# Improve 3D models from 2D images

T. Kostelijk
mailtjerk@gmail.com

March 9, 2012

## Contents

# 1 Window detection

## 1.1 Introduction

In this section we present two developed methods for robust window detection . *todo motivation*
From the previous section we know that from a serie 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.
We projected the scene to a frontal view of a building, where a building wall appears orthogonal, and showed what interesting possibilities this opens. A frontal view of a building comes with orthogonality and alignment of the windows. We exploit this properties to build a to build a robust window detector This projection also comes with some difficulties which we discuss in 1.6
We start with discussing related work and putting our work in context. Then we begin with 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.

### 1.1.1 Related work

*TODO*

## 1.2 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.
*As the backprojection isn't done at the time of writing we we also present the rectified image*

Figure 1: Original image

## 1.3 Method I: Connected corner approach

### 1.3.1 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 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.

### 1.3.2 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 $\theta$

Figure 2: Rectified image

Figure 3: Result edge detection

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

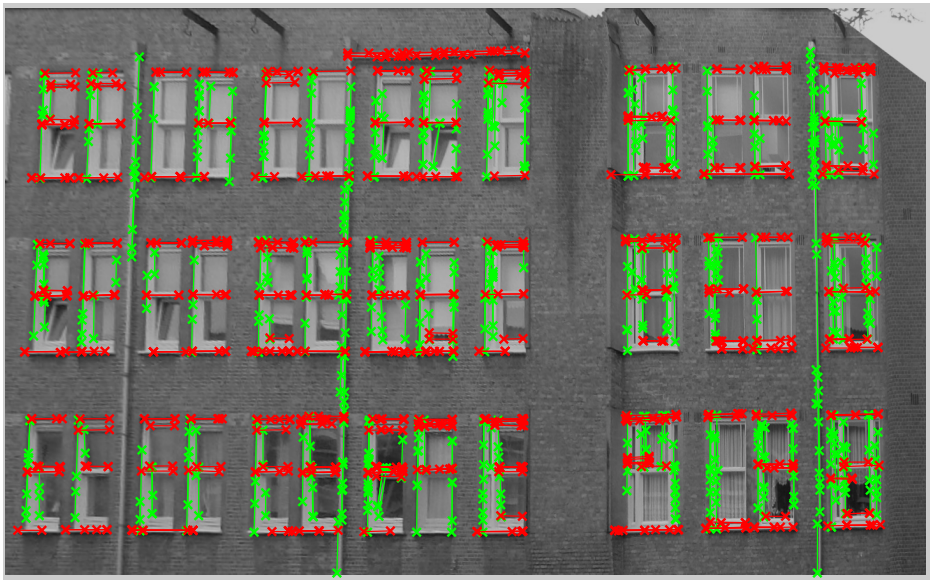

Figure 4: Houghlines with endpoints

[-30..0..30] degrees, where 0 presents a horizontal line. The vertical group has a range of [80..90..100] degrees. This ranges seem to work good on an empirical basis for all datasets. The results can be seen in Figure 3 and 4.
Next we pair up horizontal and vertical lines to discard found edges that are not origined from a window. These paired up lines form a connected corner. Often a connected corner is not fully connected or over connected. We consider different types of connected corners, see Figure 5.
To clean up the lines, the algorithm discards intersections between the horizontal and vertical Houghlines that are apart. Two intersection point distances are measured: $d_h$ for the horizontal Houghline and $d_v$ for the vertical Houghline. If the intersection falls on both associated Houghlines, the total distance $D = 0$. Otherwise the Euclidean distance is measured from the intersection to the closest endpoint. This is done for both Houghlines. If the intersection falls outside both Houghlines (Figure 5(IV)) ($d_h > 0$ and $d_v > 0$), the total distance is calculated by $D = (d_h + d_v)/2$.
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 stretched

