3D reconstruction and semantic annotation of urban scenes

T. Kostelijk
mailtjerk@gmail.com

May 27, 2012

Contents

Todo list

1 comments/questions

1.1 questions

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

2 Skyline detection

2.1 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. [3],[8],[4]).
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.

2.1.1 Related work

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

Detection of dust devils and clouds on Mars

In [3], mars Exploration Rovers are used to detect clouds and dust devils on Mars. Their approach is to first identify the sky and then determine if there are clouds in the region segmented as sky. The sky is detected by an innovative algorithm that consists of three steps. First they place seeds in a sliding window whenever the homogeneity of the window is high. Then they grow 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 **??** are created because the building has walls that have straight lines, an assumption is made about partially straight lines.

Planetary Rover localization

Cozman et al. [4] use skyline detection in planetary rovers to estimate their location. To recover the rover's position they match image structures with a given map of the landscape (hills, roads, etc) and align both images. The matching process was first based on feature matching. In an improved version the matching process was done by searching for correspondences among dense structures in the image and on the given map, so called signal based matching.
The advantage of their algorithm is the simplicity and effectiveness, this could make their algorithm suitable for this project. A big drawback is that they prefer speed over accuracy. To increase accuracy, the detector is part of an interactive system where an operator refines the skyline. For our application the skyline detector must operate without any user interaction. Furthermore it has to be robust and accurate because it provides a basis for the extraction of straight lines which offer an estimation of the buildingwall heights.
We decided to use the Rover method [4] as a basis and build a custom algorithm with higher accuracy on top of that. This is explained in the next section.

2.2 Method

2.2.1 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 [4]. Instead of using the sharpest edge we take the most upper sharp edge and classify this edge as the skyline.

Top sharp edge assumption
The first sharp edge (seen from top to bottom) in the image represents the skyline.

2.2.2 Related algorithm

To put our work in context, we first describe a related skyline detection algorithm as presented in [4].

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.

2.2.3 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 [4] 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 [2] 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

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.

increased, this makes sharp edges stand out more. Secondly the image undertakes an extra Gaussian blur, this removes a large part of the noise. Note that depending on the edge detector type this could mean that the image is blurred twice.

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

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

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

2.3 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.

14cm14cm

14cm14cm

2.4 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.

2.5 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)

TODO: Zorgen dat paragraphs ook 3.5.3.1 nummering krijgt

3 Window detection

3.1 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.

3.2 Application examples

3d City models

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

Historical buildings documentation and deformation analysis

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

Interactive 3d models

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

Figure 4: 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.

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

3.3 State of art window detection

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

3.3.1 Alternative approaches on window detection

Muller et all [7] 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

(a) original texture (b) reconstruction

Figure 5: Results of Muller et all

of scenes the system can handle. Furthermore they match template window objects which they predefine, this constraints the variety of windowtypes that could be matched. At last they use expensive algorithms that make it impossible for the system to run in real time.

Using a thermal camera, Sirmacek [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*: 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.

TODO: research question

To investigate this, we developed two methods on which one method doesn't require repeating windows nor aligned windows. One of the main targets of our research is a low requirements on the input data. First our system doesn't need a large annotated dataset. Furthermore we took the images with a mobile phone and decided to extract the windows from the image space only, this makes us independent of additional expensive data like heat or laser range images. It doesn't mean the previous work on window detection using laser or heat images isn't of good use. Instead we learned a lot from the previous research as they have to match the laser or heat data to the real image space. This matching process involves a description (semantical annotation) of the facade. Let's explain a method of that kind and discuss other approaches that are more similar to our approach.

3.3.2 Similar approaches

Pu and Vosselman [9] 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 [9] 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 [10] developed a window detector that is build on the primary work of Lee and Nevatia [6] (which is discussed next). In order to make clear orthogonal projections they rectify the facade. To determine the alignment of the windows the edges are projected into their orthogonal direction. For example the horizontal edges are projected in the vertical direction to to

establish the vertical division of the windows.
A function is developed that counts the amount of projected edges on each

Figure 6: Projection profiles of Lee and Nevatias work

location, this is the *Projection profile*, see Figure 6. A threshold is applied on the projection profile to indicate the window boundaries that is used for the window alignment.

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

As in the work of Recky et all [10] Lee et all [6] 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.

3.3.3 Our work in context

The state of the art window alignment procedure in [10] and [6] 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 [10] and [6] 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.

3.4 Method I: Connected corner approach

3.4.1 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.

3.4.2 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 7 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.

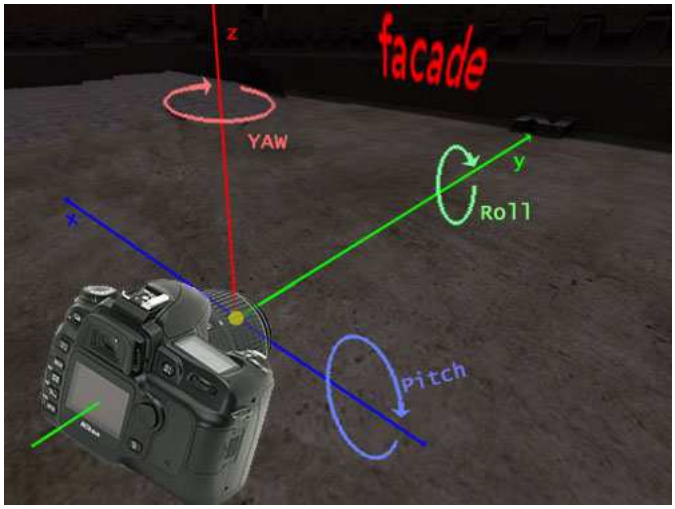

Figure 7: Pitch roll and yaw of the camera

The results of the edge detection and the Hough transform of two images can be seen in Figure 9 and 10.

TODO: 2

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

3.4.3 Connected corners extraction

As windows contain complex structures the amount of horizontal and vertical Houghlines is large at these locations. A horizontal and a vertical line are often connected in a corner of a window. In this approach we pair up these horizontal and vertical lines to determine *connected corners* that indicate a window.
Often a connected corner contains a small gap or an extension which we tolerate, these cases are illustrated in Figure 8 in the top row. A horizontal

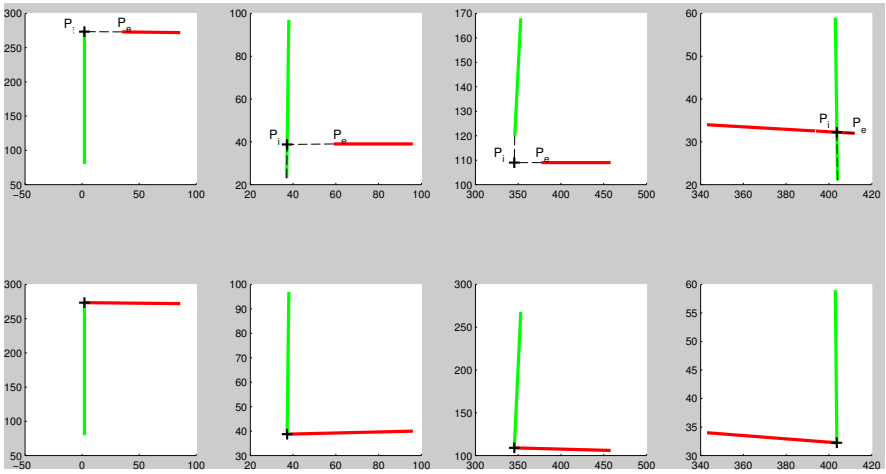

Figure 8: First row: different type of connected corner candidates. Second row: the result the clean connected corner

gap, a vertical and horizontal gap and a vertical elongation. The cleaned up corners are given in the bottom row. When the horizontal and vertical lines intersect, the gap distance is $D = 0$. When the lines do not intersect, the distance D between the intersection point P_i and the endpoint P_e of the line is measured $D = ||P_i - P_e||$, this is illustrated as dotted lines in Figure 8. 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 8. In Figure 8(I) the horizontal line is extended. Figure 8(II) shows that the vertical line is trimmed. In Figure 8(III) both lines are extended. At last, Figure 8(IV) shows how both lines are trimmed.

3.4.4 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 12, this result is discussed in section 3.4.5.

