Optimizing 3D models from 2D images

T. Kostelijk
mailtjerk@gmail.com

December 16, 2010

Abstract

Here comes the abstract

1 Introduction

2 Skyline detection

TODO MOTIVATION This section describes how the skyline of an image is detected. A skyline separates a building from the sky and is used as an indicator of the building contour.

Some previous work on skyline detection is done.
[9] yields a good introduction of different skyline detection techniques.
A comparison is done of the different skyline techniques as described in [9] and [1] is most suitable as a basis for the purpose of this project. This method is used as an inspiration. A custom algorithm is made. First the original method is explained. Then the custom algorithm with respect to the original algorithm is explained.
The skyline detection algorithm as described in [1] analyzes every column of the image separately. The smoothed intensity gradient is calculated from top to bottom. The system takes the first pixel with gradient higher then a threshold to be classified as a skyline element. This is done for every column in the smoothed intensity gradient image. The result is a set of coordinates of length W, where W is the width of the image, that represent the skyline. *Should I elaborate (with a footnote?) on the smoothed intensity gradient or could I assume this as common knowledge?.*
Taking the smoothed intensity gradient is the most basic method of edge detection and has a disadvantage. The method will not be robust to more vague edges. Instead of the smoothed intensity gradient in this thesis a smoothed edge intensity image is used. A practical study is done on the different Matlab build in edge detection techniques. The Sobel edge detector came with the most promising results. Details on this study lie without the bound of this thesis.

As a purpose of making the algorithm more precise, two other preprocessing steps are introduced. First the contrast of the image is increased, this makes sharp edges stand out more. Secondly the image undertakes a Gaussian blur, this removes a large part of the noise.
The system has no several parameters which has to be set manually by the user:

- contrast parameter,
- intensity (window size) of Gaussian blur,
- Sobel edge detector threshold
- column based inlier threshold

Should I write down what parameter values I used or is this of too much detail (Automatic parameter estimation based on the image would be interesting future work but lies without the scope of this research.)
The column based approach from [1] seem to be very useful and is therefor used as described above but on the preprocessed image.
Some results from the Floriande dataset: *TODO Results images TODO make UML scheme skyline -¿ 3d -¿ etc.*
The system assumes that the first sharp edge (seen from top to bottom) is always a skyline/building edge. This gives raise to some outliers, these are for example a streetlight or a tree. The outliers are removed as described in the next section. The Skyline detector without outlier removal has an accuracy of (ABOUT) 90 % (TODO CHECK THIS PERCENTAGE)

3 Skyline in 3D as a contour of the building

The retrieved skyline is used to update a sparse 3D model of the building. This section describes how the skyline of the 2D images are used to get the 3D contour of the building.

3.1 Project to 3D space

TODO situation scheme Every 2D pixel of an input image presents a 3D point in space. No information is known about the distance from the 3D point to the camera. What is known is de 2D location of the pixel, this reduces the possible points in 3D space to an infinite line. This line is known and spanned by two coordinates:

- The camera center
- $K'p$, where K is the Calibration matrix of the camera and p is the homogeneous pixel coordinate.

TODO why K'p,I don't remember the theory behind it and can't find it in Isaac's paper For every skyline pixel a line spanned by the above two coordinates is derived.

3.2 Intersect with building

The line is reduced to a point in 3D by intersecting it with a rough indication of the building. This point is used to update the 3D model.
To create a rough indication of the building a top-view photograph of the building is used together with an estimate of the height of the building.
In order to refine this sparse 3D model we need to know which skyline part belongs to which part of the 3D model. This is done as follows.
The building is first divided into different walls. Every wall of the building spans a plane. As described in the previous section every part (pixel) of the skyline presents an infinite line in 3D. Intersections are calculated between these infinite lines and the planes of the building walls.
Isaac, should I put a intersection formula down here or is this trivial?
Because the lines and the planes are both infinite and they have a very low change of being exactly parallel, the algorithm returns w intersections for every skylinepixel (where w is the number of walls).
Next challenge is to reduce the number of intersection for every skylinepixel to one. In other words, to determine the wall that has the largest probability of being responsible for that pixel. This is ofcourse needed to update the 3D model at the right place.