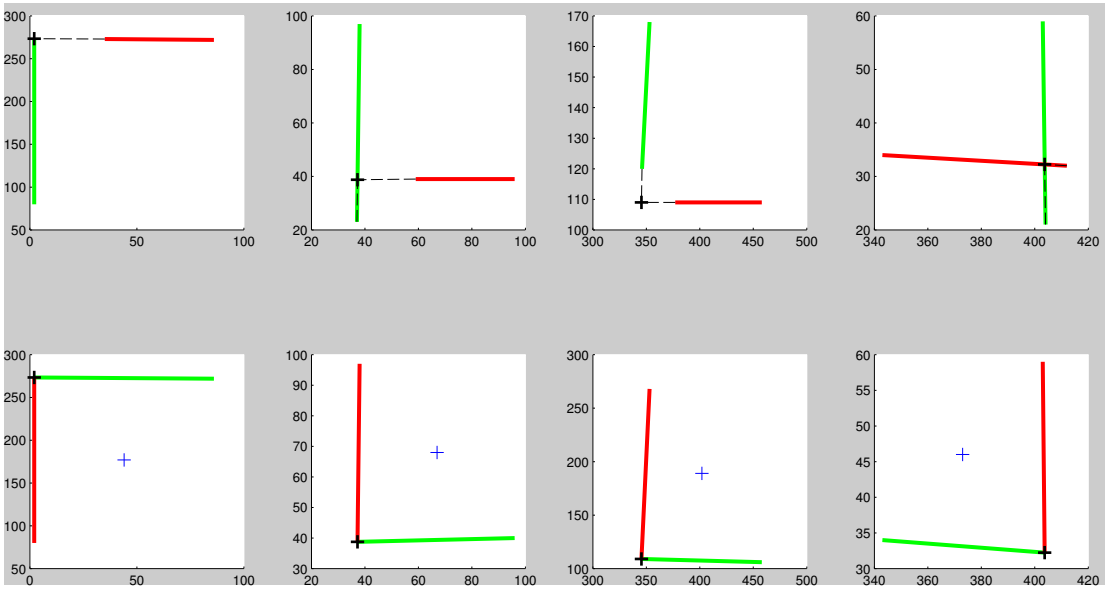

Figure 5: First row: different type of connected corner candidates. Second row: the result the clean connected corner

or trimmed, depending on the situation. The results are shown in the second row in Figure 5. In Figure 5(I) the horizontal line is stretched. Figure 5(II) shows that the vertical line is trimmed. In Figure 5(III) both lines are stretched. At last, Figure 5(IV) shows how both lines are trimmed.
Because we know the orientation of the connected corner we can estimate where a window could be located. We add a vote in the middle of the window. This is represented as a blue cross in Figure 5. This coordinate is retrieved using the X coordinate of the middle point of the horizontal line and the Y coordinate of the middle point of the vertical line of the connected corner. Note that this is officially not allowed as we did not assume orthogonal windows neither did we assume the windows to be aligned with the X and Y axis of the image. However on a empirical basis most crosses lie within the original window which is enough. Results are found in the Result section.

## 1.4 Method II: Histogram based approach

### 1.4.1 Introduction

In this method we assume that the viewing direction of the wall containing the windows is frontal. This assumption makes it possible to exploit the orthogonality and alignment of the windows. We first determine the alignment of the windows and then extract the windows.

### 1.4.2 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.

### 1.4.3 Method

The extraction of the windows is done in different steps. First the alignment of the windows is determined, this is based on collecting the Houghlines' start and endpoints. Then 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.

**Extract Window alignment** We introduce the concept alignment line. We define this as a horizontal or vertical line that aligns multiple windows. In Figure 6 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 4. For the vertical alignment lines the number of vertical Houghline start and ends is high (see green crosses in Figure 4.