3.4.5 Results

.

TODO: 2

more datasets..

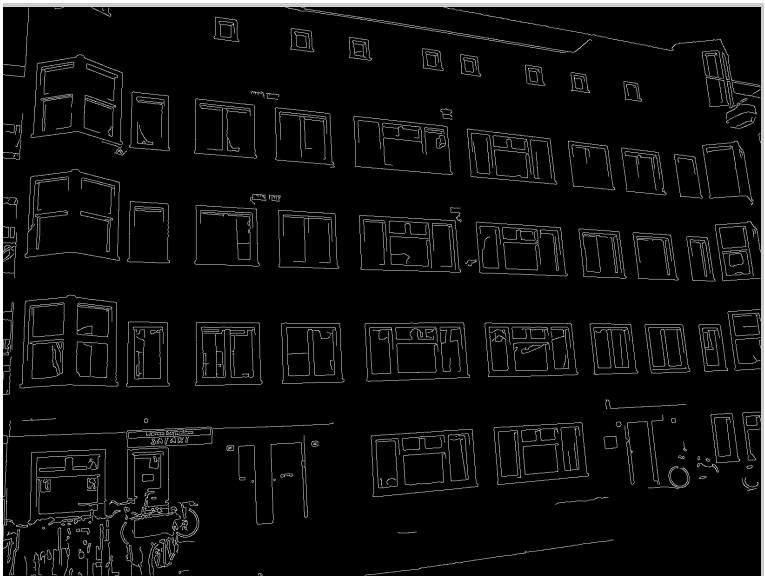

Figure 9: Edge detection

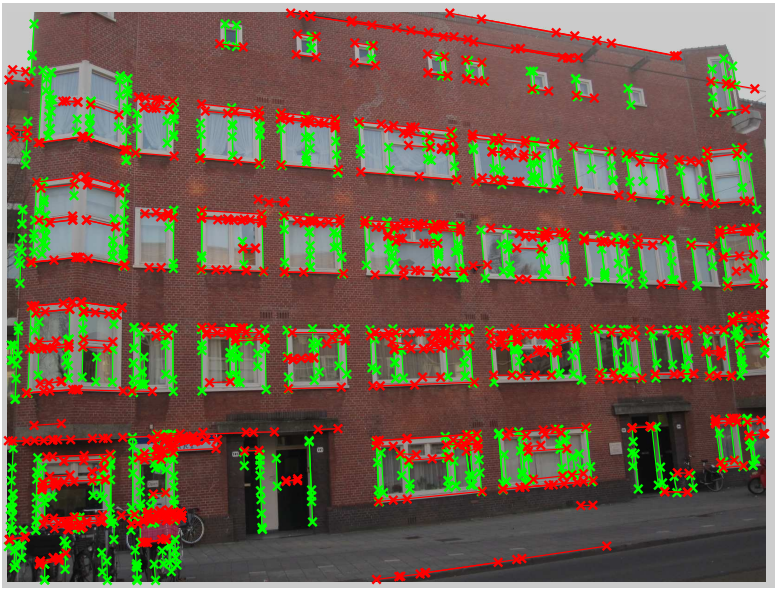

Figure 10: Result of θ constrained Hough transform

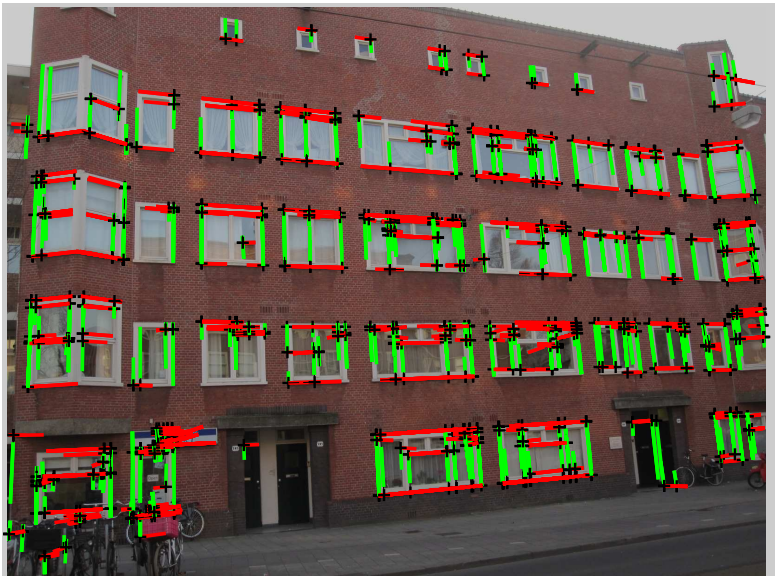

Figure 11: Found connected corners

Figure 12: Window regions

Figure 12 displays the result of the connected corner aproach on an unrectified scene. It contains 110 windows of which are 109 detected, this is 99%. Furthermore there are some False Positive areas, this is about 3 %. Sometimes a window is not detected, for example the window on the right top isn't detected, this is because its smaller then the minimum window width.

3.4.6 Conclusion

As can be concluded from the results, the connected corner method is suitable for, and robust to, scenes with a variation in window sizes and types. This makes the connected corner approach suitable for a wide range of window scenes where no or few prior information about the windows is known. Furthermore as no image rectification is required, the system has a low input requirement as it is independent of calibrated input data and 3d information about the building.

3.4.7 Future research

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

Furthermore the next step in this study would be to group the connected corners to windows and sub windows. The big window that contains sub windows could be found by calculating the convex hull of the red areas in Figure 12. The sub windows could be found using a clustering algorithm that groups the connected corners to a window. For this method it would be useful to assume the window size as this correlates directly to the inter-cluster distance. It would also be nice to incorporate not only the center of the connected corner as a parameter of the cluster space but also the length and position of the of the connected corners' horizontal and vertical line parts. The inter cluster distance and the number of grouped connected corner could form a good source for the certainty of the sub window.

3.5 Facade rectification

3.5.1 Introduction

In order to apply our second method of window detection (3.7), we need the windows on the facade to be orthogonal and aligned. Therefore we rectify the facade, which can be achieved in a manual or automatic way.

The simple rectification method uses point to point correspondences. This requires annotation of the corner points of the facade that are mapped with the corners of a rectangle. This mapping is used to calculate a transformation matrix. The downside of this method is that it isn't very accurate as it doesn't take the width and height ration of the facade into account. It only uses four point to point correspondences and 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 .

The second method involves the extraction of a 3D plane of the facade. This method is more complex and gives more accurate results. It involves a comprehensive process and a large amount of research is done in this area. Given the related work, and our focus on the annotational part of facade interpretation, we used existing software to apply the window rectification. I. Estebans *FIT3D toolbox* [5] 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* [5] 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.

3.5.2 3D plane extraction

We started by taking a series of 7 consecutive images (steady zoom, lightning, etc. parameters) of a scene. The images don't need to be chronological but need to have sufficient overlap.
We calculated the motion of the camera between the frames in a few steps First we extract about 25k SIFT features of each frame. Then we use SIFT descriptors to describe and match the features within the consecutive frames. Not all features will overlap or match in the frames therefor RANSAC is used to robustly remove the outliers. After this an *8-point algorithm* together

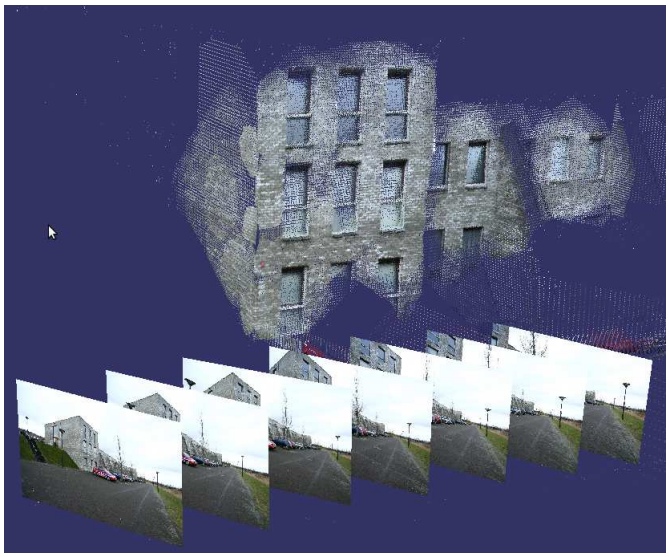

Figure 13: Example series of 7 consecutive frames, (dataset: FIT3D toolbox[5])

with a voting mechanism is used to extract the camera motion. For details please read [5].
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 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*

3.5.3 Efficient Projecting

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

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

putational very expensive as each pixel needs to be projected. To keep the computational cost to a minimum we project only the necessary data. Since we are using Houghlines we project only the coordinates of the endpoints of the found Houghlines. (This is allowed because the projective transformation we apply preserves the straightness of the lines. Note that this means we apply the edge detection and Houghline extraction on the unrectified image.)

