3D reconstruction and semantic annotation of urban scenes

T. Kostelijk
mailtjerk@gmail.com

May 11, 2012

Contents

Todo list

1 Window detection

1.1 Introduction

Semantic interpretation of urban scenes is used in a wide range of applications. We deal with an important aspect of semantic urban scene interpretation, Window detection.
This chapter is organised as follows, we start with an variety applications that use semantic interpretation of urban scenes. Then we discuss related work and put our work in context. Next we describe our first window detection approach that is invariant to viewing direction. After this we present a facade rectification method. Next this method is used for our second method that assumes orthogonal and aligned windows. Finally we show and discuss results.

1.2 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.

Figure 1: Simulation environment

Augmented reality

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

Building recognition and urban planning

Building recognition is used in the field of urban planning where the semantic 3d models are used to provide important references to the city scenes from the street level. Building recognition is done using large image datasets where the buildings are mostly described by local information descriptors. Some approaches try to describe the 3D building with laser range data. Some methods fuse the laser data with ground images. However those generated 3D models are a mesh structure which doesn't make the facade structure explicit. For a more accurate disambiguation, other types of contextual information are desired. The semantical interpretation of the facade can provide this need. In this context, window detection can be used as a strong discriminator.

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

1.3 State of the art window detection

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

1.3.1 Alternative approaches on window detection

Muller et all [4] detect regularity and symmetry in the building. The symmetry is detected in the vertical (floors) and horizontal (window rows) direction. They use shape grammars to divide the building wall in tiles, windows, doors etc. The results are used to derive a 3d model of high 3d visual quality. Although they achieved some interesting results their method has some disadvantages. As their method is fully based on detecting symmetrie, they have to assume repeating and aligned windows: this constraints the variety of scenes the system can handle. Furthermore they match template window objects which they predefine, this constraints the variety of windowtypes that could be matched. At last they use expensive algorithms that make it impossible for the system to run in real time.

(a) original texture (b) reconstruction

Figure 2: Results of Muller et all

Using a thermal camera, Sirmacek [7] detects heat leakage on building walls as an indicator for doors or windows. Windows are detected with L-shaped features as set of *steerable filters*. The windows are grouped using *perceptual organization rules*: they search and group intersecting L-shapes to close a window shape. A shape is determined closed if it can seperate an inside region from the outside, i.e. determine if it is a window.

Ali et all [1] describe the windows with *Haar* like features, the features are combined using a cascading classifier. The cascading classifier, which acts like a decision tree, is learned using the *Ada boost* algorithm. They also use window detection to determine the region of the facade.

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

To investigate this, we developed two methods on which one method doesn't require repeating windows nor aligned windows. One of the main targets of our research is a low requirements on the input data. First our system doesn't need a large annotated dataset. Furthermore we took the images

with a mobile phone and decided to extract the windows from the image space only, this makes us independent of additional expensive data like heat or laser range images. It doesn't mean the previous work on window detection using laser or heat images isn't of good use. Instead we learned a lot from the previous research as they have to match the laser or heat data to the real image space. This matching process involves a description (semantical annotation) of the facade. Let's explain a method of that kind and discuss other approaches that are more similar to our approach.

1.3.2 Similar approaches

Pu and Vosselman [5] combine laser range images with ground images to reconstruct facade details. They solve inconsistency between laser and image data and improve the alignment of a 3d model with a matching algorithm. In one of the matching strategies they compare the edges of a 3d model to extracted Hough lines of both ground and laser range images. They match the lines by comparing the angle, location and length differences. These criteria are also used in our approach.

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

The work of Pu and Vosselman [5] provides a useful practical application of window detection and it amplifies the need for a robust window detection technique that is independent of laser range data.

Recky et all [6] developed a window detector that is build on the primary work of Lee and Nevatia [3] (which is discussed next). In order to make clear orthogonal projections they rectify the facade. To determine the alignment of the windows the edges are projected into their orthogonal direction. For example the horizontal edges are projected in the vertical direction to to establish the vertical division of the windows.
A function is developed that counts the amount of projected edges on each location, this is the *Projection profile*, see Figure 3. A threshold is applied on the projection profile to indicate the window boundaries that is used for the window alignment.

Figure 3: Projection profiles of Lee and Nevatias work

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

As in the work of Recky et all [6] Lee et all [3] perform orthogonal edge projection to find the window alignment. As different shapes of windows can exist in the same column/row, they only use the window alignment as a hypothesis. Then, using this hypothesis, they perform a refinement for each window independently. Although this comes with accurate results, the iterative refinement is a computational expensive procedure. As we want to run our system in realtime this method is not suitable for our application.

1.3.3 Our work in context

The state of the art window alignment procedure in [6] and [3] is very robust. Therefore we decided to use this method as a basis and improved the alignment algorithm, furthermore we build a different window classification method.
Our improvement on the alignment procedure is as follows. In the previous work [6] and [3] they use a single projection profile for each direction. We

improved this process by fusing two (more advanced) projection profiles for each direction. E.g. for the determination of the horizontal division of the windows we fuse both horizontal and vertical projection profiles.

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

1.4 Method I: Connected corner approach

1.4.1 Situation and assumptions

A window consists of a complex structure involving a lot of connected horizontal and vertical lines, we use this property to detect the windows. We introduce the concept *connected corner*, this is a corner that is connected to a horizontal and vertical line. The search for these connected corners is based on edge information. The connected corners give a good indication of the position of the windows,
In this approach the viewing direction is not required to be frontal. The windows could be arbitrarily located and they don't need to be aligned to each other neither to the X and Y axis of the image. As such the windows are detected individually.

1.4.2 Edge detection and Houghline extraction

Edge detection is done as it is described in section **??**. From the edge images two groups of Houghlines are extracted. The groups fall in the two window directions horizontal and vertical. This is done by controling the allowed angles, θ bin ranges, in the Hough transform. The horizontal group has a range of $\theta = [-30..0..30)$ degrees, where $\theta = 0$ presents a horizontal line. The vertical group has a range of $\theta = [80..90..100)$ degrees.
We use these ranges because 1) the user hardly ever holds the camera exactly orthogonal. 2) we work with unrectified facades, meaning we deal with perspective distortion. To be more concrete, if the user takes a photo (Figure 4 with a certain Yaw $\bar{0}$, the horizontal lines become skew. The range of the vertical group is smaller then the horizontal group as the user often takes photo with a low pitch value and a high yaw.

