Semantical notation and 3D reconstruction of urban scenes

T. Kostelijk
mailtjerk@gmail.com

April 7, 2012

Contents

1 Skyline detection

1.1 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 flights, car localization, etc. In this project it is used at urban images to provide a contour of a building. The contour will be used to provide 3D information about the scene. This is a novel way of using skyline detection. For our application the skyline detector must be accurate, robust and must operate without any user interaction. This makes it different from existing skyline techniques (e.g. [2],[8],[3]).
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.

1.1.1 Related work

Castano et al. [2] present a clear introduction of different skyline detection techniques.

Detection of dust devils and clouds on Mars In [2], 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 this 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 [8], scientists detect the horizon to stabilize and control the flight of Unmanned

Air Vehicles.
S.M. Ettinger et all [8] 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 2.3 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. [3] 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 [3] as a basis and build a custom algorithm with higher accuracy on top of that. This is explained in the next section.

1.2 Method

1.2.1 Situation and assumptions

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.
The question is: how are we going to detect the sky-building transition point?
In general, the color of the sky is very different than the color of the building. The use of a color-based edge detector would be an intuitive decision. However, the sky and the building itself also contains edges (caused by for example clouds and windows). So how do we determine the right edge? The number of possible edges could be decreased by thresholding the intensity of the edge but it would still be a difficult task to determine the right edge. Furthermore the algorithm 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 [3]. 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.

1.2.2 Related algorithm

To put our work in context, we first describe a related skyline detection algorithm as presented in [3].

To increase the difference between sharp and vague edges, and to let sharp edges stand out more and vague edges disappear, the images are converted to Gaussian smoothed images. The smoothed image is first divided in $\#w$ columns. Next, each column produces a new column that stores its vertical derivatives. This is 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. The system walks through the values of a column, starting from the top. When it detects a pixel with a gradient higher then a certain threshold it stores its y-value (the location of the highest sharp edge of that column) and continues to the next column. The result is a set of y coordinates of length w, that represent the skyline.

1.2.3 Improved algorithm

Taking the smoothed intensity gradient is the most basic method of edge detection and has the disadvantage that it is not robust to more vague edges. It is not surprising that the algorithm in [3] was used in an interactive system where the user has to refine the result.

Our aim is to develop a autonomous skyline detector, the only user interaction that we allow is to provide the system some parameters. 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. The related algorithm uses the smoothed intensity gradient as a method to detect edges. Because of the accuracy disadvantage of this method we took another approach in detecting edges. We tested different edge detecting types.
The output of the different edge detection techniques was studied on an empirical basis and the Canny edge detector came with the most promising results. 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.
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 inlier threshold is set to 1, which means that it finds the first pixel that is white. This is, as in the related algorithm, done from top to bottom for every column in the image.

Because we know we are looking for sharp edges we improved the algorithm by introducing two preprocessing steps. First the contrast of the image is increased, this makes sharp edges stand out more. Secondly the image undertakes an extra Gaussian 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

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.

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.

1.3 Results

Two different datasets are used.
The first dataset is the *Floriande* dataset, which is included in the Fit3d toolbox [5]. The dataset consists of eight images with resolution 1728x1152px.

The second dataset is named the *Spil* dataset and it contains 40 images with resolution 3072x2304px.

The output of the edge detector and skyline detector on the *Floriande* dataset [5] can be seen in Figure 1 We emphasize the effect of different thresholds of the edge detector on the *Spil* dataset in Figure 2 and Figure 3.

1.4 Discussion

Consider Figure 1, 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 2 a change in the threshold parameter of the edge detector can erase tough outliers. In Figure 3 the risk of using a to 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 seperate 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 the part where we can add some artificial intelligence. This is described in the next section.

1.5 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 denote 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.

(a) The output of the edge detector

(b) The output of the skyline detector. The skyline elements are marked red

Figure 1: The output of the edge detector and the skyline detector.

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

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

Figure 2: The skyline detector on two different thresholds

Figure 3: The output of the skyline detector with a too high threshold (0.70)

2 Improving the 3D building

2.1 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 transfered 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.

2.2 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 4, 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.

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 6.

2.3 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.

2.3.1 Hough transform

A widely used method for extracting line segments is the Hough transform [4]. 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*).
In the Hough transform, the main idea is to consider the characteristics of a straight line not as its image points $(x1, y1)$, $(x2, y2)$, etc., 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 input of a Hough transform is a binary image, in our case the output of the skyline detector (chapter 1).
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 (m,b pair) that crosses this particular pixel. Lines (m, b 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* and 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 m,b 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 2D output of the skyline detector are displayed and evaluated in the Result section.

2.4 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.

2.4.1 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* [5] for this. See preliminaries **??** 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 [5] comes with K. For the *Spil* 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 **??**.

2.4.2 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 **??**.

2.4.3 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 **??** 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 wat we want.
Now we have the building divided op 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 in lines (which represent the possible points in 3D) and these lines are intersected with the planes of the building walls. This results in 2xlxw 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.

2.4.4 Associating line segments with building walls

Assumptions We consider each building consisting of seperate 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 appearence 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_{proj_W} can overlap W in four different scenarios, this is explained in Figure 10. The wall W is spanned by $abcd$, and l_{proj_W} 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 2 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 2: Types of overlap with corresponding number op 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, 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:

- Ff $\angle d$ 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 of the point where l intersects the determined vertical wall side (da or cb) and measuring its remaining length.

Full or no overlap Now let's consider the overlap types where the *in polygon* test returned 0. As you can see in Figure 10a and 10d this resulted in either full or no overlap. Again we analyze the vector angles to determine the remaining overlap-type. If only one of the angles is obtuse 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 its 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:

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

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.

2.4.5 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*.

2.5 Results

Let's return to the output of the skyline detector in Figure 1.

Figure 5 shows the top 3 longest Houghlines, the endpoints are denoted with a black and blue cross. All three line segments lie on the building

contour. The left line segment covers only a part of the building wall. The middle line segment covers the full wall. The left and middle line segment are connected. The right line segment covers the wall until the tree occludes. Figure 6 3 displays the line segments (originated from different views) projected 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 7 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.