We begin by extracting the coordinates of the endpoints of the Hough transformed line segments. We store them in two groups, horizontal and vertical.We discard the dimension that is least informative by project the coordinates to the axis that is orthogonal to its group. This means that for each horizontal Houghline two coordinates are projected to the X axis and for

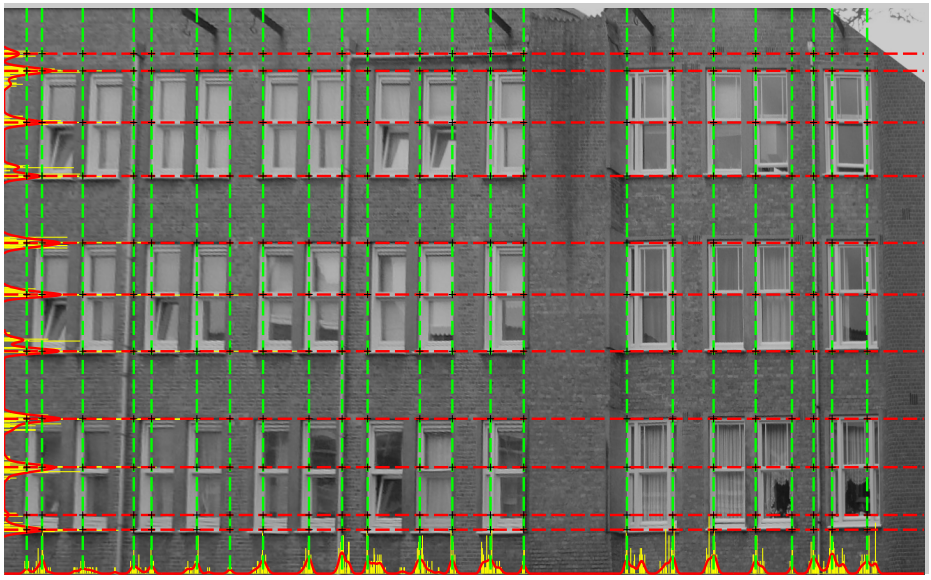

Figure 6: (smoothed) Histograms and window alignment lines

each vertical Houghline two 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 6.
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 fare more then the desired number of alignment lines. A common solution would be to decrease the number of bins of the histograms. A disadvantage of this method is that this comes with a price, it decreases the accuracy. Therefor we keep the maximum resolution and, instead, smooth the values using a moving average filter. The result, red lines in Figure 6 , 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 threshold-

ing 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 made up and can be altered). Next we create a binary function P that 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 classified 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. Therefor 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 6.
The image is now divided in a grid of rectangular areas. The next challenge is to classify the areas as window and non-window areas: the window classification.

**Window classification** Instead of classifying each rectangle independently we classify full rows and columns as window or non-window areas. This approach results in more accurate classification as it uses a full row and column as evidence for a singular window.
The method exploits the fact that the windows are assumed to be aligned. A row that contains windows is remarkable by its high amount of vertical Houghlines, Figure 4 (green). For the columns the number of horizontal Houghlines (red) is high at window areas. We use this property to classify the rows/columns.
For each row the number of vertical Houghline pixels that lie in this row 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 row influences the outcome, this total value is normalized by the size of the row.

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

Leaving us with $||R||$ (number of rows) scalar values that give a rank of a row begin a window area or not. This is also done for each column (using

the normalized horizontal amount of Houghlines pixels) which leaves us with $C$.

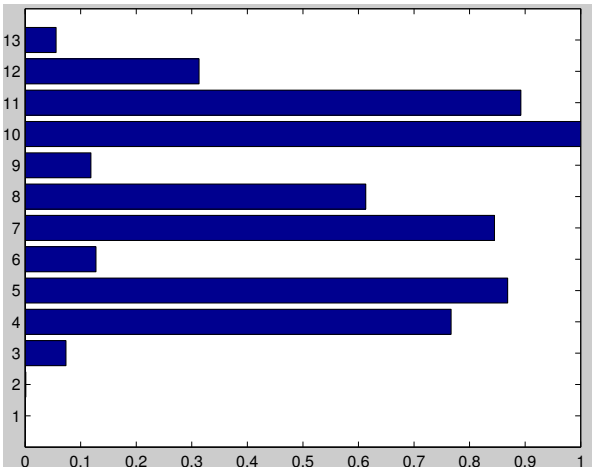


Figure 7: Normalized vertical Houghline pixel count of the rows (R)

If we examine the distribution of $R$ and $C$, we see two clusters appear: one with high values (the rows/columns that contain windows) and one with low values (non window rows/columns). For a concrete example we displayed the values of $R$ in Figure 7. Its easy to see that the high values, row 4,5,7,8,10 and 11, correspond to the six window row areas in Figure 14.
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 row 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. 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 14 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 dis-

played as red rectangles in Figure 14.
As the figure gives a binary representation of the windows it is not possible to see the probabilities behind the classification. Therefor we developed a probabilistic function.