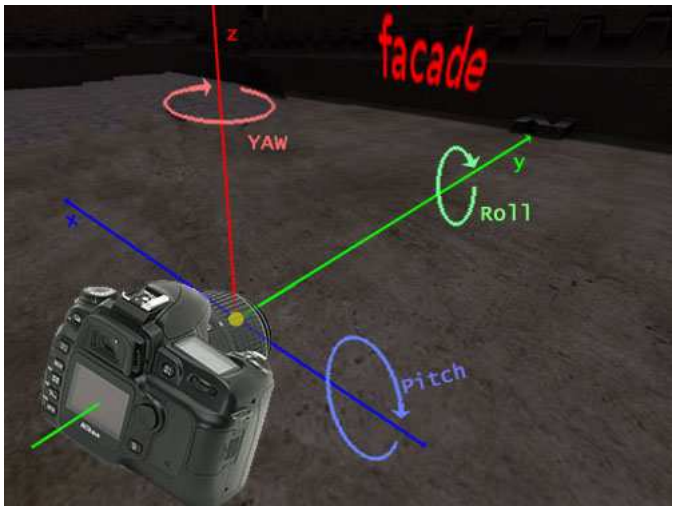

Figure 4: Pitch roll and yaw of the camera

The results of the edge detection and the Hough transform of two images can be seen in Figure 6 and 7.

1.4.3 Connected corners extraction

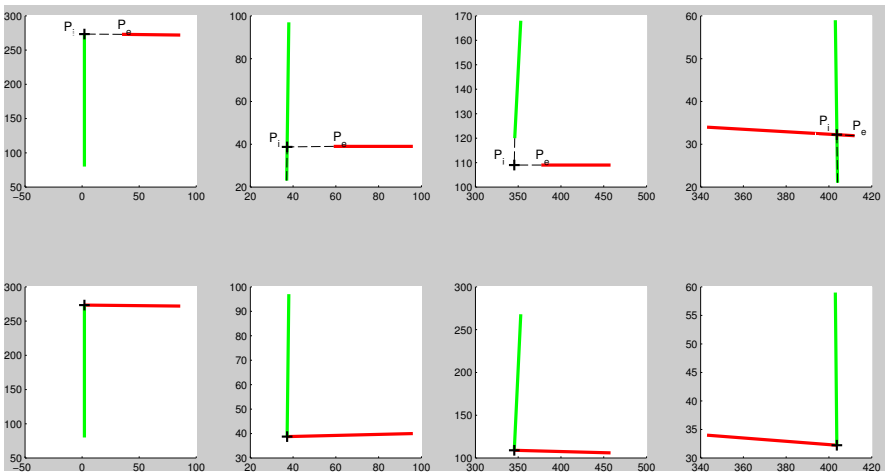

Figure 5: First row: different type of connected corner candidates. Second row: the result the clean connected corner

As windows contain complex structures the amount of horizontal and verti-

cal 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 which we tolerate, these cases are illustrated in Figure 5 in the top row. A horizontal gap, a vertical and horizontal gap and a vertical elongation. The cleaned up corners are given in the bottom row. When the horizontal and vertical lines intersect, the gap distance is $D = 0$. When the lines do not intersect, the distance D between the intersection point P_i and the endpoint P_e of the line is measured $D = ||P_i - P_e||$, this is illustrated as dotted lines in Figure 5. 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 5. In Figure 5(I) the horizontal line is extended. Figure 5(II) shows that the vertical line is trimmed. In Figure 5(III) both lines are extended. At last, Figure 5(IV) shows how both lines are trimmed.

1.4.4 Window area extraction

To retrieve the actual windows, each connected corner is mirrored along its diagonal. The connected corner now contains four sides which form a quadrangle window area. All quadrangles are filled and displayed in Figure 9, this result is discussed in section 1.6.6.

1.4.5 Results

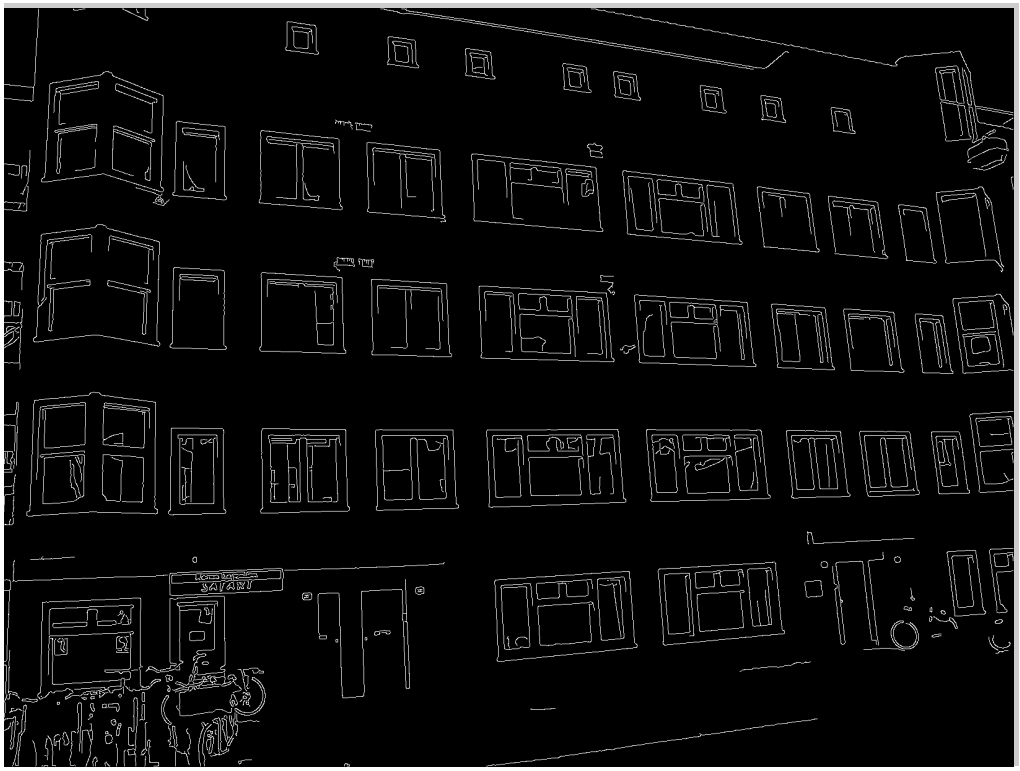

Figure 6: Edge detection

Figure 9 contains 110 windows of which are 109 detected, this is 99%. Furthermore there are some False Positive areas, this is about 3 %.

