.

Conclusion

It is a typically artificial intelligent approach, to look for a high interpretation of the data. We can conclude that the results become very robust if one takes a high level interpretation of the Histogram function.

Furthermore we can conclude that:

- Redundant lines found by the window alignment module cause no

problems as long as they are located at a boundary of a window or in a window.

- The grouping module contained a small weakness, a fix is provided in the next section.
- The improved window classification outperformed the basic window classification.

Future research

TODO: put a opbouwende line in future work

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 downscale the images and/or add Gaussian noise to the data.

TODO: speak about video, or sequence of images, detection + tracking etc

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

Future work:Grouping error minimalisation

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 47. 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 of a area to be threatened as a non-window area. In this way small negatively classified areas cannot interupt the adjacent windows.

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.

TODO: compare classification methods and explain differences

7.8 Conclusion

TODO: We showed that projecting the image to a frontal view is a good preprocessing step of a robust window detector.

TODO: Furthermore our algorithms work in real time.

8 Conclusion

9 Appendices

9.1 Figures

TODO: check if appendix images not already in ch5!!



Figure 48: Dataset: Anne1, Result edge detection

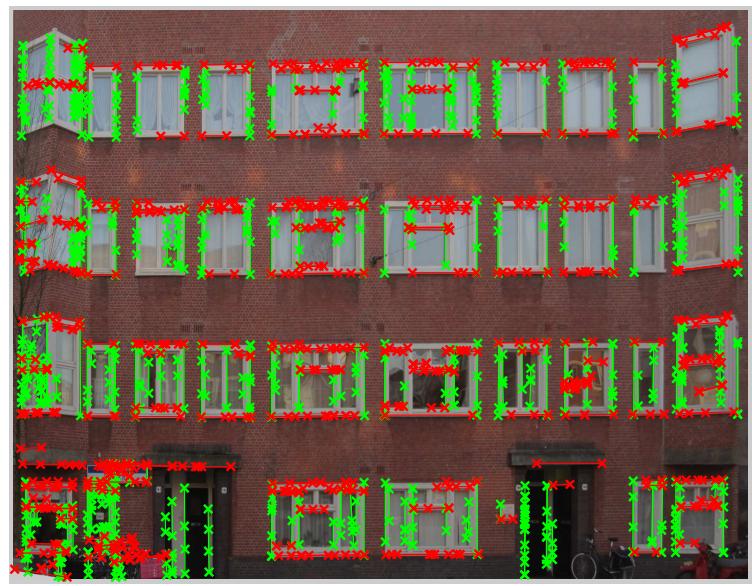


Figure 49: θ -constrained Hough transform

9.1.1 K-means bad luck

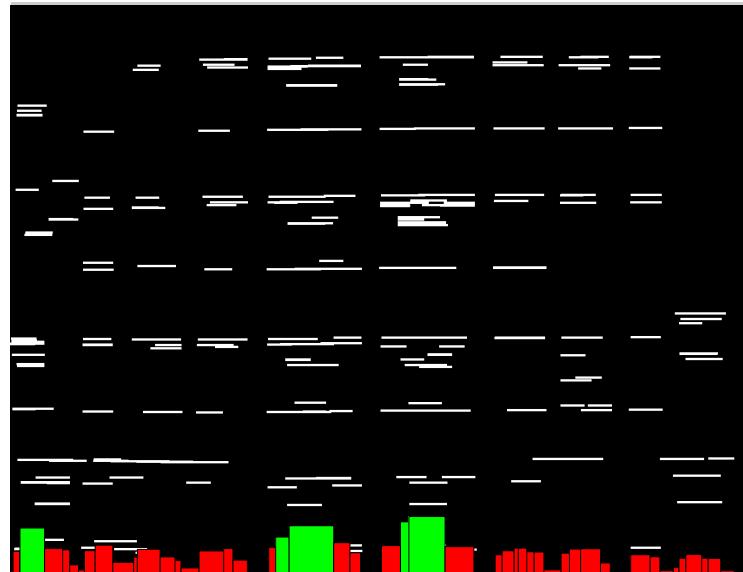


Figure 50: Dataset: Dirk, Horizontal Houghlines

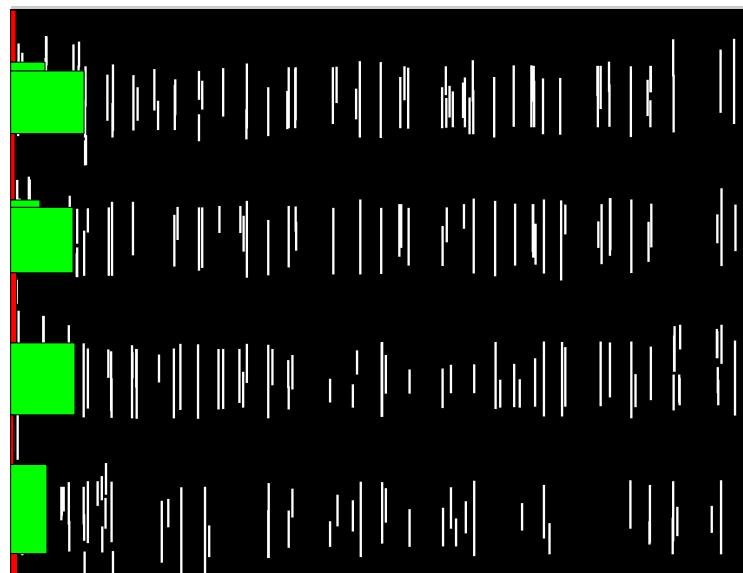


Figure 51: Dataset: Dirk, Vertical Houghlines

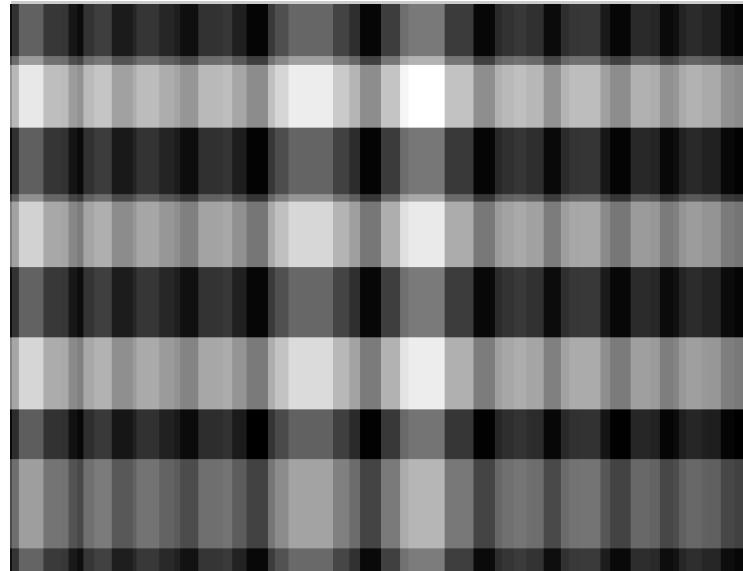


Figure 52: Dataset: Dirk, Basic window classification method:Measure of certainty, the whiter the area the higher the measure of certainty. The values of the window areas in the middle are relative large, this is due the horizontal subwindows.

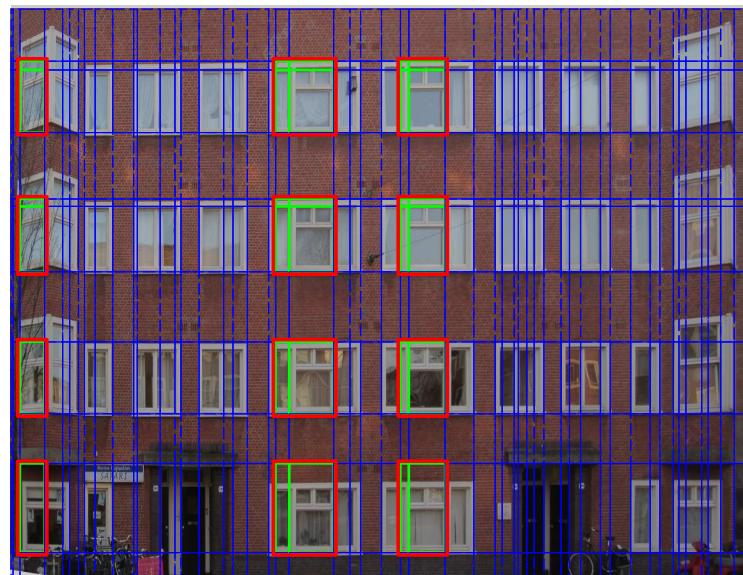


Figure 53: Dataset: Dirk, Extracted windows

References

- [1] 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.
- [2] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.
- [3] 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.
- [4] Fabio Cozman, Eric Krotkov, and Carlos Guestrin. Outdoor visual position estimation for planetary rovers. *Auton. Robots*, 9:135–150, September 2000.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM Trans. Graph.*, 26(3), July 2007.
- [9] 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.
- [10] Shi Pu and George Vosselman. Refining building facade models with images.

- [11] 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.
- [12] 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.