2.6 Discussion

As can be seen in Figure 5, the top three Houghlines correspond the the three most prominent building walls which is a good result. Wat 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 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 6 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

2.7 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.

2.8 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 *FillGap* 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 te 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 of in the accurateness of the height estimation and the computational costs.

2.8.1 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 8, 6 different roof shapes are displayed.

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 the ground plane and the top roof line to guid the search. Some building 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 predefined 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.

3 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 change of being on a building corner and the on average biggest change 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 5 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).

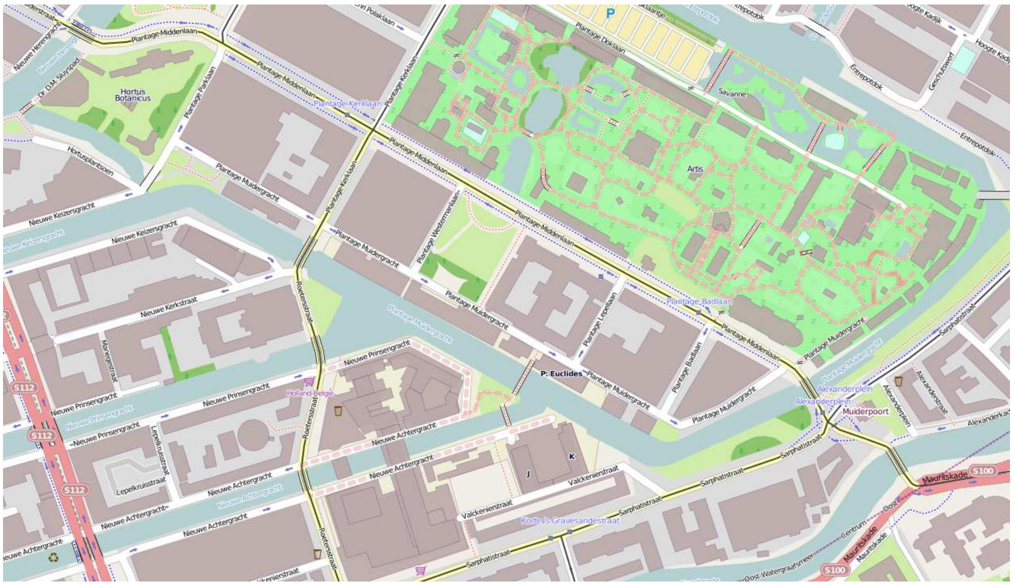

Figure 4: Openstreetmap with annotated buildings

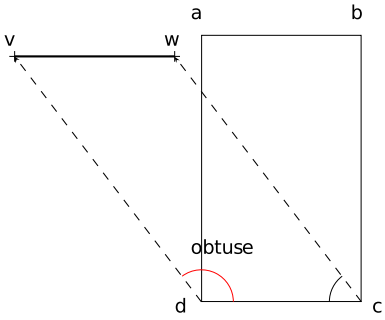

Figure 10a

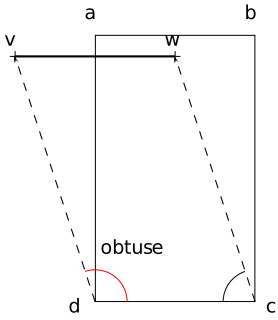

Figure 10b

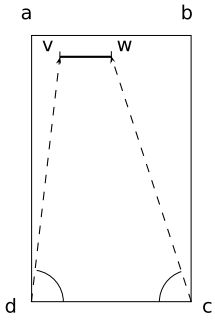

Figure 10c

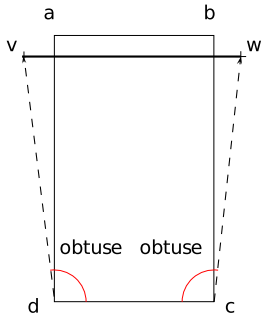

Figure 10d

Figure 5: Three best ranked lines of the Hough transform on the skyline detector

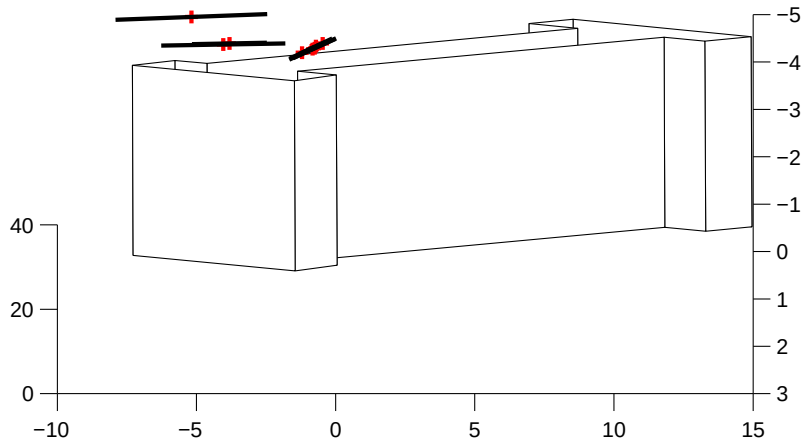

Figure 6: Houghlines projected on the most likely wall

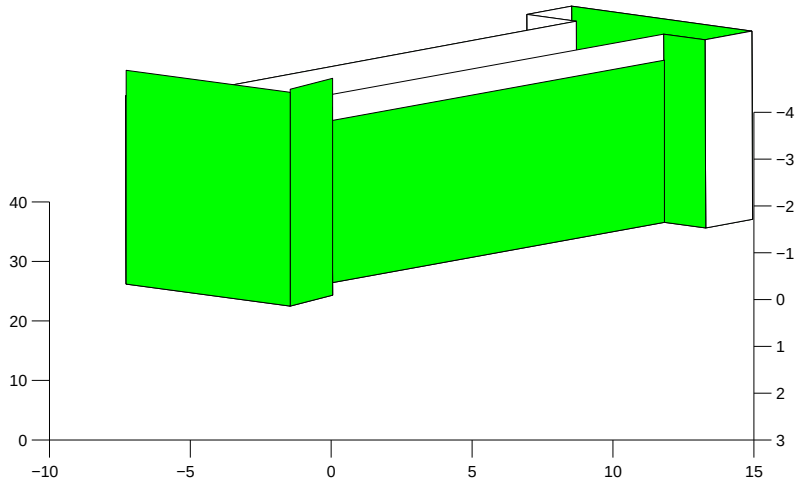

Figure 7: Improved 3D model

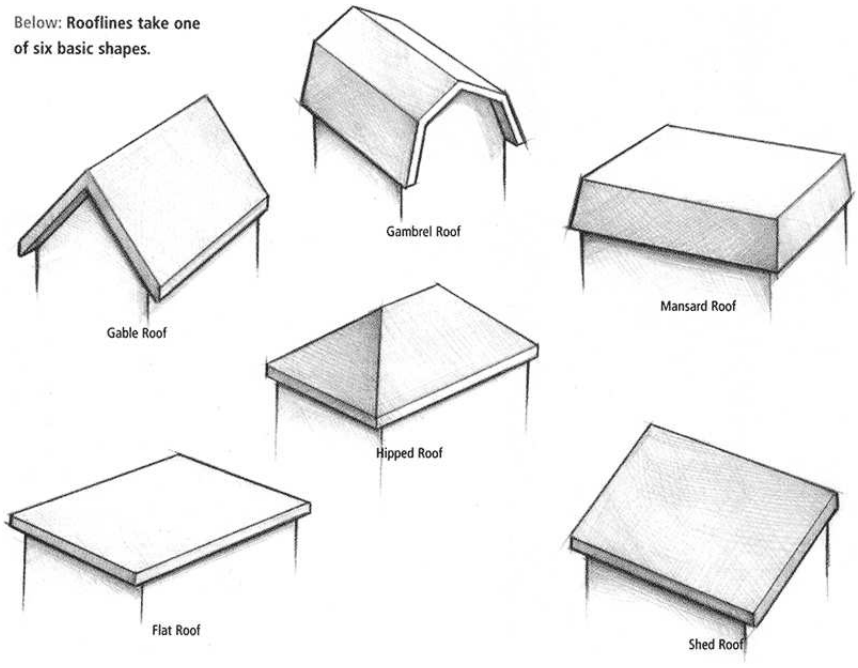

Figure 8: Different types of roofs

4 Window detection

4.1 Updated/New since 28-3-20012

al het commentaar verwerkt
explanation different types of connected corners
Future research-¿ Window alignment refinement
Discussion -¿ nummeric evaluation of connected corner result in
ik werk niet meer met houghline eindpunten maar met de gehele lijn.
ipv rectangular areas, en rows, gebruik ik nu blocks en blockrows

4.2 Q

Is it useful to include the connected corner result of the occluding tree? 23
Zal ik de presentatie met de blokken met grijswaarden weglaten?

4.3 Introduction

This chapter deals with one of the tasks of semantic urban scene interpretation, Window detection. Semantic interpretation of urban scenes is used in a wide range of applications.

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.

Augmented reality Some mobile platforms apply augmented reality using facade and window detection to make a 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.

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. This chapter presents two developed methods for robust window detection.
We start with discussing related work and putting our work in context. Then we describe a window detection approach that is invariant to viewing direction. After this we present our second method that assumes orthogonal and aligned windows. Finally we show and discuss results.

4.4 Related work

A large amount of research is done on semantical interpretation of urban scenes. First we discuss related work that has a big overlap with our approach in detail. After this, we briefly discuss the research that is done on window detection using other approaches.