3.3 Find most likely wall

3.3.1 When is a wall responsible?

Lets define the intersection between the projected skylinepixel line and the plane of the wall as intersection point isp. And the wall sides as $w1, w2, w3, w4 \in W$. And d as a distance measure which is explained later on.
If we assume that a certain wall was responsible for that pixel, the intersection (i.e. projected pixel) must lie either

(1) Somewhere on the wall
or
(2) On a small distance d from that wall (1) is calculated by testing if the pixel lies inside the polygonal representation of the wall. This is done using the Matlabs buildin in-polygon algorithm. If this test succeeds we consider d to be 0.

(2) Note that this is treated as an inlier because the 3D model is sparse and the height of the building is estimated. It is calculated as follows:
First the distances from isp to four wall sides are calculated. For every wall the minimum distance is stored.
$min_{w \in W} d(isp, w)$
This is done for every wall. The wall with the smallest distance is the one that most likely presents the pixel.
$argmin_{W \in Walls}(min_{w \in W} d(isp, w))$
do I write this down correctly?

If there are two (or even more) walls that are classified equally well to present the pixel (that is if they succeed the in-polygon or have exactly the same d value) then the nearest wall is selected. The nearest wall is calculated by taking the wall with the smallest distance from the isp to the camera center. *formula here? or: trivial?*
How this intersection point - wall distance d is calculated is explained in the next section.

3.3.2 Calculate the intersection point - wall distance

A wall consists of four corner points. The corner-point pairs that are on a side of the wall connect line segments, there are four line segments. These line segments span infinite lines.

The intersection point (isp) is projected orthogonally on these four lines, a projected isp is called isp_{proj}. *todo image? todo projection formula?*
If isp is close to a wallside, $e(isp, isp_{proj})$ (where e is defined as the Euclidean distance) is small. But this doesn't mean that if $e(isp, isp_{proj})$ is small isp is always close to the wall. In fact there are some candidates that happen to have a very small $d(isp, isp_{proj})$ but in fact lie far away from the wall. This is because isp is projected to an infinite line spanned by the wallside and could be projected far next to the wallside. *An example can be seen in figure: TODO Figure* Because of this artefact, it is not robust to calculate the perpendicular projection distance. Instead d is calculated differently if isp_{proj} doesn't lie on the wallside. In this case the Euclidean distance between isp_{proj} and the closest corner-point of the wallside is returned.
Formally: Let $c1, c2, c3, c4 \in Cornerpoints$ be the corners of a wall.
Let $between(a, b, c)$ be a function that returns true if a lies between b and c.
if $between(isp_{proj}, c1, c2)$
$d = e(isp, isp_{proj})$
otherwise
$d = min_{c \in Cornerpoints} e(isp, c)$

TODO nice latex code with large $\}$ sign

3.3.3 Appendix?: Determine whether isp_{proj} lies on or next to a wallsegment

To determine whether the isp_{proj} lies on or next to a wallsegment the projection calculation is skipped and isp is used instead together with a computational cheap trick.
todo insert http://softsurfer.com/Archive/algorithm_0102/Pic_segment.gif here
First consider Figure ?? , the angles between the segment P0P1 and the vectors P0P and P1P from the segment endpoints to P. If both angles are equal to or less than 90° then the isp_{proj} wil be on the line segment P0 P1. If not, the

isp_{proj} lies to the left or to the right of segment P according to whether one of the angles is acute or obtuse. The angles are acute or obtuse if the dot product of the vectors involved are respectively positive or negative.
To summarize: determining in which region the isp_{proj} lies is boiled down to two dot product calculations with the advantage that the actual projection calculation can be skipped.
Todo come back on outlier removal

4 Update 3D model

5 References

- [1] Castano, Automatic detection of dust devils and clouds on Mars.
- [9] Cozman, Outdoor visual position estimation for planetary rovers.