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

The Houghline endpoints are projected to the 3D plane 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 its corresponding Houghline.

The next step is to transform the facade (and therefore the Houghlines) to a frontal view. Instead of transforming the facade we rotate and translate the camera. This means the viewing direction (z-axis) of the camera needs to be equivalent to the normal of the facade plane. Lets denote the unit vector spanning the original viewing direction as z and the unit vector spanning the desired viewing direction/normal of the facade z'. We calculated a rotation matrix from the axis-angle representation.

This presentation uses a unit vector u indicating the direction of a directed axis, and an angle describing the magnitude of the rotation about this axis, see **??**. 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 from a(n almost) frontal view (the facade is almost rectified) α is low. α and u are used to create a rotation matrix R which is applied to the 3D point cloud. The result is a set of rectified 3D points that are grouped to their Houghlines.

3.5.4 Results

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

Figure 14: Dataset: Anne1, Original, (unrectified) image

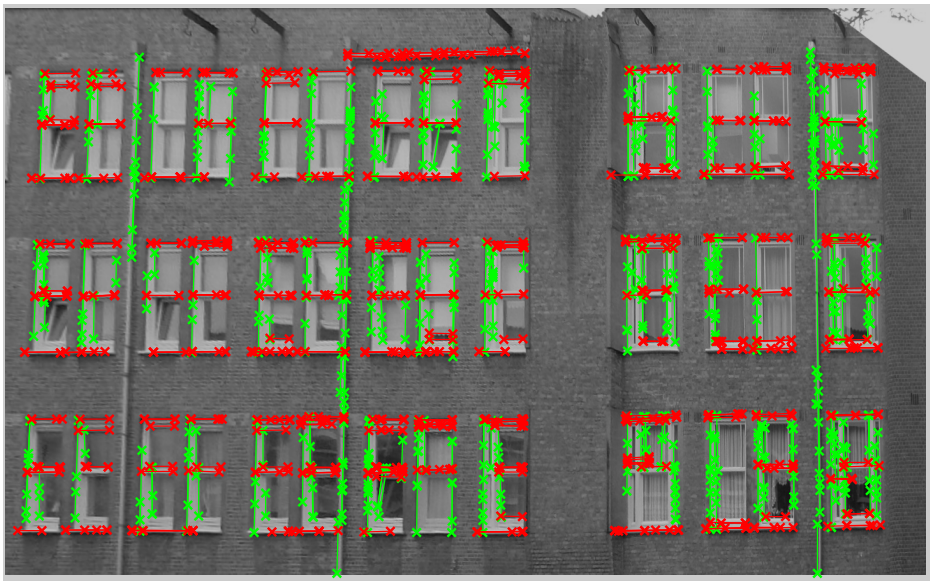

Figure 15: Dataset: Anne1, Projected Houghlines on rectified image

Figure 16: Dataset: Dirk, Original, (unrectified) image,

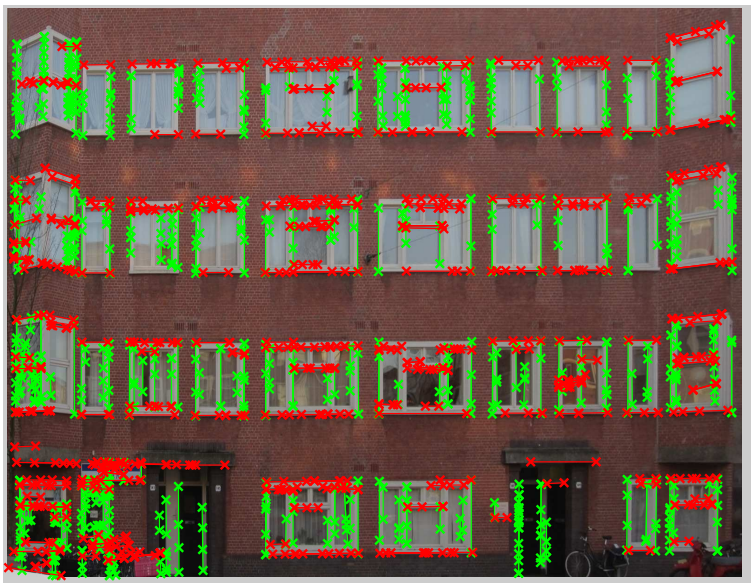

Figure 17: Dataset: Dirk, Projected Houghlines on rectified image

TODO: Met iemand overleggen wat betere volgorde is

3.6 Datasets

To avoid overfitting we used multiple datasets that have a large propertie variety. Furthermore every dataset has a challenge/handicap. We used three datasets that are recorded in the suburban area 'de Baarsjes' of Amsterdam. All images have a original resolution of 3072x2304 px and are downscaled (using trilinear interpolation) to 1280x1024 px.

3.6.1 Anne1

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

Figure 18: Dataset: Anne1, Rectified image

3.6.2 Anne2

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

Figure 19: Dataset: Anne2

3.6.3 Dirk

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

Figure 20: Dataset: Dirk

TODO: 2

tabel van datasets en handicaps!

3.7 Method II: Histogram based approach

3.7.1 Introduction

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

Situation and assumptions

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

3.7.2 Extracting the window alignment

Regular window alignment

TODO: 2

Schematje We introduce the concept alignment line. We define this as a horizontal or vertical line that aligns multiple windows. In Figure 21 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 w x h is the dimension of the image. The histograms are presented as small yellow bars in Figure 21.
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 21 , 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 21.

Alternative window alignment

As you can see in Figure 22 a few window alignment lines are not found and a few lines are found at wrong locations. The right side of the window frame of the first 4 columns of windows is not found. This means we have to find another way to detect the window alignment on these positions.

For the vertical alignment we only took vertical lines into account. In this method we examine the projection profile of the *horizontal* Houghlines projected on the X axis, X_h, Figure 23. 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 23.

$$D = abs(X_h')$$

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

Fusing the window alignment methods

We have presented two window alignment methods, next we fuse the methods to gain a robust window alignment. The 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 24. If this is the case we want to fuse the results by grouping the peaks. Most of the times the peaks indicate the same window alignment but have some disparity. This is often the case when horizontal lines stop at the *inside* of a window frame while the vertical edges are located at the *outer* side of the window frame (this is supported by the fact that the disparity is often the size of the windowframe part). Yet, in other cases close peaks indicate different windows that are just 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 by increasing the individual thresholds (from 0.3*max peak to 0.5*max peak). Note that this has a positive side effect that the peaks that are found are more certain. After this we group the peaks as follows:
First we calculate the average of the maximum window frame part and the minimum window distance and call this the maximum peak group distance G. Next we compare all peaks and if the distance between two peaks is lower than G, we discard the peak with the least evidence (lowest peak). The result, a set of unique peaks, can be seen in Figure 30.

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

TODO: 2

peak merging result misses for 1 dataset?