4.4.1 Similar approaches

Pu and Vosselman [9] use laser images together with Hough line extraction 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 Hough lines of ground images. They match the lines by comparing the angle, location and length difference of the model edges with the extracted Houghlines. These criteria is also used in our approach.
They also detect windows and use them to provide a significant better alignment of the 3d model. The windows are extracted from the holes from laser points of a wall, these results where far from accurate.
To summarize, the work of Pu and Vosselman provides a useful practical application of window detection and it amplifies the need for a robust window detection technique that is independent of laser data.

In [10] Recky et all developed a window detector that is build on the primary work of Lee and Nevatia [6] (which is discussed next). In order to be able to assume aligned windows they rectify the facade. After this they apply a threshold on an orthogonal projection of the extracted edges. For example they use a vertical edge projection to establish the horizontal division of the windows which is very similar to our approach,.
The next step, labeling the areas containing windows, is however very different as they use color to disambiguate the windows. 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. In our method we use the same source, edge information, for the window alignment and for the window labeling. As we don't require color transformation and only apply math on line segment endpoints, our algorithm performs in real-time.
As in the work of Recky et all [10] Lee et all [6] perform orthogonal edge projection to find the window alignment. As different shape of windows can exist in the same column, they use the window alignment as a hypothesis.

Then, using this hypothesis, they perform a refinement for each window independently. More on this in Future research.

4.4.2 Other approaches

Muller et all [7] detect symmetry in the building. The symmetry is detected in the vertical (floors) and horizontal (window rows) direction. The 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.
Using a thermal camera, Sirmacek [11] 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*.
Ali et all [1] describe the windows with Haar like features which are fed into a (Ada boost) cascaded decision tree.

4.5 Method I: Connected corner approach

4.5.1 Situation and assumptions

We introduce the concept *connected corner*, this is a corner that is connected to a horizontal and vertical line. In this method we search for connected corners based on edge information. The connected corners give a good indication of the position of the windows, as a window consists of a complex structure involving a lot of connected horizontal and vertical lines.
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.

4.5.2 Method

Edge detection and Houghline extraction Edge detection is done as is described in chapter *(not included chapter)* From the edge image we extract two different groups of Houghlines, horizontal and vertical. We set the θ bin ranges in the Hough transform that control the allowed angles of the Houghlines to extract the two groups. The horizontal group has a range of [-30..0..30] degrees, where 0 presents a horizontal line. The vertical group has a range of [80..90..100] degrees. These ranges seem to work well on an empirical basis for all datasets. The results of two images can be seen in Figure 11 and 12.