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

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

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

As you can see $P$ is normalized, this is to ensure the value of the maximum probability is exactly 1. The results can now be relatively interpreted, e.g. if the rectangle's $P = 0.5$ then the system nows for 50 percent sure it is a window, compared to its best window ($P = 1$). And, as the normalization implies this, there are always one or more window with $P = 1$.
To get insight about the probabilities that lie behind the individual rows and columns we designed another representation in Figure 8 The whiter the area the more probable a rectangle is classified as a window.

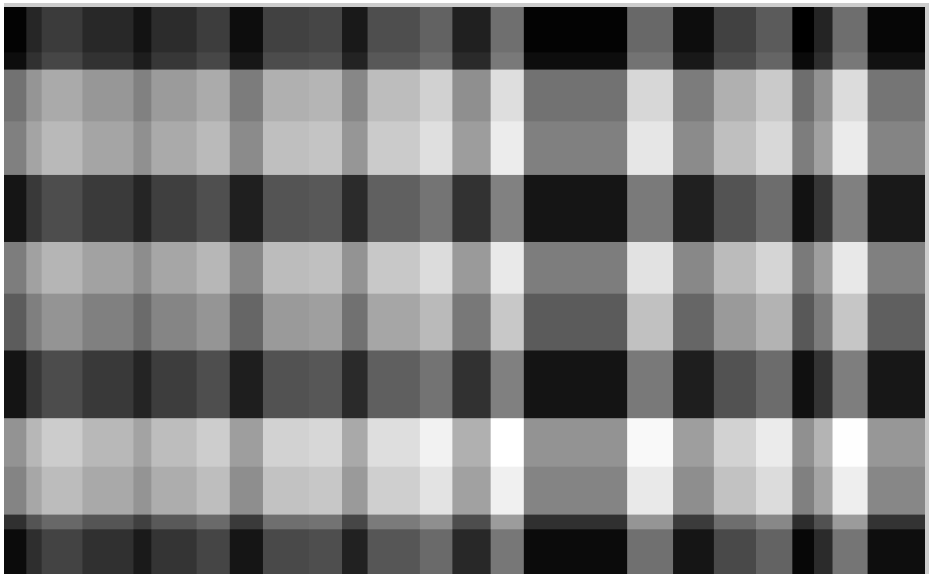

Figure 8: Window classification probabilities, white means high.

## 1.5 Results

We tested both methods on different datasets.

Figure 9: Edge detection

*TODO include images other datasets*
*todo compair methods and explain differences*

## 1.6 Discussion

*todo discuss results*

**Method I: Connected corner approach** A disadvantage of this method is that it only finds plausible window centers (and for each window center one of the corner positions). It would be more useful to find the complete window region. However, for the purpose of estimating a people count for a real-time evacuation plan generator, the count of windows would be enough.