TODO: 2

peak merging result plaatje met en zonder merging onder elkaar

Results

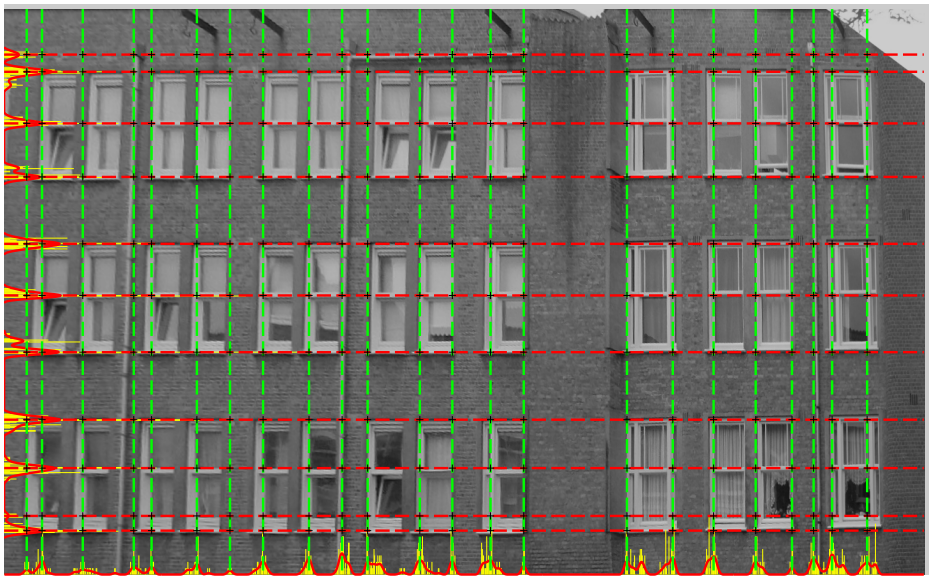

Figure 21: Dataset: Anne1, Regular window alignment (paralel projection): Based on a smoothed histogram (red line) that displays the amount of overlapping Houghlines, for the column division the horizontal houghlines are counted (at each y position), for the row division the vertical houghlines are counted (at each x position)

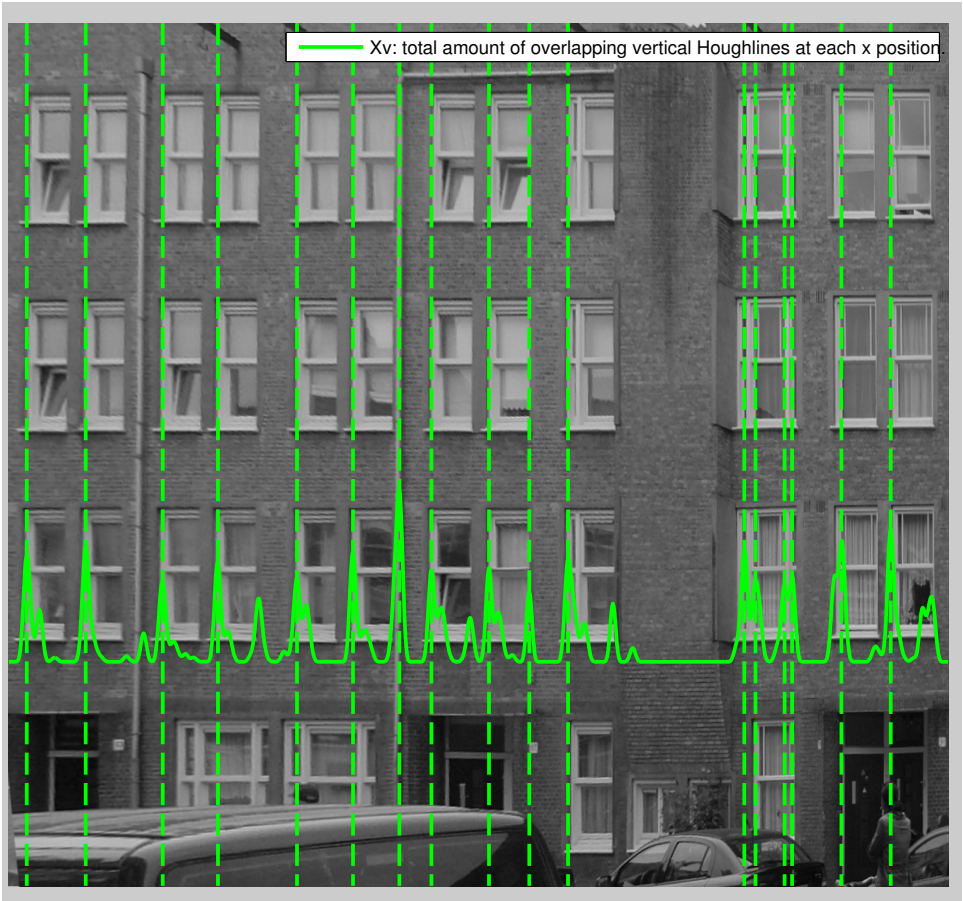

Figure 22: Dataset: Anne2, Regular window alignment: Based on a histogram that displays amount of overlapping vertical Houghlines at each x position

Figure 23: Dataset: Anne2, Alternative window alignment (orthogonal projection): Based on the shape of the smoothed histogram function. For the column division, the number of *horizontal* Houghlines is counted (Note that this is the orthogonal oposite of the regular window alignment method). Peaks (that represent a big decrease or increase of the histogram function) are used for the alignment.