Figure 9: Original image

Figure 10: Rectified image

Figure 11: Result edge detection

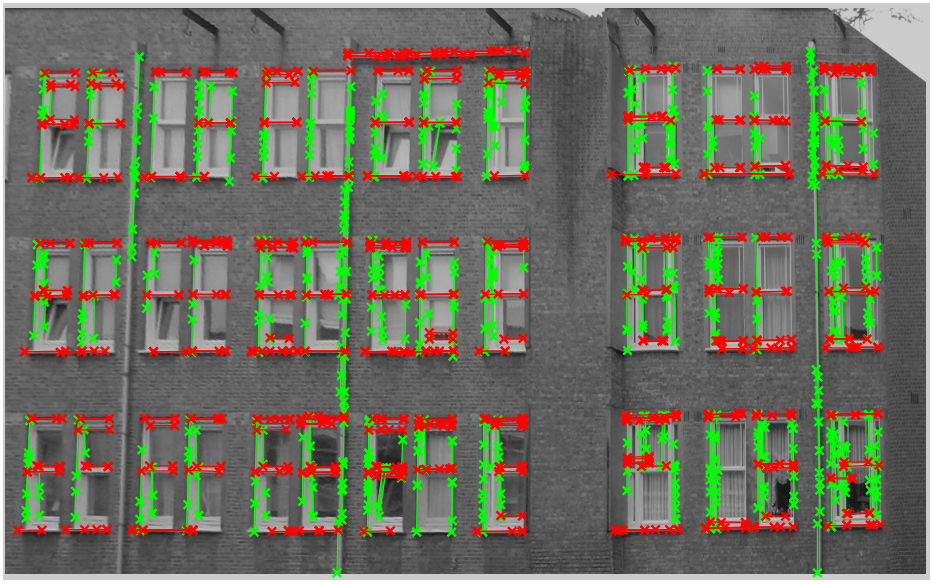

Figure 12: Houghlines with endpoints

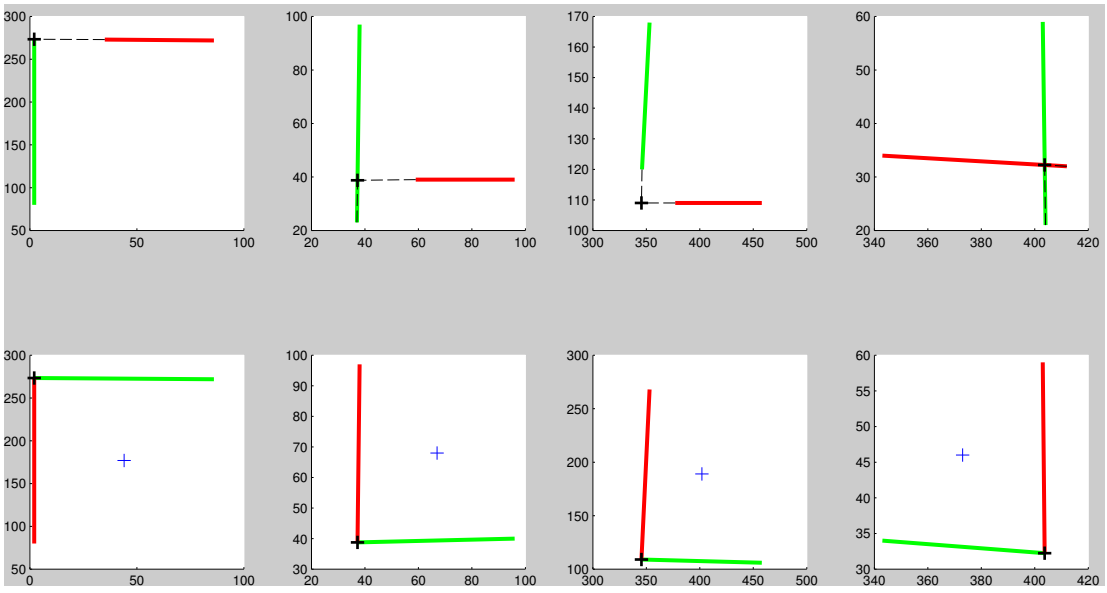

Figure 13: First row: different type of connected corner candidates. Second row: the result the clean connected corner

Extract connected corners As windows contain complex structures the amount of horizontal and vertical houghlines is large at these locations. A horizontal and vertical line is 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 wich we tollerate, these cases are illustrated in Figure 13 in the top row. A horizontal gap a vertical and horizontal gap and a vertical alongation. 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 between the intersection point and the endpoint of the lines is measured, this is illustrated as dotted lines in Figure 13. Next, D is compared to a *maximum intersection distance* threshold $midT$. And if $D <= midT$, 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 13. In Figure 13(I) the horizontal line is extended. Figure 13(II) shows that the vertical line is trimmed. In Figure 13(III) both lines are extended. At last, Figure 13(IV) shows how both lines are trimmed.

Extract window areas 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 20 and 24. This result is discussed in section 4.7.

4.6 Method II: Histogram based approach

4.6.1 Introduction

From the previous chapter we know 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 another viewing point.
For accurate and robust window detection we projected the scene 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. First we determine the the alignment of the windows and then we label the areas that contain the windows.

4.6.2 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.

4.6.3 Method

The extraction of the windows is done in different steps. First we rectify the image making the assumptions are valid. The rectification process is done as described in chapter *TODO REF* 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.

Efficient Projecting As we are interested in the frontal view of the building, it would be straight forward to project the original image. However this is computational very expensive as each pixel needs to be projected. To keep the computational cost to a minimum we project only the Houghlines. The edge detection and Houghline extraction is done on the original unprojected image. We only project the Houghline segment endpoints. If h is the number of Houghlines, the number of projections is $2h$ When we project the original image this is w x h where w,h are the dimensions of the image. To give an indication, for the *Spil* dataset this means 600 projections in stead of 1572864.
However for the purpose of display we also presented the rectified images.