1.4.6 Discussion

The window on the right top isn't detected, this is because its smaller then the 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*

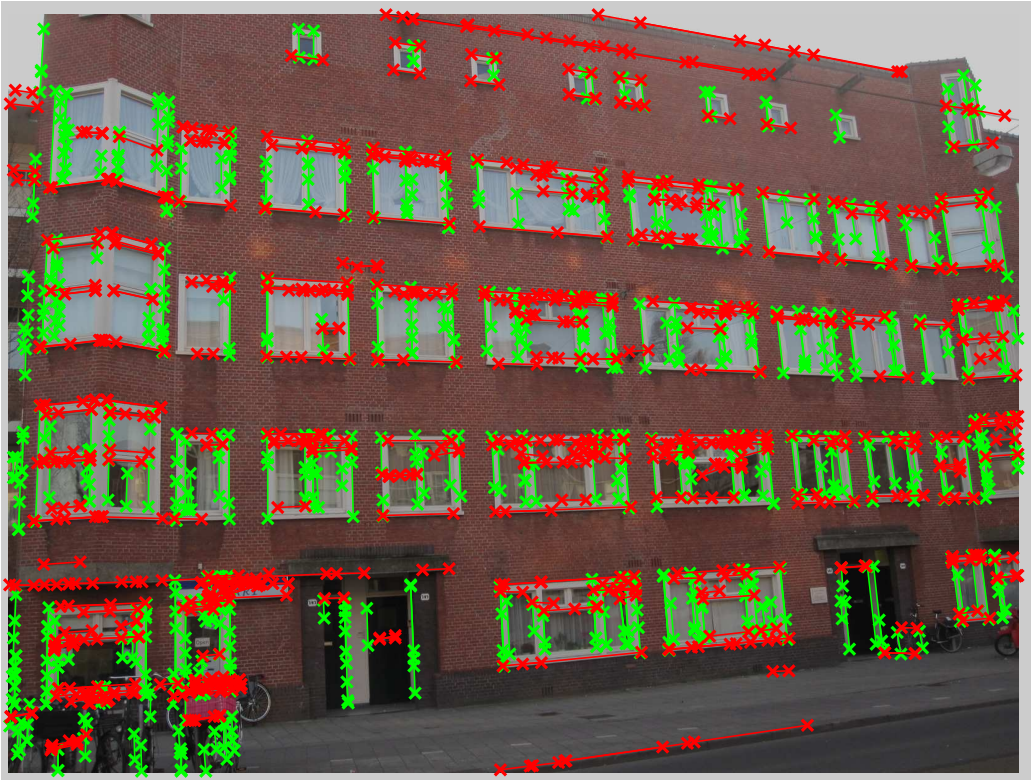

Figure 7: Result of θ constrained Hough transform

1.5 Facade rectification

1.5.1 Introduction

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

The simple rectification method uses point to point correspondences. This requires annotation of the corner points of the facade that are mapped with the corners of a rectangle. This mapping is used to calculate a transformation matrix. The downside of this method is that it isn't very accurate. This is because it only uses four point to point correspondences and doesnt take the camera lens distortion into account. Another downside is that it requires (manual) annotation of the corner points. A more accurate and fully automatic method is desired .

Figure 8: Found connected corners

The second method involves the extraction of a 3D plane of the facade. This method is more complex and gives more accurate results. It involves a comprehensive process and lots of research is done in this area. Given the related work, the automity of this module and our focus on the annotational part, we used existing software to apply the window rectification. I. Estebans *FIT3D toolbox* [2] comes with an addon which extracts a 3D model from a series of frames. One of the planes of the 3D model was used to rectify the facade, making it ready for robust window detection.
Using this addon of the *FIT3D toolbox* [2] we calculated the motion between a series of frames in order to extract a point cloud of matching features. This point cloud is used to extract a plane. which is used to rectify the facade by a projection process. We now explain the steps in more detail.

Figure 9: Window regions

1.5.2 Method: 3D plane extraction

We started by taking a series of 6 consecutive consequent (steady zoom, lightning, etc. parameters) images of a scene.
The images are chronological and have sufficient overlap. We calculated the motion of the camera between the frames in a few steps First we extract about 25k SIFT features of each frame. Then we use SIFT descriptors to describe and match the features within the consecutive frames. Not all features will overlap or match in the frames therefor RANSAC is used to robustly remove the outliers. After this an *8-point algorithm* together with a voting mechanism is used to extract the camera motion. For details please read [2].
The frames where matched one by one which returns an estimation of the camera motion that is not accurate enough. Therefore a 3-frame match is done which gives more accurate results. Unfortunately this result comes

Figure 10: Original image

with a certain amount of reprojection error, this error is minimized using a numerical iterative method called *bundle adjustment*. The final result is a very accurate estimation of the camera motion.
The next step is to use this camera motion to obtain a set of 3D points (corresponding to the matching image features). This is achieved using *linear triangulation* method.
Next a RANSAC based plane fitter is used to accurately fit a plane through the 3D points. *todo example figure*

Efficient Projecting

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

It would be straight forward to rectify the full image. However this is computational very expensive as each pixel needs to be projected. To keep

Figure 11: Rectified image

the computational cost to a minimum we project only the necessary data. Since we are using Houghlines we project only the endpoints of the found Houghlines. (This is allowed because the projective transformation we apply preserves the straightness of the lines. Note that this means we apply the edge detection and Houghline extraction on the unrectified image.)

If h is the number of Houghlines, the number of projections is $2h$ When we rectify the full image the number of projection is w x h, where w,h are the width and height of the image. To give an indication, for the *Spil* dataset this means we apply 600 projections in stead of 1572864: a factor of almost 3k faster.

The Houghline endpoints are projected to the 3D plane we extracted in the same way as we explained in chapter **??**. To wrap up, we send rays from the camera center trough the houghline endpoints and calculated the intersection with the 3D plane. The result is a 3D point cloud where each point is labeled to their corresponding Houghline.

The next step is to transform the facade (and therefore the Houghlines) are seen upfront. Instead of transforming the facade we rotate and translate the

camera. This means the viewing direction (z-axis) of the camera needs to be equivalent to the normal of the facade plane. Lets denote the unit vector spanning the original viewing direction as z and the unit vector spanning the desired viewing direction/normal of the facade z'. We calculated a rotation matrix from the axis-angle representation.

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

1.5.3 Results

1.6 Method II: Histogram based approach

1.6.1 Introduction

From the previous section we saw that from a series of images, a 3D model of a building can be extracted. Furthermore we saw that using this 3D model the scene could be converted to a frontal view of a building, where a building wall appears orthogonal. This frontal view enables us to assume orthogonality and alignment of the windows. We exploit this properties to build a robust window detector. First we determine the alignment of the windows and then we label and group the areas that contain the windows.

1.6.2 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.

1.6.3 Method

The extraction of the windows is done in different steps, first we rectify the image as described in the previous section. Then the alignment of the

windows is determined, this is based on a histogram of the Houghlines'. We use this alignment to divide the image in window or not window regions. Finally these regions are classified and combined which gives us the windows.

Extract Window alignment

We introduce the concept alignment line. We define this as a horizontal or vertical line that aligns multiple windows. In Figure 16 we show the alignment lines as two groups, horizontal (red) and vertical (green) alignment lines. The combination of both groups give a grid of blocks that we classify as window or non-window areas.

How do we determine this alignment lines? We make use of the fact that among a horizontal alignment line a lot of horizontal Houghlines are present, see Figure 15. For the vertical alignment lines the number of vertical Houghlines is high, see green lines in Figure 15.