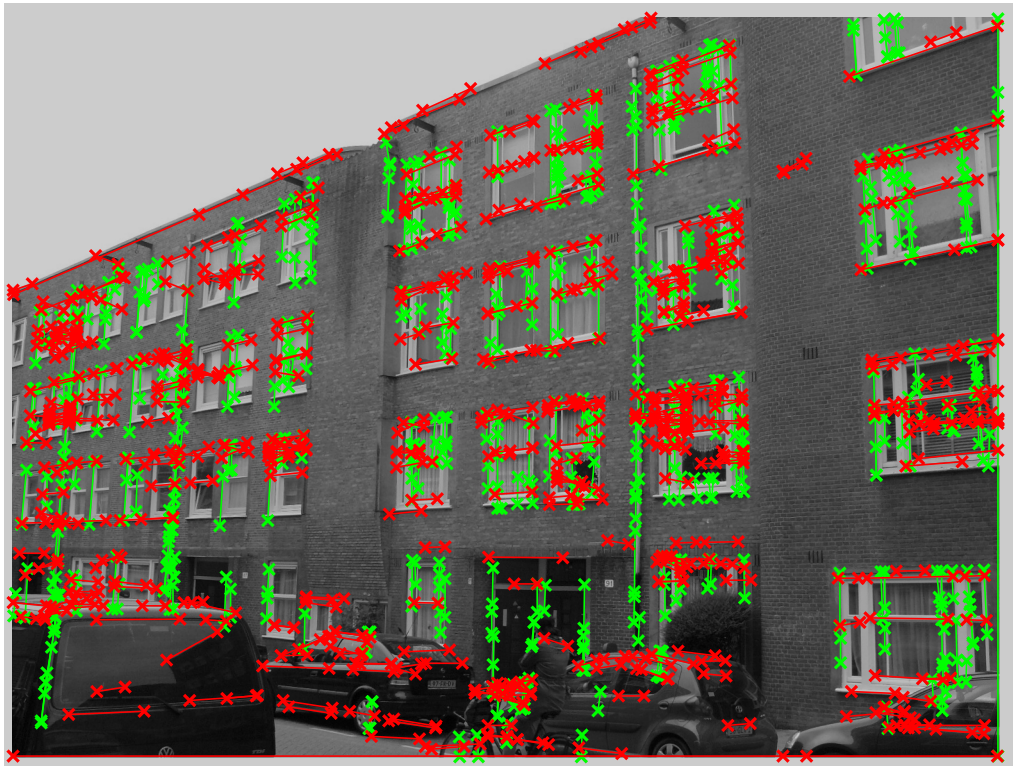

Figure 10: Result of $\theta$ constrained Hough transform

The big advantage is that this method doesn't require the windows to be aligned. Furthermore it's robust to a variation in window sizes. 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** One drawback is that the outcome is non-deterministic, as it depends on to the random initialization of the cluster centers.
*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 extention (middle of figure 1) a drainpipe or the building wall itself is occluding a part of the window. The less frontal