Extract Window alignment We introduce the concept alignment line. We define this as a horizontal or vertical line that aligns multiple windows. In Figure 14 we show the alignment lines as two groups, horizontal (red) and vertical (green) alignment lines. The combination of both groups give a grid of rectangles 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 start and end (see red crosses in Figure 12. For the vertical alignment lines the number of vertical Houghline start and ends is high (see green crosses in Figure

12.

We begin by extracting the pixelcoordinates 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 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(orizontal) and V(ertical), containing respectively w and h bins where wxh is the dimension of the image. The histograms are presented as small yellow bars in Figure 14.
The peaks are located at the positions where an increased number of Houghlines start or end. These are the interesting positions as they are highly correlated to the alignment lines of the windows.
It is easy to see that the number of peaks is far more then the desired number of alignment lines. Therefore we smooth the values using a moving average filter. The result, red lines in Figure 14 , is a smooth function 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 Peak. (This value works for most datasets but is a paramater 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 14.
The image is now divided in a new grid of blocks based on these alignment lines. The next challenge is to classify the blocks as window and non-window areas: the window classification.

Window classification Instead of classifying each block independently, we classify full rows and columns of blocks as window or non-window areas. This approach results in more 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 12 (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.

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

Leaving us with $||R||$ (number of blockrows) scalar values that give a rank of a blockrow begin a window area or not. This is also done for each blockcolumn (using the normalized horizontal amount of Houghlines pixels) which leaves us with C.
If we examine the distribution of R and C, we see two clusters appear: one with high values (the blockrows/blockcolumns that contain windows) and one with low values (non window blockrows/blockcolumns). For a specific example we displayed the values of R in Figure 15. Its easy to see that the high values, blockrow 4,5,7,8,10 and 11, correspond to the six window blockrows in Figure 25.
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 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

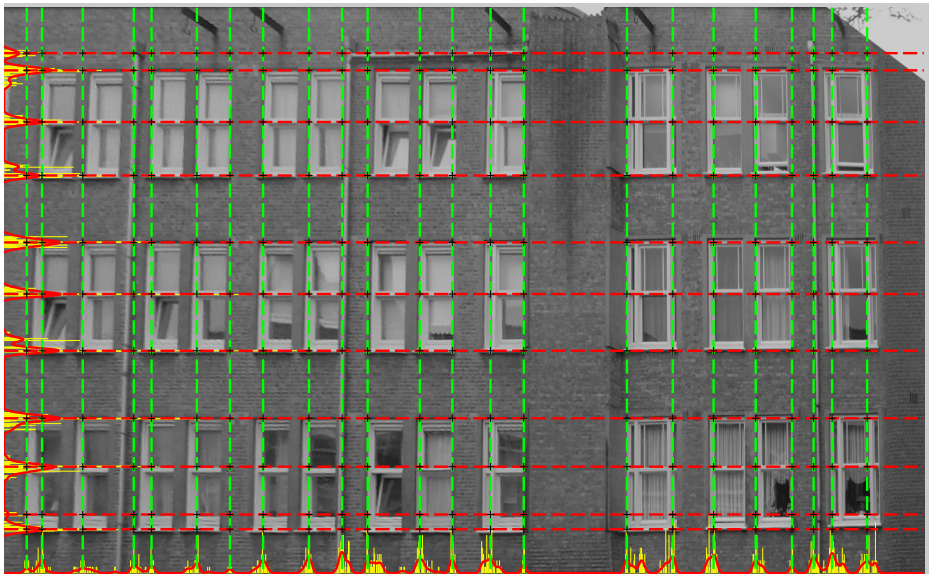

Figure 14: (smoothed) Histograms and window alignment lines

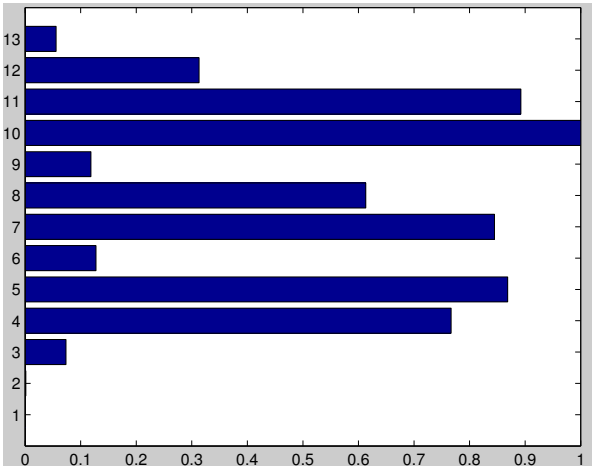

Figure 15: Normalized vertical Houghline pixel count of the blockrows (R)

$k = 2$) as the classification procedure. This results in a set of Rows and Columns that are classified as window an non-window areas.
The next step is to determine the actual windows W. A rectangular area $w \in W$ 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 25 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 rectangles. These are displayed as red rectangles in Figure 25.
To get insight about the probabilities that lie behind the individual blockrows and blockcolumns we designed another representation in Figure 16 The whiter the area the more probable a rectangle is classified as a window.

4.7 Results

. We tested both methods on different datasets.

TODO include images other datasets
todo compair methods and explain differences

4.8 Discussion

todo discuss results

Method I: Connected corner approach Figure **??** contains 110 windows of which are 109 detected, this is 99%. Furthermore there are some False Positive areas, this is about 3 %. The window on the right top isn't detected, this is because he is smaller then our minimum window width.
The big advantage of this method is that it doesn't require the windows to be aligned. Furthermore it's robust to a variation in window sizes and types. This makes this approach suitable for a wide range of window scenes where no or few prior information about the windows is known. *TODO*

Method II: Histogram based approach It could be a drawback that the outcome is non-deterministic, as it depends on to the random initialization of the cluster centers. Our results could be correct by coincidence. To exclude this artefact, we ran the cluster algorithm 10 times, fortunately it resulted in the same classification.
TODO

Occlusion If the image isn't the frontal view of the buildingwall we project the image see section ?This projection comes with some difficulties, occlusion. In a few cases an buiding wall extension (middle of figure 9) a drainpipe or the building wall itself is occluding a part of the window. The less frontal the view, the more occlusion negatively effects the cleanness of the projection. However, this occlusion artefact is in most cases no problem as the system combines the windows probabilities.