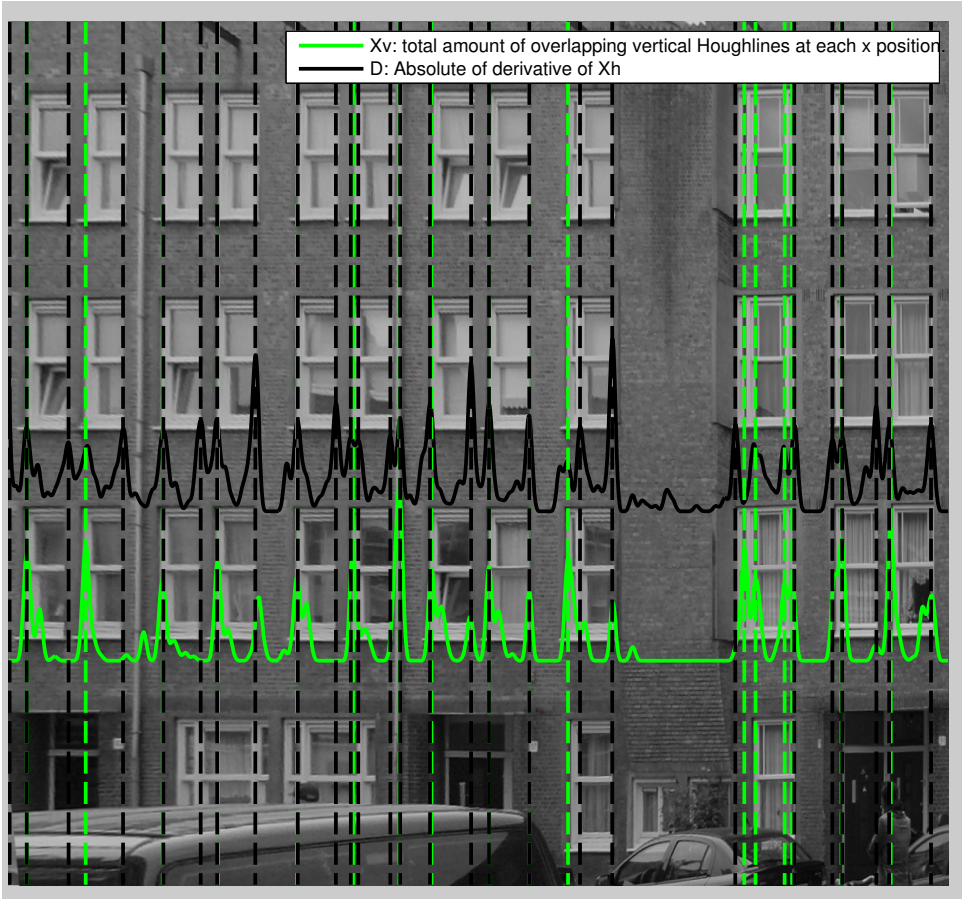

Figure 24: Dataset: Anne2, Regular+Alternative window alignment combined

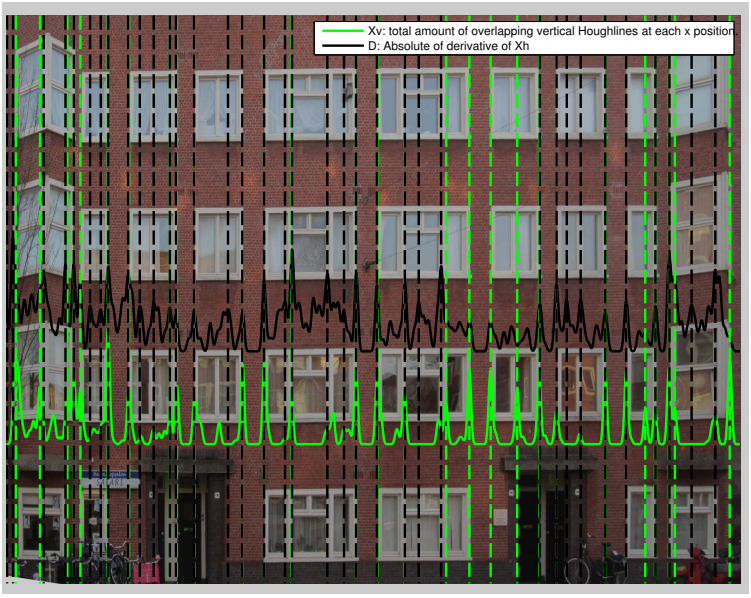

Figure 25: Dataset: Dirk, Regular+Alternative window alignment combined

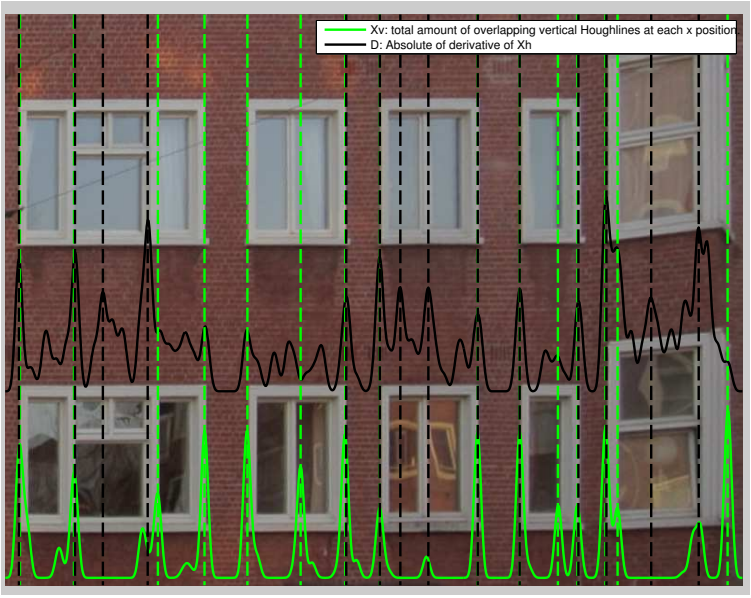

Figure 26: Dataset: Dirk, Regular+Alternative window alignment combined, zoomed

Discussion

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

Discussion:Many alignment lines

In Figure 25, too many alignment lines detected and not every alignment line is placed correctly. This is mainly because the windows are partially aligned: the two front doors contain many vertical edges that are not aligned with the upper windows therefore a double alignment is found (doors and windows). Another cause of this artefact is that the bicycles on the left (see Figure 36) cause a large amount of found Hough lines which implies a (false) alignment line. Although the causes are clear, we don't have to worrie about this additional window alignment lines that lie inside a window area as adjacent window areas are grouped by the classification module.
Fortuntatelly no alignment lines are found at non-window areas.

Discussion:Fusing the methods

If we evaluate the independent window detection results on the Anne2 dataset we see (Figure 23, 22) that in both datasets neither the regular window alignment method nor the improved window alignment detect 100% of the window alignment lines. The effect of fusing the methods is high as, after fusing, 100% of the window alignment is found on both datasets: each window alignment line is detected by at least one method. This is basically explained by the fact that both methods are based on a different Houghline direction. We now discuss the interesting cases, where only one method succeeds in detecting the window alignment.

If we take a look at the regular window alignment in Figure 22 we see that for the first 4 window columns the right side of the window is not detected. This is because the original (unrectified) image (Figure 14) is not a frontal image. This makes building wall extensions (middle of Figure 14) or drainpipes occlude parts of the window. In this case the windows are countersunked into the wall making the building wall itself ocluding the right window frame (Figure 22). The color of the reflection of the window is very similar to the bricks and because the window and wall are not seperated by the window frame, the edge detector doesn't find a strong edge. This means that on all positions where the window frame is missing, no vertical Houghlines will

be detected and (as the regular window aligment is based on the amount of vertical Houghlines) no window alignment will be found. This artefact can be studied in the edge image Figure 35: few or no edges are present, and at the low height of the peaks in Figure 22 at these positions.

However, the alternative window alignment does find a window alignment on this positions. This is because the method is based on the opposite Houghline direction : for the vertical window alignment the horizontal Houghline direction is taken into account. This occlusion artefact has no effect on the horizontal Houghlines which makes the alternative window alignment method a strong alternative for the alignment of (partially) occluded windows.

In general, the alternative window alignment performs better then the regular window alignment method. This is because this method takes the horizontal window frame parts into account which is a prori stronger as there are more horizontal window parts then vertical window parts present. E.g. Figure 15 every window has (as it has two vertical subwindows) 3 horizontal window frame parts but only 2 vertical window parts). Furthermore the method is more robust because it relies on a higher level of histogram interpretation (by using the derivative). However, in a few cases alternative window alignment method is outperformed by the regular window alignment . For example, in Figure 26, the left side of the second window column is not detected. This is because this window type has no horizontal divisions. Only the top and bottom of the windowframe produce an edge, therefore the derivative of the amount of Houghlines will return a small peak on this position (which is too small to survive the threshold). The threshold is a priori hard to survive because it has a high value as it is determined by taking a fraction of the maximum peak which is located at the windows that do have horizontal divisions (for example the most left or most right window column)).

TODO: result1 drain pipe result niet erg ivm vlg stap

Conclusion

We can conclude that developing histogram functions of the amount of Houghlines is a strong approach towards determination of the window alignment as the results in this work and in previous work a very prommising.
We proposed two window detection methods and can conclude that the

alternative approach performs better because it is based on a horizontal window division and becasue it uses a higher level interpretation of the histogram function.
We showed that window alignment becomes a challenging task if the windows are partially occluded. One of the main solutions we proposed is to use multiple window detection approaches. To have a 100% detection rate one need to be certain that if one approach fails at least one other approach must succeed. We proposed the strong combination of two methods where the first method filled the gab of the second method and the other way arround.
We can conclude that a method that fills the gab of the occluded alignment locations must rely on Houghlines that lie in the orthogonal direction of the occlusion. In our case the occluded vertical window parts are supported by a method that detects horizontal window parts.

Conclusion:Relative thresholding

Furthermore we discussed the implication of the use of different window types. Applying a threshold that is relative to a maximum peak doesn't work well on this window types that differ (in for example horizontal divisions) as window alignment lines are missing (Figure 26). This means that if we want to stick with the relative thresholding method, we have to assume a certain equality of the window types.

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

TODO: diagram maken window alignment -¿ peak merging -¿ etc

Future research

Future research:Determine window alignment of different window types

A solution of the missing window alignment lines on a scene with different window types (Figure 26) would be to decrease the threshold if multiple window types are found. One could design a measure of variety of the window types. This could be done by taking the variation of the derivative of the amount of Houghlines. This amount of variation will determine the amount of decrease in threshold. Let's explain this by two examples: If there is just a few variation the maximum peak is very representitive for a window so the threshold could be for example 0.7*max peak. However

if there is a large variation is found (which means multiple window types are present), the threshold should be lowered to 0.3*max peak to detect the hard window types (with few horizontal divisions). Another method would be to cluster the amount of Houghlines in n+1 values where n is the number of window types (the 1 is for the non-window area). Areas that transcent from a window area cluster to a non-window area cluster (or vice versa) are determined as the window alignment locations. For both methods the challenge is again to detect the window alignment with unknown window types but keep the number of false positives (e.g. a drain pipe) zero.

Future research:Peak grouping

We fused the result of the horizontal and vertical Houghlines and discarded peaks that where close. A more accurate result would be achieved if close peaks where averaged, the height of the peak could be used as a weight. We used a manual 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, or outside the window could add important evidence, some methods of the window classification could be used to detect these values.

Future research:Window alignment refinement

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

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

ment or refinement of the windows.