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

Next we calculate two histograms H(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 16.
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 16 , is a smooth *projection profile* which contains the right number of peaks. The peaks are located at the average positions of the window edges. Next step is to calculate the peak areas and after this the peak positions.
Before we find the peak positions we extract the peak *areas* by thresholding the function. To make the threshold invariant to the values, we set the

threshold to $0.5 \cdot$ max Peak. (This value works for most datasets but is a parameter that can be changed). Next we create a binary list of peaks P, P returns 1 for positions that are contained in a peak, i.e. are above the threshold, and 0 otherwise. We detect the peak areas by searching for the positions where P = 1 (where the function passes the threshold line). If we loop through the values of P we detect a peak-start on position s if $P(s-1), P(s) = 0, 1$ and a peak-end on e if $P(e-1), P(e) = 1, 0$. I.e. if P = 0011000011100, then two peaks are present. The first peak covers positions $(3, 4)$, the second peak covers $(9, 10, 11)$.

Having segmented the peak areas, the next step is to extract the peak positions. Each peak area has only one peak and, since we used an average smoothing filter, the shape of the peaks are often concave. Therefore we extract the peaks by locating the max of each peak area. These locations are used to draw the window alignment lines, they can be seen as dotted red lines and dotted green lines in Figure 16.

1.6.4 Improved window alignment

As you can see in Figure 21 a few window alignment lines are not found and a few lines are found at wrong locations. Lets explain the missing alignment lines. The right side of the window frame of the first 4 windows is occluded by the wall. This is because the original image is not a frontal image. The reflection of the window has about the same color as the bricks of the wall, this means that the edge detector doesn't find a strong edge on positions where the window frame is missing. This artefact can be seen in the edge image Figure 14, few or no edges are present, and at the low height of the peaks in Figure 21 at these positions. This means we have to find another way to detect the window alignment on these positions.

For the vertical alignment we only took vertical lines into account. In this method we examine the projection profile of the *horizontal* Houghlines projected on the X axis, X_h, Figure 22. On the positions of the desired vertical alignment lines there appears to be a big decrease or increase of X_h at the window frame. This is because on these positions a window containing (a large amount of horizontal lines) starts or end.
We detect these big decreases or increases by creating a new pseudo peak profile D that takes the absolute of the derivative of X_h, , Figure 22.

$$D = abs(X_h')$$

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

1.6.5 Fusing the window alignment methods

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

If a window alignment position is found by both methods, often the peaks are located very close to eachother, see Figure 23. If this is the case, the peaks should form a group. Sometimes the peaks indicate the same window alignment but have some disparity. This is often the case when horizontal lines stop at the inside of a window frame while the vertical edges are located at the outer side of the window frame (the disparity is exactly the size of the windowframe part). Yet, in other cases close peaks indicate different windows that are just hapen to be located closely. To apply a proper grouping of the peaks the challenge is to distinguish these two cases.
First we decrease the total number of found window alignment locations and make the result more robust by increasing the individual thresholds (from 0.5*max peak to 0.7*max peak). After this we groupe the peaks, we first take the average of the maximum window frame part and the minimum window distance, the maximum peak group distance G. Next we compare all peaks and if the distance between two peaks is lower than G, we discard the peak with the least evidence (lowest peak). The result, a set of unique peaks, can be seen in Figure 24.
Note that the advanced peak grouping is only required for the vertical alignment of the windows: The horizontal inter window distance is often big enough to not be mistaken by a window frame part.
The image is now divided in a new grid of blocks based on these alignment. The next challenge is to classify the blocks as window and non-window areas: the window classification, we developed two different methods for this.

Window classification, method I

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 Hough-

lines, Figure 15 (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.

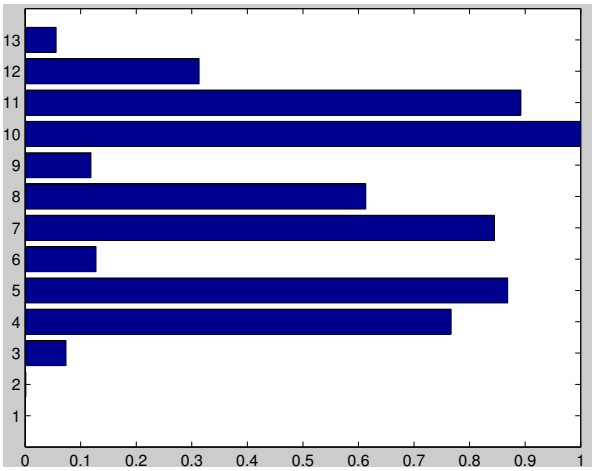

Figure 12: Normalized vertical Houghline pixel count of the blockrows (R)

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 12. Its easy to see that the high values, blockrow 4,5,7,8,10 and 11, correspond to the six window blockrows in Figure 17.
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 $k = 2$) as the classification procedure. This results in a set of Rows and Columns that are classified as window an non-window areas.
The next step is to determine the actual windows W. A block $w \in W$ that is crossed by R_j and C_k is classified as a window iff *k-means* classified both R_j and C_k as window areas. These are displayed in Figure 17 as green rectangles.
The last step is to group a set of windows that belong to each other. This is done by grouping adjacent positively classified blocks. These are displayed as red rectangles in Figure 17.
As the figure gives a binary representation of the windows it is not possible to see how certain a block is classification. To get insight about this we developed a measure of certainty function.

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

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

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

As you can see C is normalized, this is to ensure the value of the maximum certainty is exactly 1. The results can now be relatively interpreted, e.g. if the rectangle's $P = 0.5$ then the system nows for 50 % sure it is a window, compared to its best window ($P = 1$). And, as the normalization implies this, there is at least one window with $P = 1$.
The visualization of the measure of certainty is shown in Figure 13, the whiter the area the higher the measure of certainty.
The stripe patterns support that the classification process exist of individual row and column classification. The bright rows and columns indicate window blockrows and blockcolumns whereas the dark rows and columns indicate non-window areas. The area of window positions is particularly white, as it intersects a bright (positively classified) row and column. One can compare this result to the windows in the original image (Figure 11).

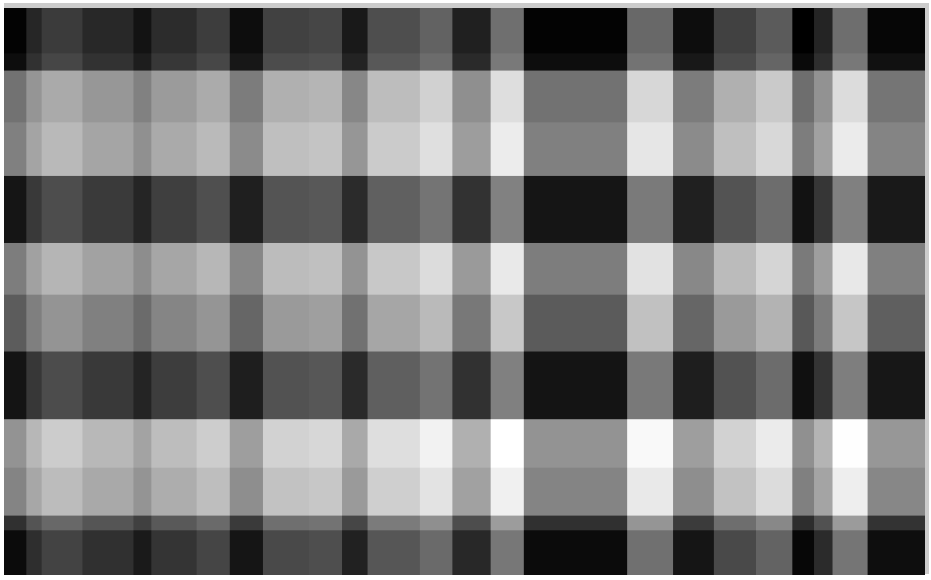

Figure 13: Window classification measure of certainty, white means high.