4.9 Conclusion

TODO

4.10 Future research

4.10.1 Method I: Connected corner approach

It would be nice to group the connected corner to groups of subwindows. The big window that contains subwindows could be found by calculating the convex hull of the red areas in Figure 20. The subwindows 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 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 probability of the subwindow.

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 or the angle of viewing.

4.10.2 Method II: Histogram based approach

It would be nice to investigate the effect of the occlusion and to exploit 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*

Window alignment refinement To get more accurate result or to handle scenes with poor window alligment a refinement procedure could be applied. As mentioned in the related work, Lee et all [6] 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 of. A system that only refines the results when the recources 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.

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

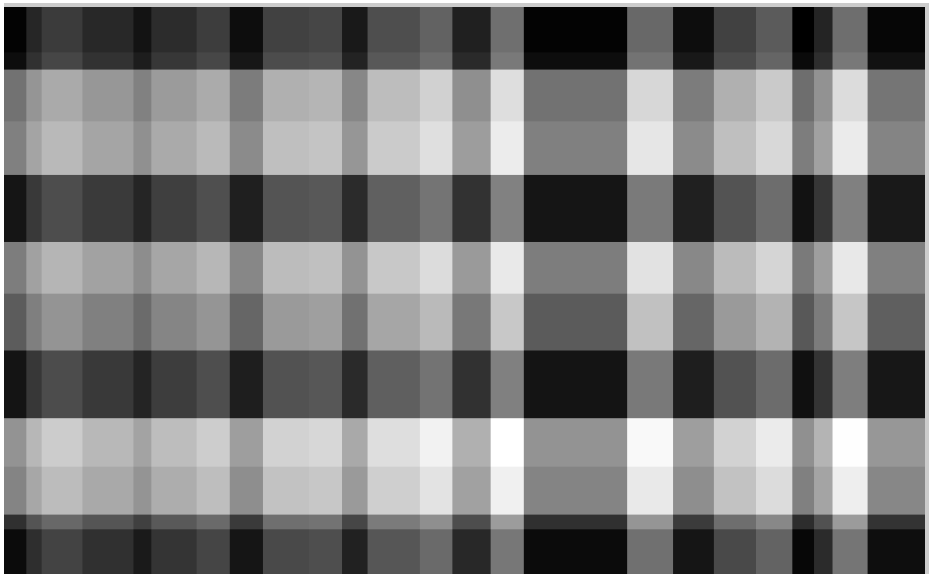

Figure 16: Window classification probabilities, white means high.