3.7.3 Basic window classification (based on line amount)

The image is now divided in a new grid of blocks based on these alignment. The next challenge is to classify the blocks as window and non-window areas: the window classification, we developed two different methods for this.
Instead of classifying each block independently, we classify full rows and columns of blocks as window or non-window areas. This approach results in a accurate classification as it combines a full blockrow and blockcolumn as evidence for a singular window.
The method exploits the fact that the windows are assumed to be aligned. A blockrow that contains windows will have a high amount of vertical Houghlines, Figure 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.

TODO: DIT PLAATJE eruit en mooie staafdiagrammen erin

If we examine the distribution of R and C, we see two clusters appear: one with high values (the blockrows/blockcolumns that contain windows) and one with low values (non window blockrows/blockcolumns). For a specific example we displayed the values of R in Figure 27. Its easy to see that the high values, blockrow 4,5,7,8,10 and 11, correspond to the six window blockrows in Figure 29.
How do we determine which value is classified as high? A straight forward approach would be to apply a threshold, for example 0.5 would work fine. However, as the variation of the values depend on (unknown) properties like the number of windows, window types etc., the threshold maybe classify

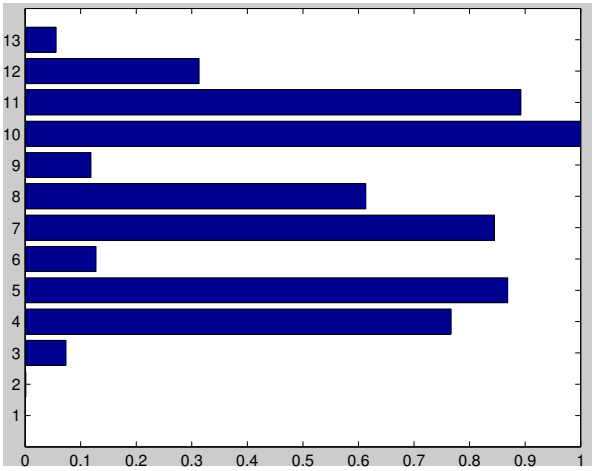

Figure 27: Classification values for window block rows representing the normalized vertical Houghline pixel count of (R)

insufficient in another scene. Hence working with the threshold wouldn't be robust.
Instead we use the fact that a blockrow is either filled with windows or not, hence there should always be two clusters. We use *k-means* clustering (with $k = 2$) as the classification procedure.

TODO: ref

This results in a set of Rows and Columns that are classified as window 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 29 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 29.
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 28, the whiter the area the higher the measure of certainty.

Results

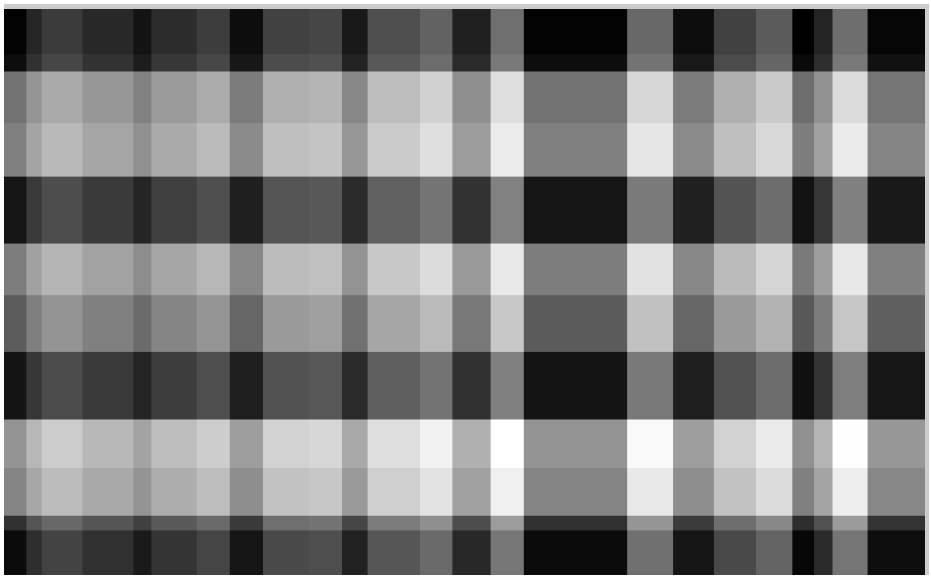

Figure 28: Dataset: Anne1, Basic window classification method:Measure of certainty, the whiter the area the higher the measure of certainty.

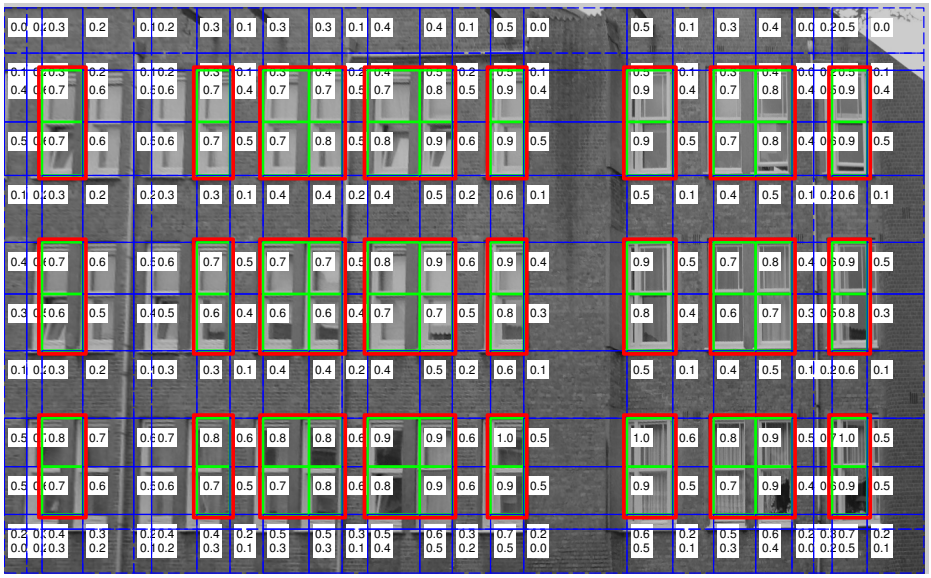

Figure 29: Dataset: Anne1, Basic window classification method: The extracted windows, red:the grouping

TODO: include results of binary classification

The bright rows and columns in Figure 28, indicate window blockrows and blockcolumns, whereas the dark rows and columns indicate non-window areas. The stripe patterns support that the classification process exist of individual row and column classification. The area of window positions is particularly white, as it intersects a bright (positively classified) row and column. One can compare this result to the windows in the original image (Figure 18).

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

Discussion

The outcome of this method is non-deterministic, as it depends on to the random initialization of the cluster centers. This means that our results could be correct by coincidence. To exclude this artefact, we ran the cluster algorithm 10 times on all datasets. Unfortunatelly for the Dirk dataset, 2 of 10 times it resulted in a bad result. This result can be found in Appendix Figures 37, 38, 39 and 40. This result can be explained as follows. The Dirk dataset has window types in the middle of that contain a horizontal subdivision while the others don't. The windows in the middle will cause k-means to drag the cluster center to a value that is too high making windows classified as non-window areas.

Future research

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

Conclusion

We can conclude that this certainty based classification works quitte good on this dataset. However, it requires a very well alignment of the windows which is a disadvantage. As it is based on the number of found Houghlines, the errors in the window alignment will propagate to the classification. This makes this classification method inappropriate for a system where one cannot fully rely on the window alignment.
This conclusion amplifies the need for a robust classification method (that is independent, (or at least less sensitive)) to window alignment errors.

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

TODO: introduce a little bit

TODO: say that anne2 dataset is more rectified

If we take a look at Figure 30 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 30. We investigate the positions where $D = X'_h$ changes from sign, these are the peaks or valleys of X_h. X_h is concave at the sign changes from positive to negative (+,-) and X_h is convex if the sign changes (-,+).