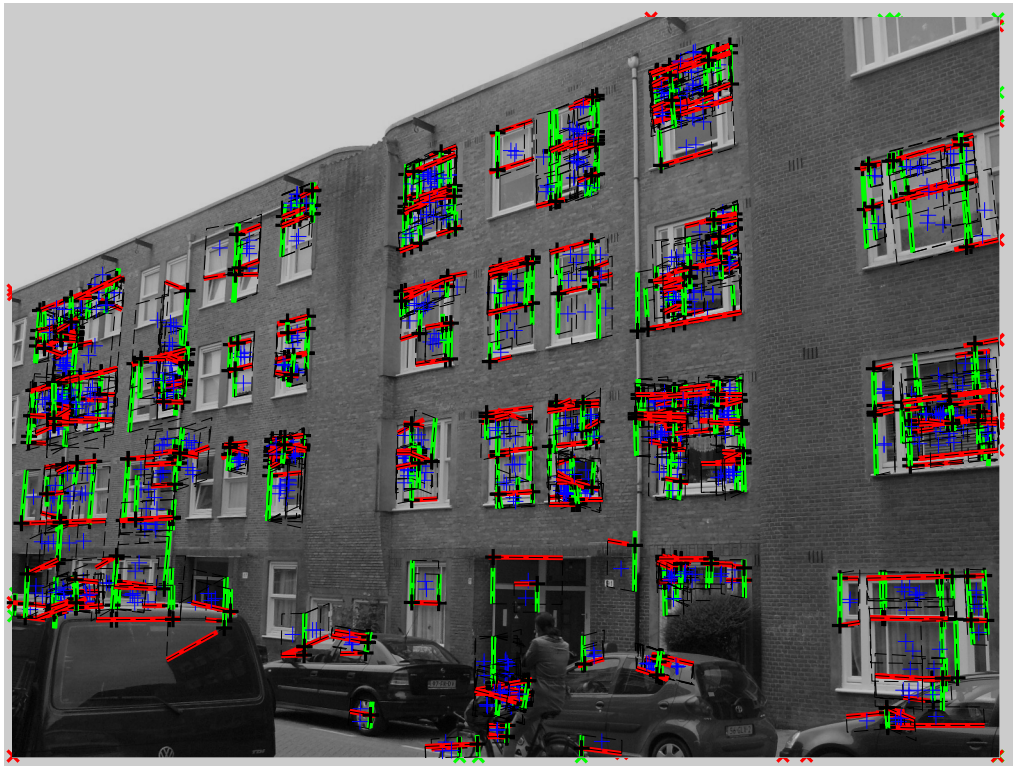

Figure 11: Found connected corners

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.

## 1.7 Conclusion and Future work

### 1.7.1 Conclusion

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

### 1.7.2 Future work

**Method I: Connected corner approach** It would be nice to have a clustering algorithm that groups connected corners to a window. For this method it would be useful to assume the window size as this correlates di-

Figure 12: Connected corner as windows

rectly 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 that a window is found.
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.

**Method II: Histogram based approach** It would be nice to investigate the effect of the occlusion and to exploit the robustness of the window de-