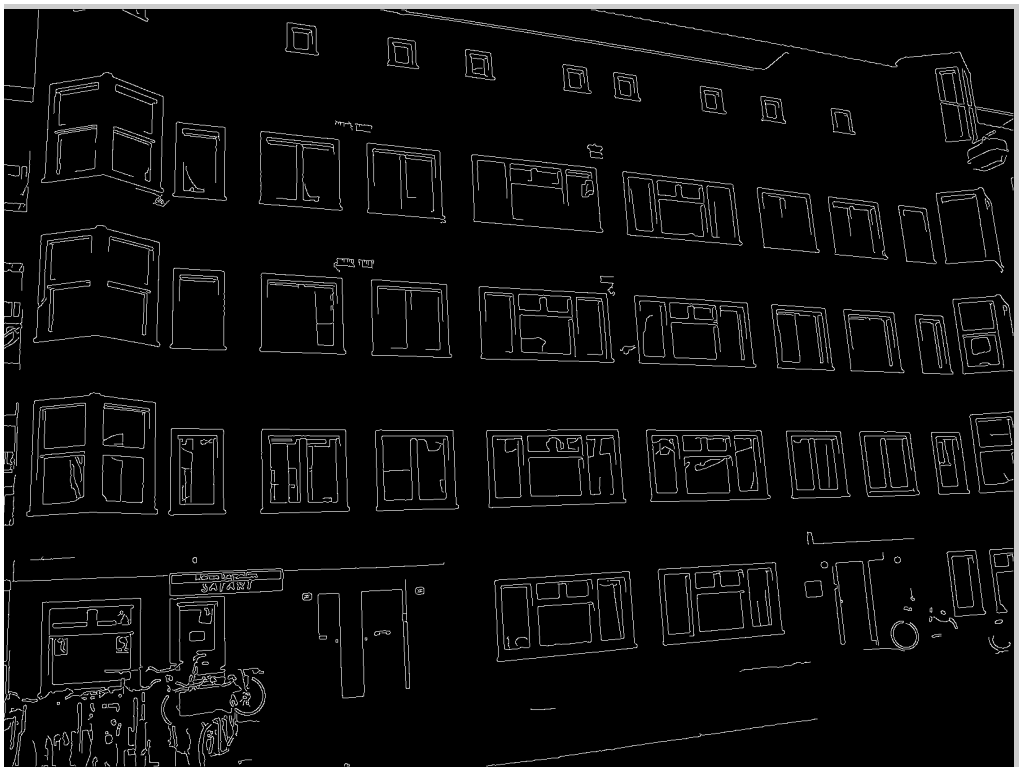

Figure 17: Edge detection

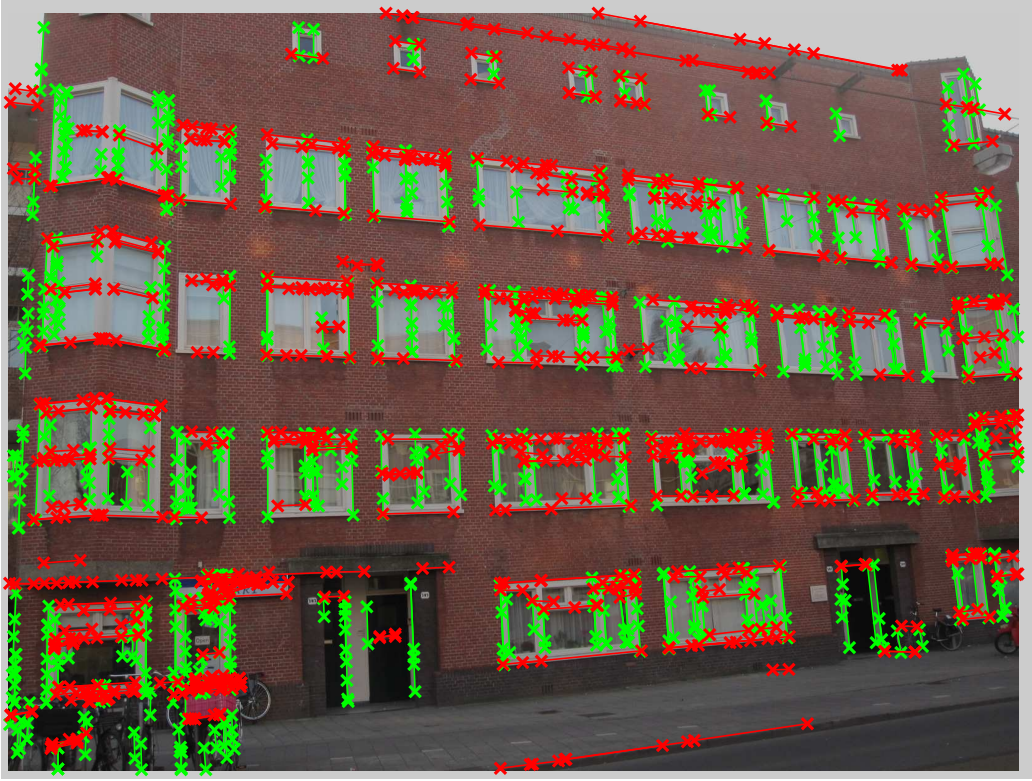

Figure 18: Result of θ constrained Hough transform

Figure 19: Found connected corners

Figure 20: Window regions

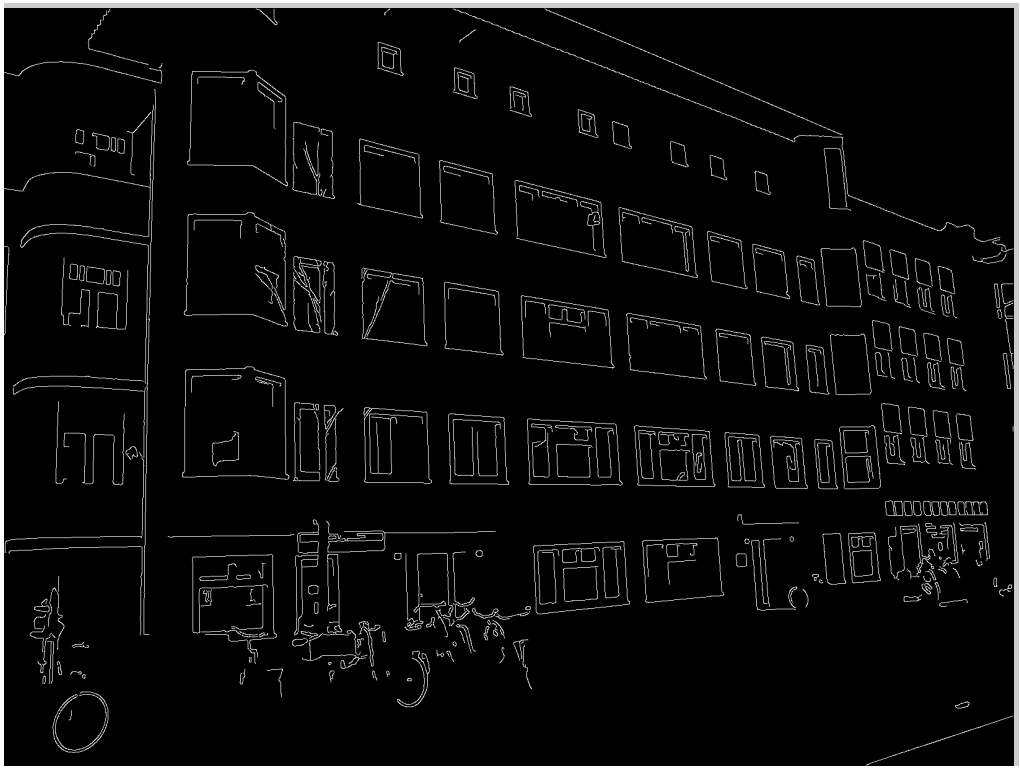

Figure 21: Edge detection (with occluding tree)