Window classification, method II

If we take a look at Figure 24 we see that X_h (the amount of horizontal Houghlines) has two shapes that repeat: At the location where a window is present X_h is concave whereas at non-window areas X_h is convex. This is because lots of horizontal lines are found at certain window positions (the window centers) and few are found at positions that are far away from these certain positions, which are the centers of non-window areas (that lie between windows). The shape type of X_h is used as a cue for our second window classifier.

This shape type of X_h is detected as follows: As in **improved window alignment**, the first step is to examine the derivative of X_h, blue line in Figure 24. We investigate the positions where $D = X_h'$ changes from sign, these are the peaks or valleys of X_h. X_h is concave at the sign changes from positive to negative (+,-) and X_h is convex if the sign changes (-,+).

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

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

For the sake of representation only blockcolumns are presented, the method for the blockrows is almost the same, the projection profiles are projected to the Y-axis.

1.6.6 Results

. We tested all methods on different datasets.

Figure 14: Result edge detection

Results of the Anne dataset:

TODO: test

Results of the Dirk dataset:

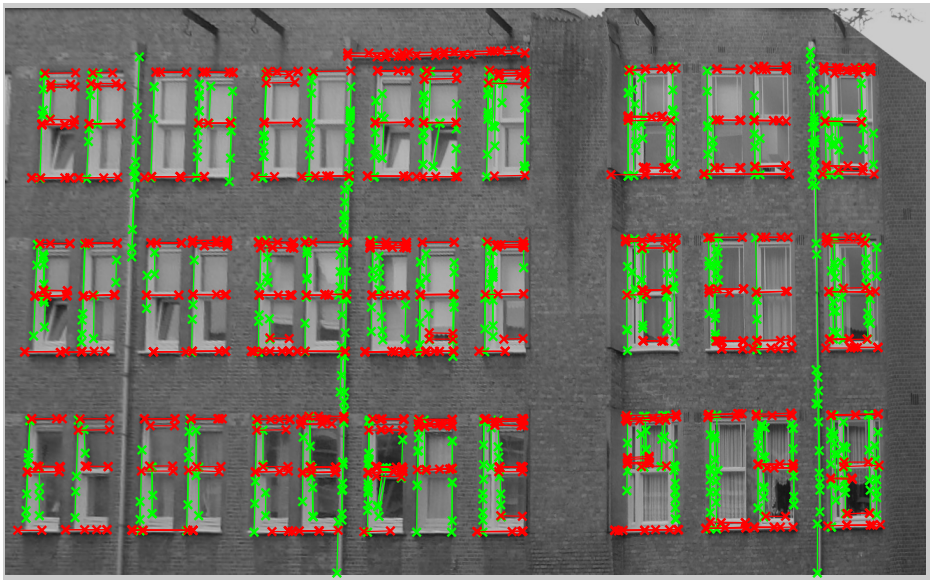

Figure 15: Houghlines with endpoints

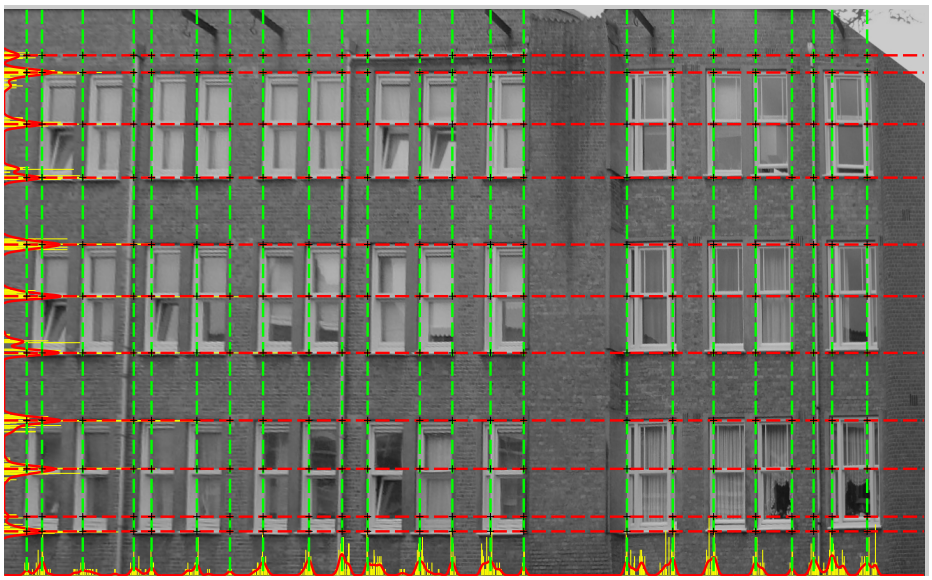

Figure 16: (smoothed) Histograms and window alignment lines

Facts of Figure 28
the amount of alignment lines is very large.

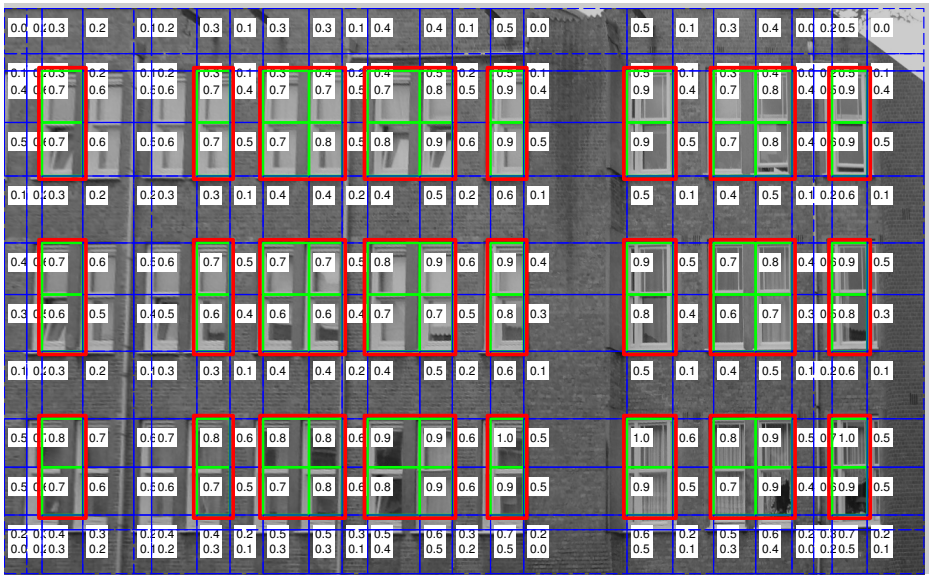

Figure 17: Classified windows

Facts of Figure 30

- 100 % of the alignment lines are positioned at window areas.
- 100 % of the windows are detected
- 44 windows are found
- 5 windows (11%) are false positive
- 39 windows (87%) are true positive

Grouping of positive classified windows:

- 39 windows are positive classified of which:
- 35 windows (90%) are grouped correct
- 4 windows (10%) are grouped incorrect

This first two window rows are classified as two groups but this should be one.