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

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

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

Results

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

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

Results:Dirk dataset

Figure 30: Dataset: Anne2, Improved window classification method: The red line shows concave shapes at window locations and convex shapes at nonwindow locations

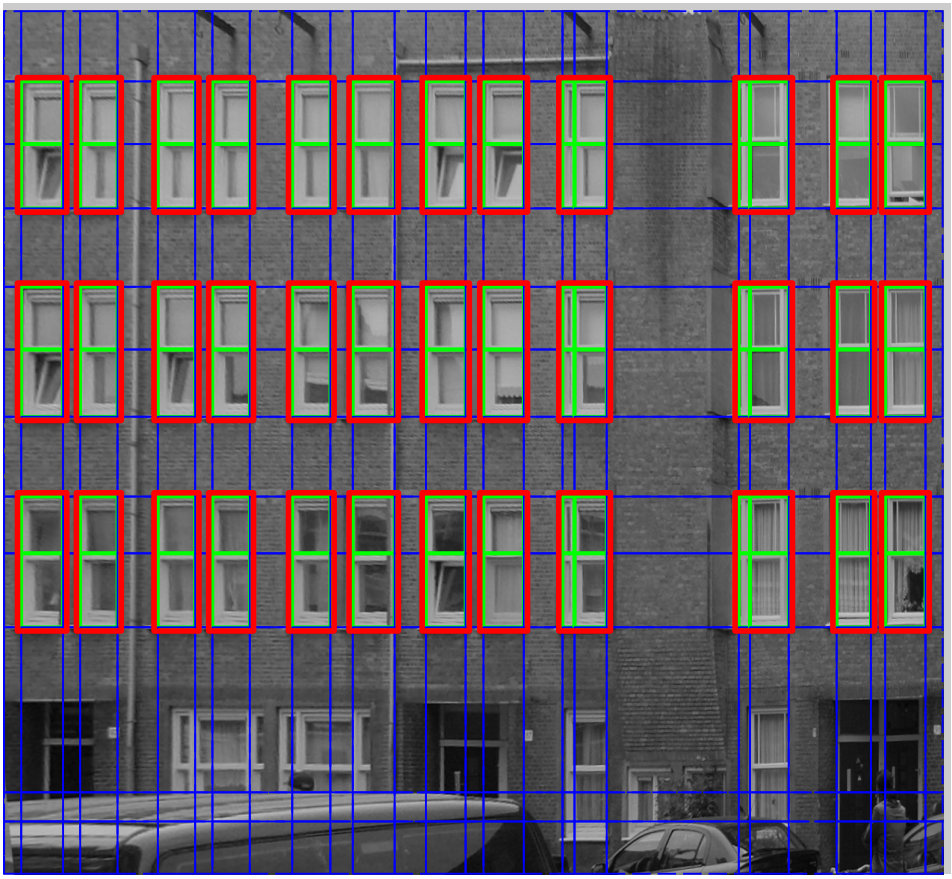

Figure 31: Dataset: Anne2, Improved window classification method: The extracted windows, red:the grouping

Although the scene contains a lot occlusion artefacts and suffers resolution loss, especially on the left side of the image, the results for the Anne2 dataset, a 100% detection and a 100% true positive rate are very well. This is mainly due the high interpretation level of the Histogram function. We used the fact that the histogram function has a very consistent pattern, at every window area its shape is convex and every non-window its shape is concave.

Results:Anne2 dataset

Figure 32: Dataset: Dirk, Rectified image of a realistic scene which is realistic as it contains light spots, bicycles. Note that the windows are partially aligned and the differ in size and type

TODO: IMPROVED result on Anne1 dataset, investigate which comibnations of dataset andmethods i miss

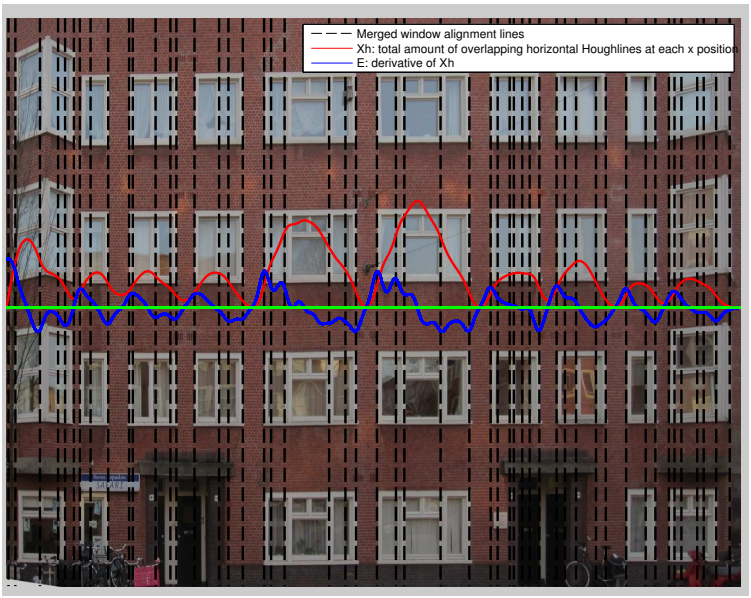

Figure 33: Dataset: Dirk, Improved window classification method: The red line shows concave shapes at window locations and convex shapes at nonwindow locations

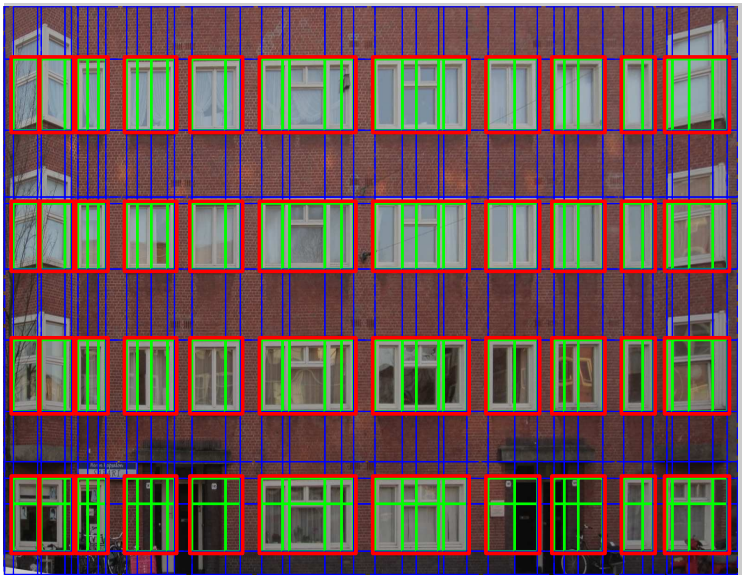

Figure 34: Dataset: Dirk, Improved window classification method: The extracted windows, red:the grouping

Table 2: Improved window classification results on Anne2 and Dirk dataset

Type	Anne2 dataset (Fig 31)	Dirk dataset (Fig 34)
Detection rate	100 %	100 %
True positive rate	100 %	87 %
False positive rate	0 %	11 %
False negative rate	0 %	9 %

Table 3: Grouping results on Anne2 and Dirk dataset

Type	Anne2 dataset (Fig 31)	Dirk dataset (Fig 34)
Grouped correct	100 %	90%
Grouped incorrect	0 %	10%

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

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

TODO: Interpretation of the results of the Anne1 dataset

TODO: compaire to other classification methods

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

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

TODO: Meer over lof spreken

We discussed in the section about window alignment that the classification module would handle the redundant window alignment lines. This can be seen in Figure 34 where the classification took advantage of the fact that all alignment lines are located at a boundary of a window or in a window. The peak fusing module discarded many close alignment lines and the residual redundant alignment lines create small green adjacent subwindows which where grouped to clear red big windows.

This grouping result is promissing as 90% of the grouping went well. This is a quitte good result given the scene contains a large variety in window shape, window size and window type. This result is mostly due an almost perfect window alignment and classification (as the grouping only groups adjacent positively classified subwindows). The first two window rows in Figure 34 however are classified as two groups but this should be one. Normally the peak merging step would handle this problem, but with this dataset we could not use a large *maximum peak distance* value because the windows in the image (especially the first and last columns) are very close. More on this in *Future research on window classification (3.7.4)*.