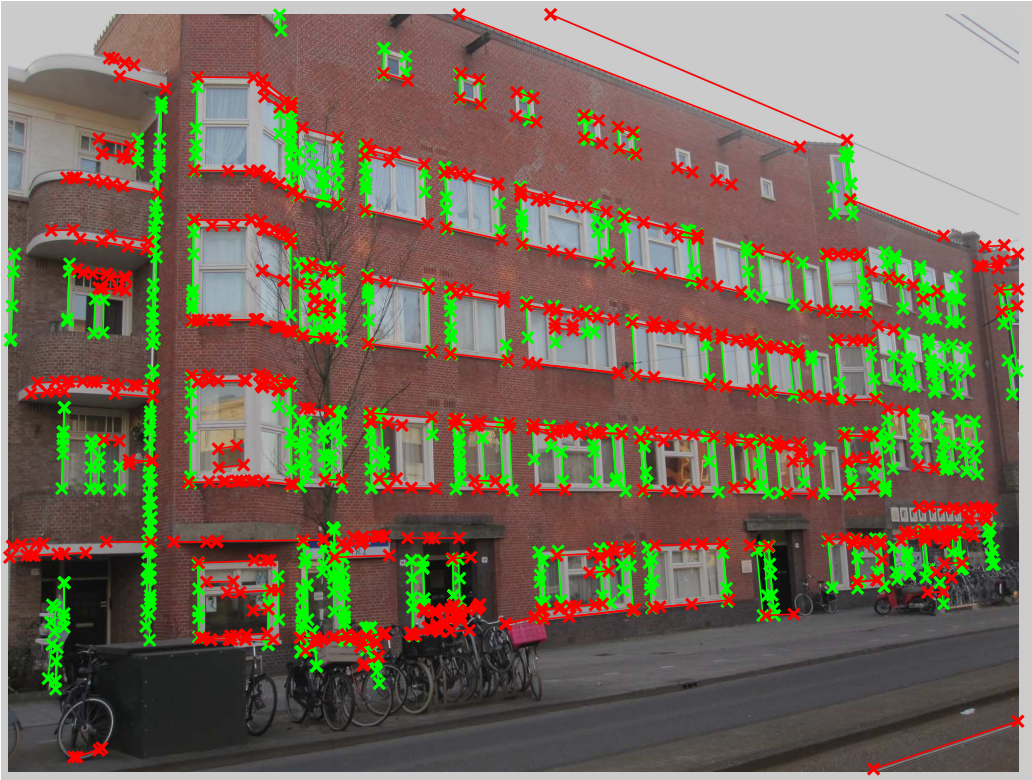

Figure 22: Result of θ constrained Hough transform (with occluding tree)

Figure 23: Found connected corners (with occluding tree)

Figure 24: Window regions (with occluding tree)

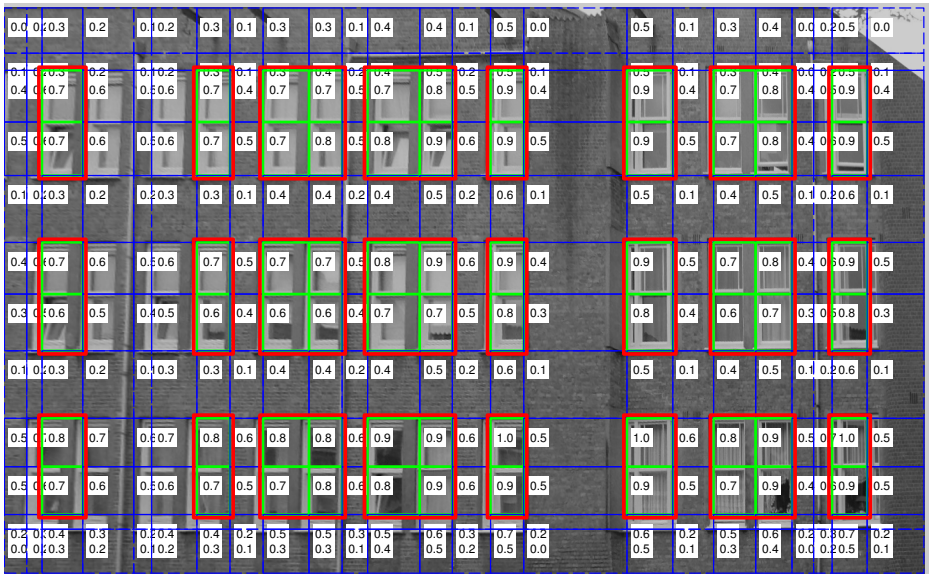

Figure 25: Classified rectangles

Figure 26: Original Image

Figure 27: Window alignment lines and histograms

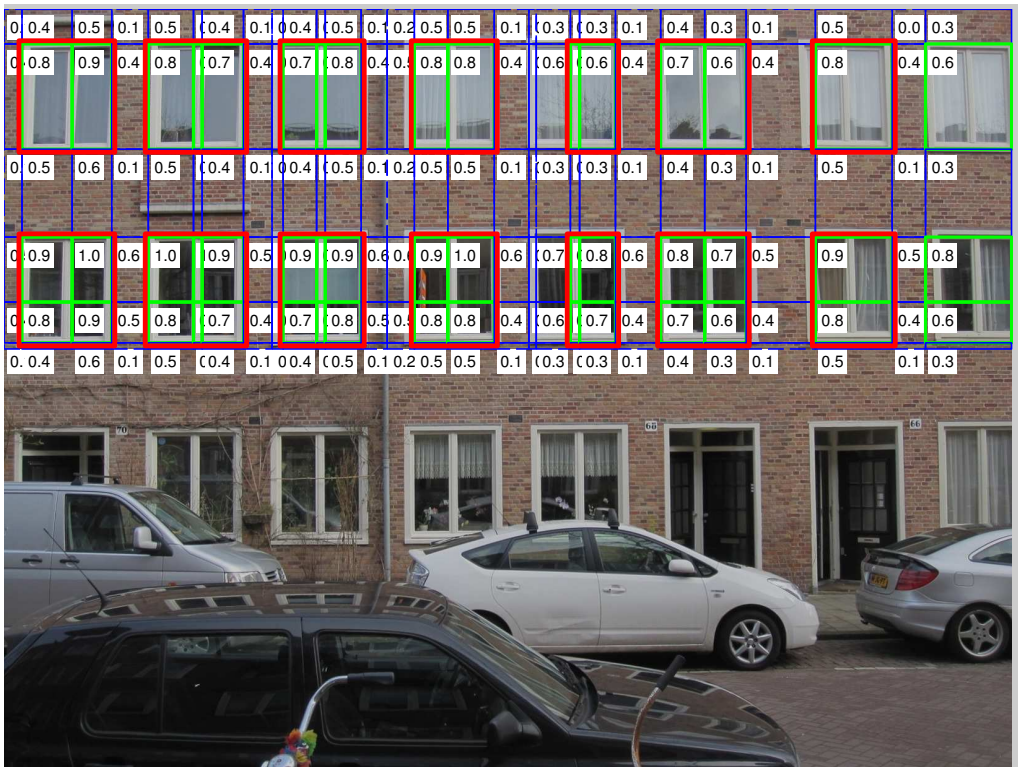

Figure 28: Classified rectangles

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] 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.

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.