Figure 18: Original Image

1.6.7 Discussion

Interpretation of the results of the Anne dataset

Interpretation of the results of the Dirk dataset

It is easy to see in Figure **??** that there are too many alignment lines detected and that not every alignment line is placed correctly. This is mainly because the scene is partially aligned, the two front doors contain many vertical edges that are not aligned with the windows. Another cause of this artefact is that the bicycles on the left cause a large amount of found Hough lines, see 31. However, for the actual detection of the windows this is not a problem because after many close alignment lines are discarded by the peak fusing module the many subwindows are grouped to single windows, see red rectangles in Figure 30.

Figure 19: Window alignment lines and histograms

This first two window rows are classified as two groups but this should be one. Normally the peak merging step would handle this problem, but with this dataset we could not use a large *maximum peak distance* value because the windows in the image (especially the first and last columns) are very close.
todo compare classification methods and explain differences

Method II: Histogram based approach

We described a basic method of window alignment and build a significant improvement. If we compare the basic method (Figure 17) with the improved method (Figure **??**) we see that the improved method detected each window alignment with an maximum error of 3 pixels whereas the basic method has an error of 15 pixels. Its hard to say something about the classification method I because the window alignment lines had a large error.

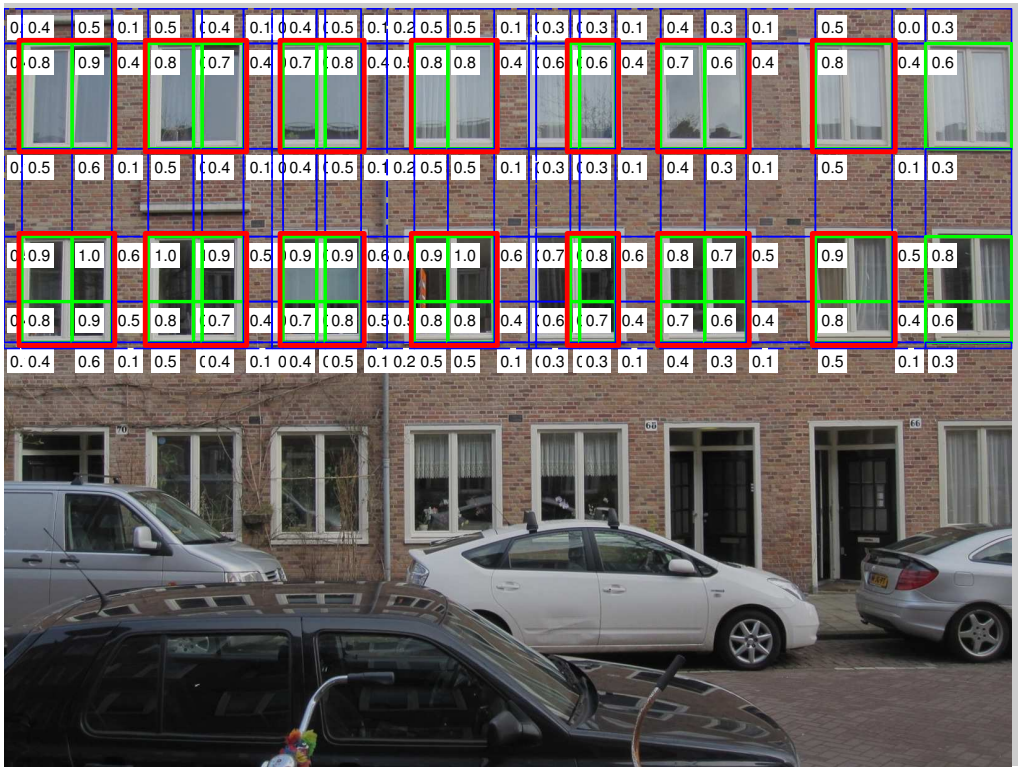

Figure 20: Classification method I: Grouped windows

The improved classification method Figure 25 however, classified 100 % of the windows correct.

Dirk dataset

It could be a drawback that the outcome of method I 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 on all datasets, fortunately it resulted in consistent classifications.
todo

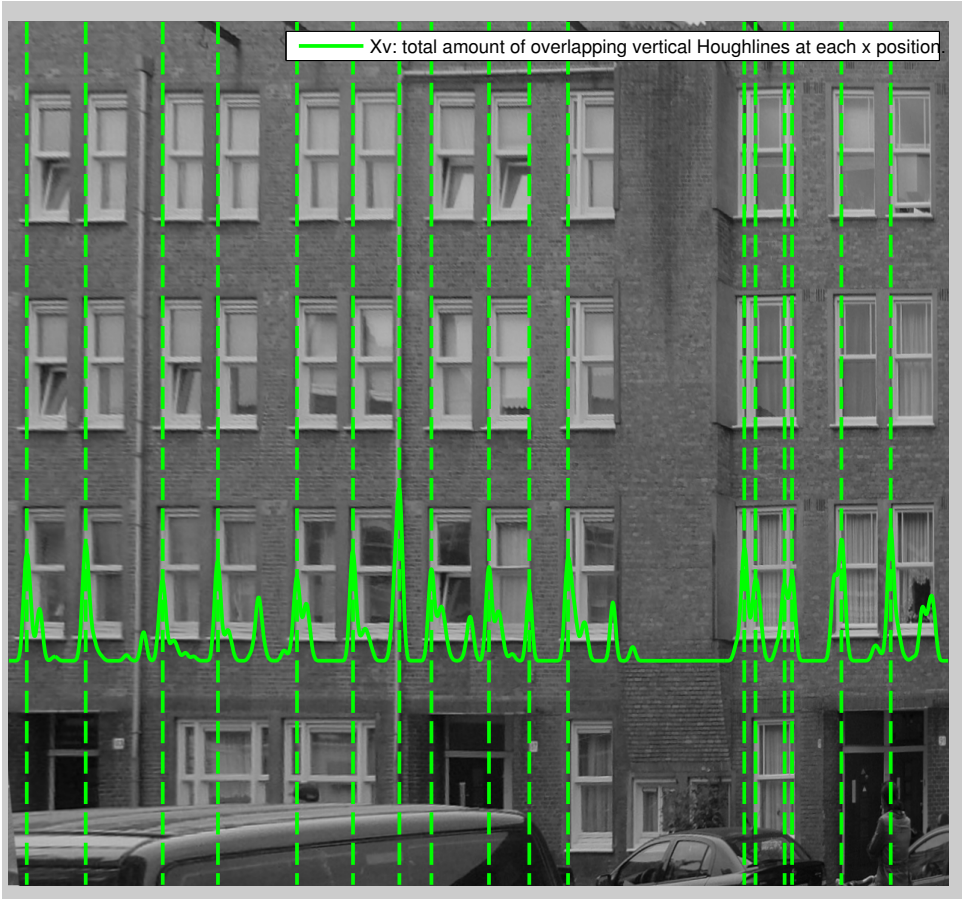

Figure 21: Dataset: Anne, Improved window alignment: Xh: total amount of overlapping horizontal Houghlines at each x position

Occlusion

If the image isn't the frontal view of the building wall we project the image see section ?This projection comes with some difficulties, occlusion. In a few cases an building wall extension (middle of figure 10) 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 full blockrows and blockcolumns.