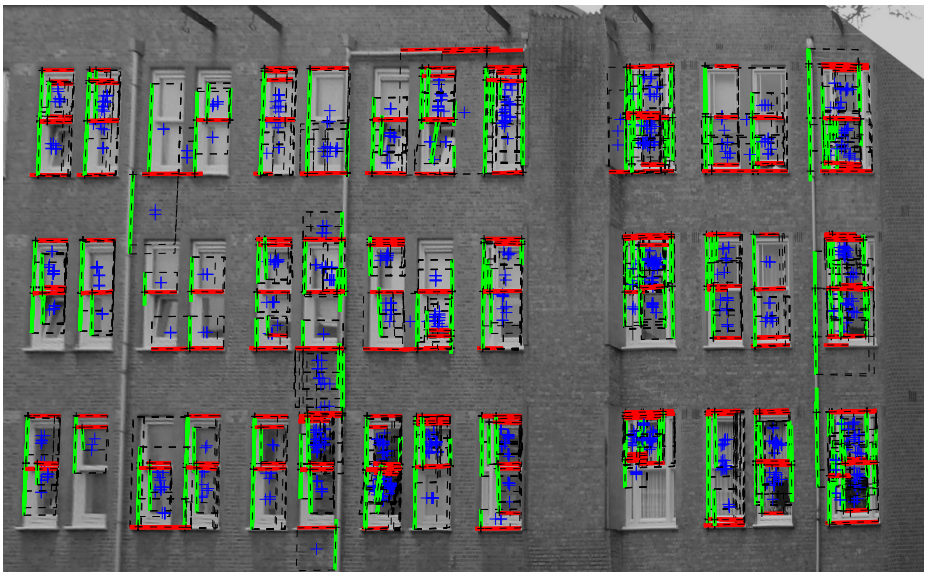

Figure 13: Found connected corners on the rectified image

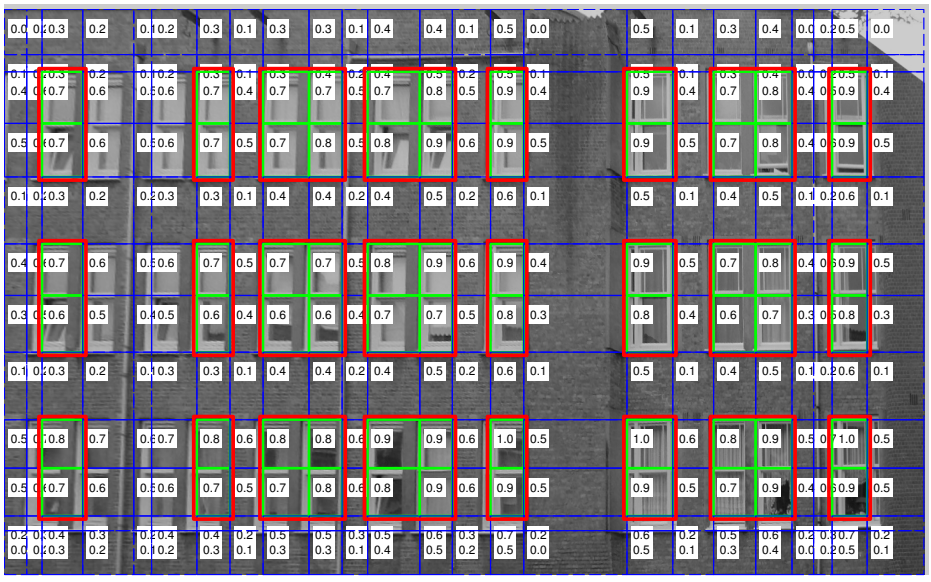

Figure 14: Classified rectangles

Figure 15: Original Image

tector under extreme viewing angles. For example the viewing angle could be plotted against the percentage of correct detected windows. *TODO*

Figure 16: Window alignment lines and histograms

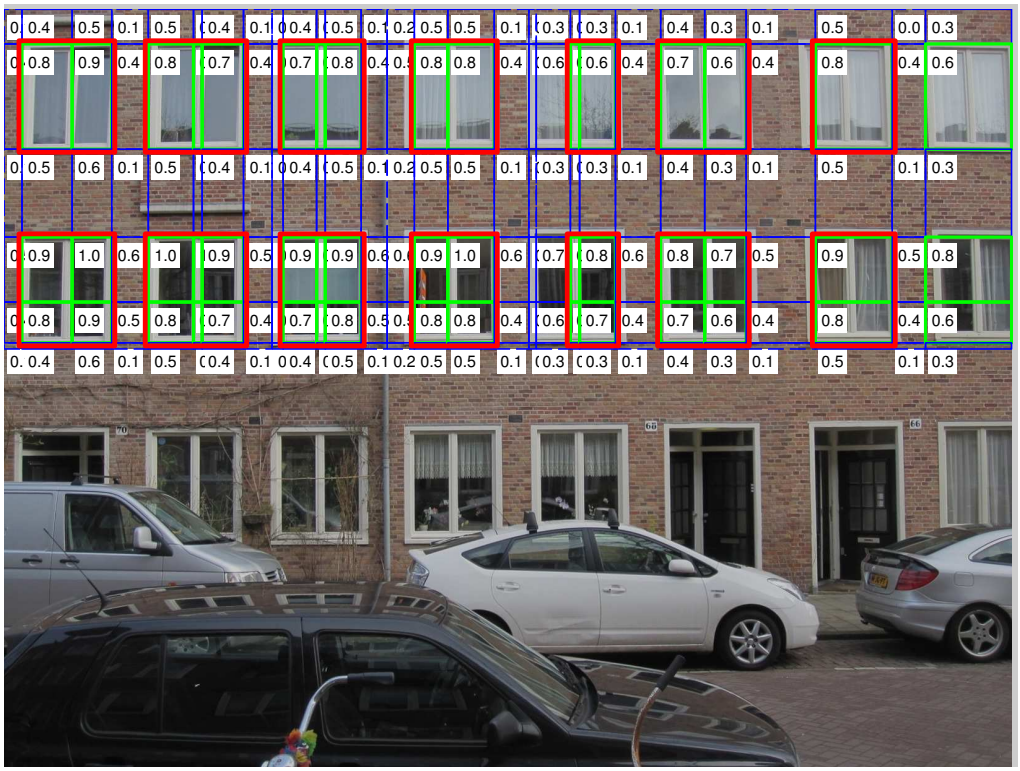

Figure 17: Classified rectangles