Conclusion

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

- Redundant lines found by the window alignment module cause no

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

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

Future research

TODO: put a opbouwende line in future work

Future work:Extensive evaluation methods

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

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

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

Future work:Grouping error minimalisation

Although the grouping module gave promising results it can be optimized. The grouping module now groups adjacent positively classified window areas. Some window in Figure areas are false negatively classified, see the left side of Figure 34. This is caused by a small area between the windows that is classified as a non-window area. This could be solved by by adding a minimum size constraint of a area to be threated as a non-window area. In this way small negatively classified areas cannot interupt the adjacent windows.

Performance measure for extreme viewing angles

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

TODO: compare classification methods and explain differences

3.8 Conclusion

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

TODO: Furthermore our algorithms work in real time.

4 Appendices

4.1 Figures

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

Figure 35: Dataset: Anne1, Result edge detection

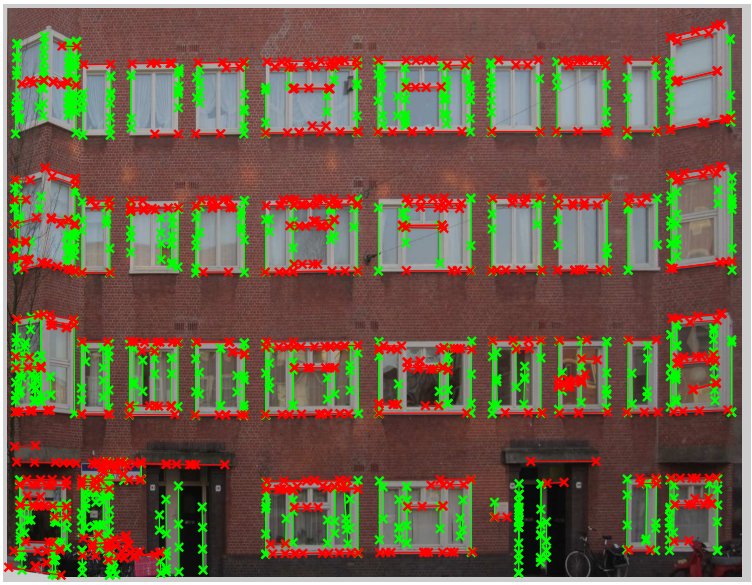

Figure 36: θ-constrained Hough transform

4.1.1 K-means bad luck

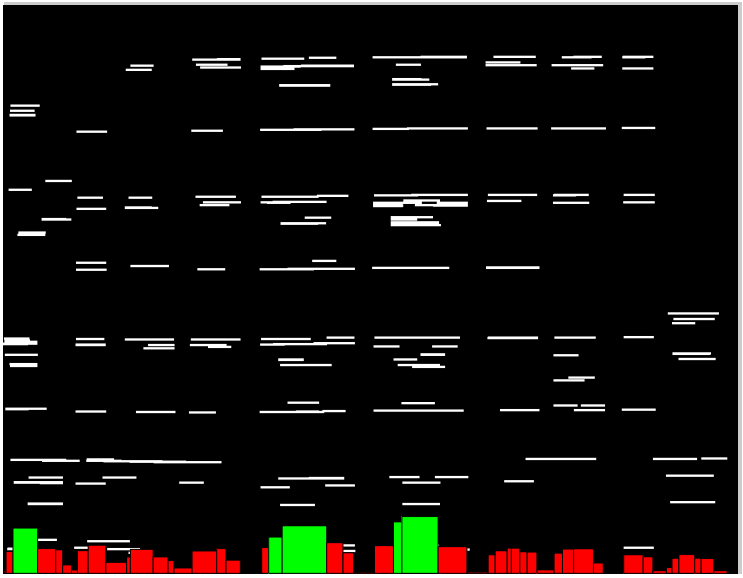

Figure 37: Dataset: Dirk, Horizontal Houghlines

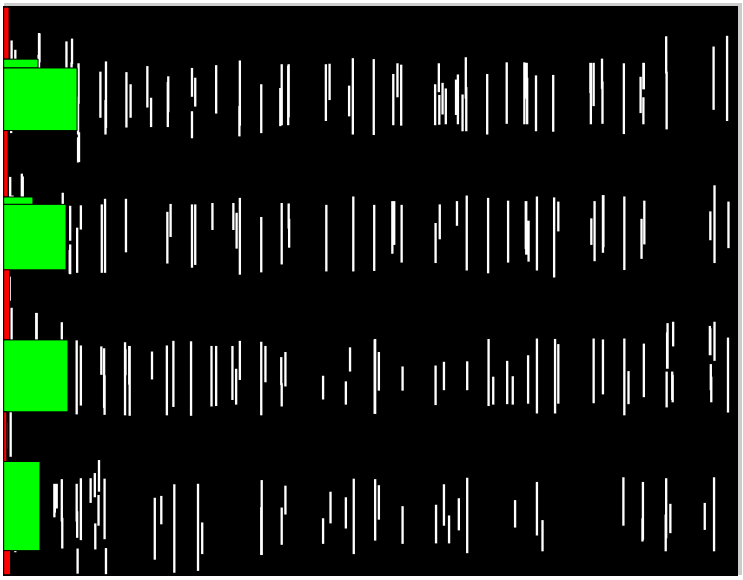

Figure 38: Dataset: Dirk, Vertical Houghlines

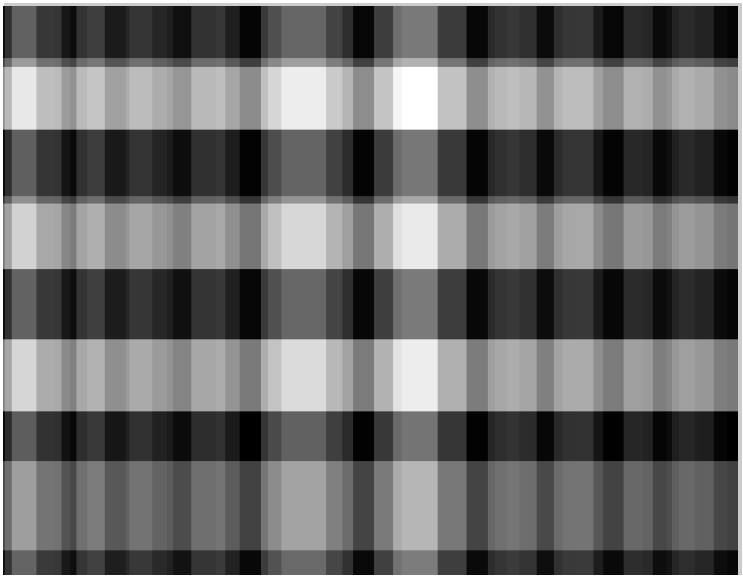

Figure 39: Dataset: Dirk, Basic window classification method:Measure of certainty, the whiter the area the higher the measure of certainty. The values of the window areas in the middle are relative large, this is due the horizontal subwindows.

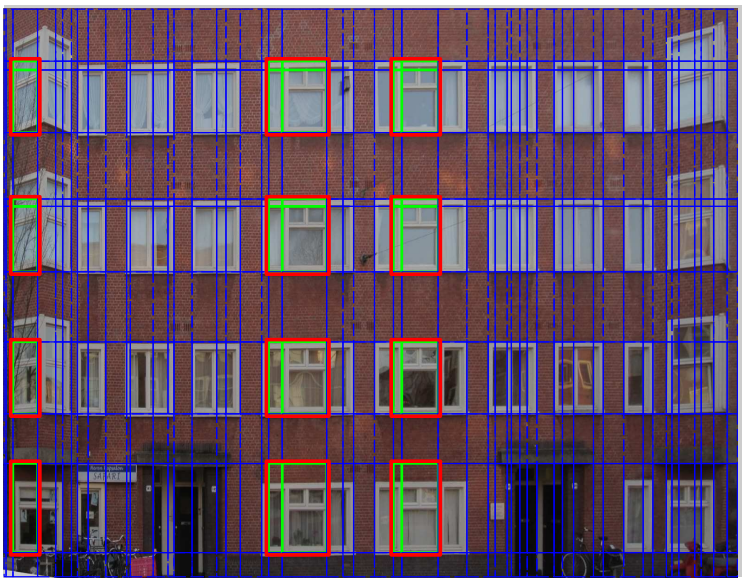

Figure 40: Dataset: Dirk, Extracted windows

References

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

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

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

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

[5] 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.