Figure 22: Dataset: Anne, Improved window alignment: Using the shape of the horizontal Houghlines amount as window alignment

1.7 Conclusion

todo

1.8 Future research

1.8.1 Method I: Connected corner approach

It would be nice to group the connected corner to groups of sub windows. The big window that contains sub windows could be found by calculating the convex hull of the red areas in Figure 9. The sub windows could be found using a clustering algorithm that groups the connected corners to a

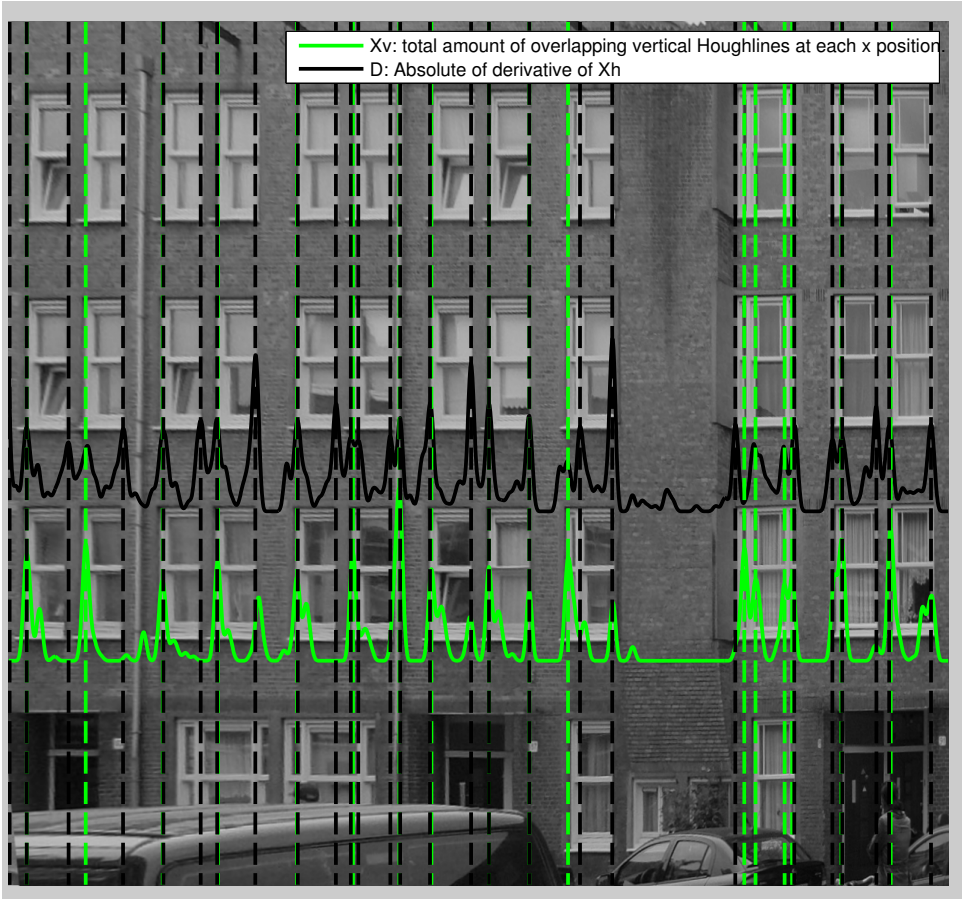

Figure 23: Dataset: Anne, Improved window alignment: Both methods combined

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 sub window.

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 com-

Figure 24: Dataset: Anne, Improved classification method: Concave and convex shapes on window/nonwindow locations

plete rectangles.
The later is difficult because the edges are often incomplete due to for example occlusion or the angle of viewing.

1.8.2 Method II: Histogram based approach

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

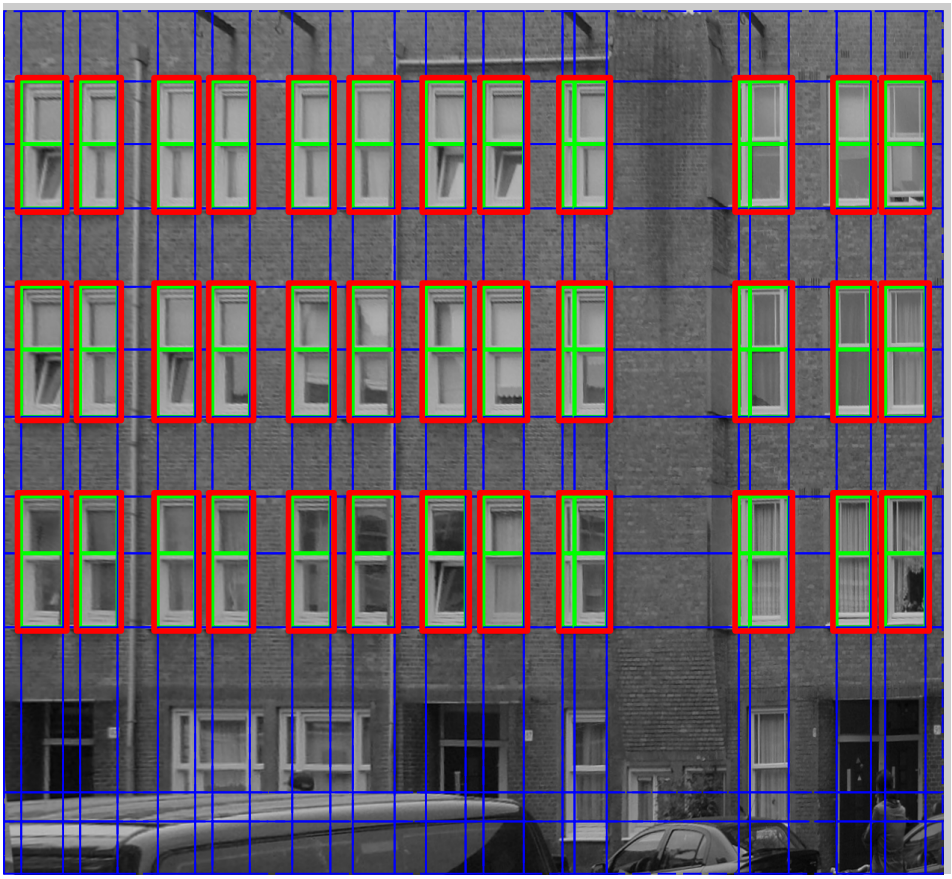

Figure 25: Dataset: Anne, Improved classification method: the extracted windows, red:the grouping

Window alignment: Peak grouping

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

Figure 26: Original, (unrectified) image,

Figure 27: Rectified image, the windows differ in size, type and shape and are partially unaligned

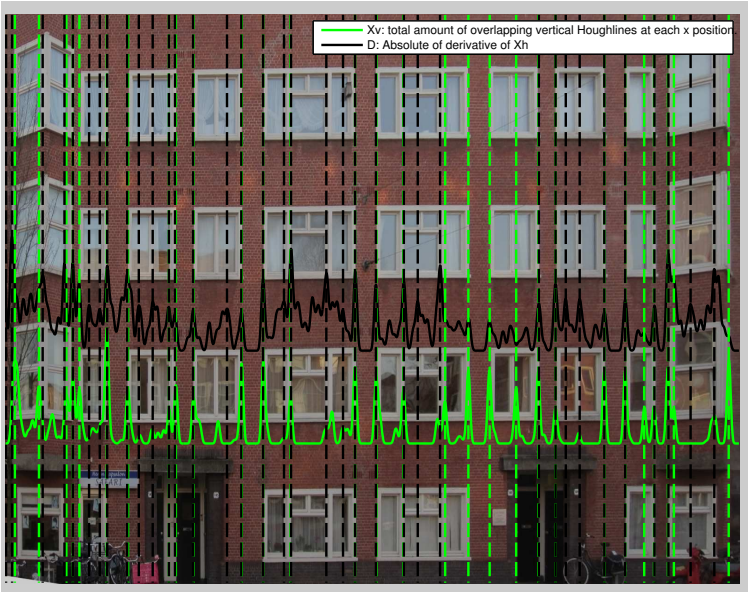

Figure 28: Vertical window alignment lines based on Xv and D

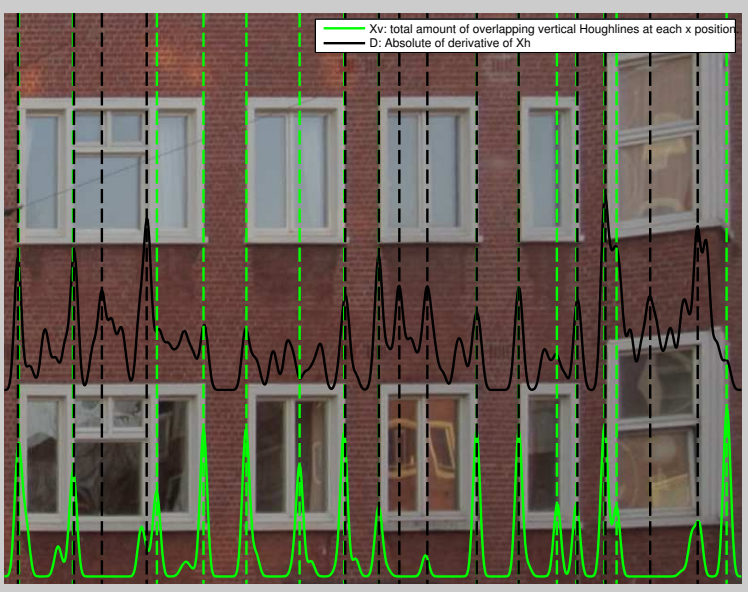

Figure 29: Vertical window alignment lines based on Xv and D zoomed

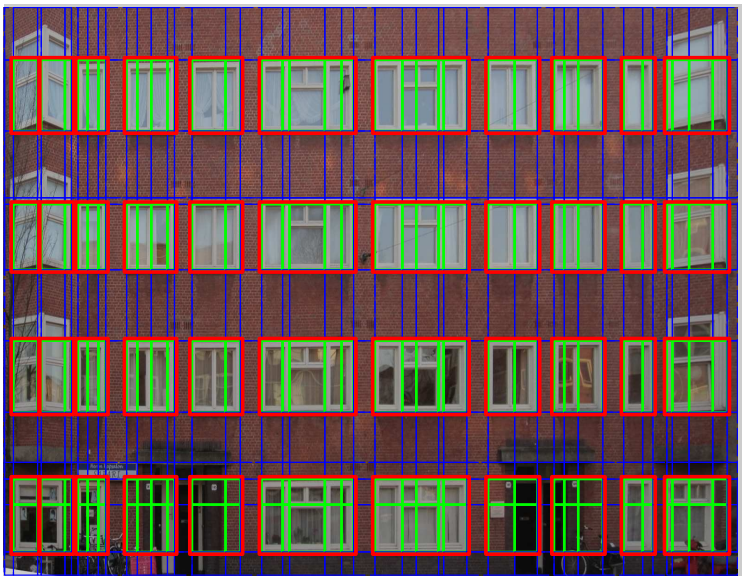

Figure 30: Improved classification method: Grouped windows

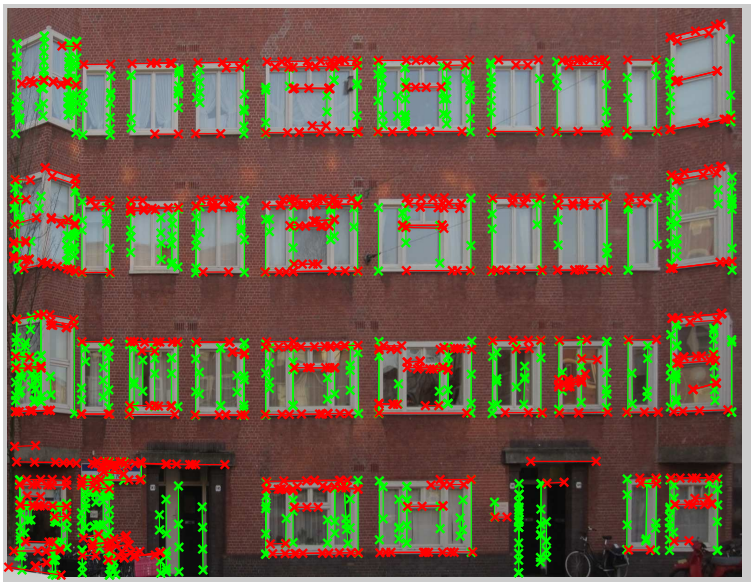

Figure 31: θ-constrained Hough transform

or outside the window could add important evidence, some methods of the window classification could be used to detect these values.

Window alignment refinement

To get more accurate result or to handle scenes with poor window alignment a refinement procedure could be applied. As mentioned in the related work, Lee et all [3] 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 recourses are available. For example if a car is driving and uses window detection for building recognition the refinement is disabled. But if the car is lowering speed the refinement procedure could be activated. Resulting in accurate building recognition which opens the door for augmented reality.

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 *multi scale Harris corner detector* could help an accurate alignment or refinement of the windows.

References

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

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.