Camera grid extrinsic self-calibration

Tim Lenertz

July 6, 2017

Contents

1 Introduction

This report describes the method used to calibrate the extrinsic camera parameters of the 3DLicorneA data sets. If is applicable for dense 2D data sets, where cameras are placed on a more or less regular grid on a plane parallel to the scene. The optical flow of tracked features, and aggregated values from the depth maps, are used to compute the camera positions and orientations. No calibration pattern needs to be present in the scene.

1.1 Requirements

To use the method, there are the following requirements:

- The intrinsic camera parameters are known. This is the camera matrix K and optionally the distortion coefficients. The method was used for the case where there is no distortion (Kinect v2), but with additional steps it can be used with distorted images.
- The intrinsic camera parameters are the same for each image.
- There is a depth map for all or some views. There is an extension to use the method without depth maps, and instead giving a depth only for some feature points.
- The camera centers are arranged on an approximately regular 2D grid on a plane P. The distance between adjacent camera positions (in X and Y direction) is sufficiently small that feature points can be tracked using optical flow. For this disparities of features on adjacent views should be smaller than 15 pixels.
- It is assumed that camera centers lie always exactly on the plane. That is, they never more towards or back from the scene.
- The camera is facing approximately perpendicular to the plane P, towards the scene. There can be a small rotation $\mathbf{R}$ of the camera relative to P. It is estimated from the images, as part of the calibration. The yaw, pitch and roll angles should be smaller than 5°. It is assumed that this angle remains constant for all views.
- There can be some missing images and depth maps in the data set.
- It is not necessary that the tracked feature points remain visible across the whole range of views. Calibration can be done on subsets of the camera positions, and then stitched together.

The method calculates one global rotation matrix $\mathbf{R}$, and for each view v, a 2D vector (x_v, y_v) of the camera center position on the plane P. From this it computes an extrinsic matrix $\mathbf{Rt}$ for each view.

2 Method

Calibration is done in four steps: (1) Compute *image correspondences* using feature tracking. (2) Estimate the *camera rotation* $\mathbf{R}$. (3) Estimate *straight depths* of the tracked features, i.e. their distance to P. (4) Deduce camera positions.

2.1 Overview

First the algorithm selects several *feature points* on one on multiple *reference views*. Using optical flow, it then tracks the position of the same features on the other views.

With the pin-hole camera model, it is be possible to calculate the camera position on P directly from a feature point's positions on different views, if the camera is pointing perpenticular to P, and the feature's distance to P is known. So the algorithm first needs to estimate $\mathbf{R}$, and the straight depths of the features.

To calculate $\mathbf{R}$, two methods are used. One uses a non-linear model which estimates $\mathbf{R}$ only from the slope of the lines that the tracked features make when the camera moves horizontally and vertically, without knowledge of the features' depths. The other method uses the depths of the features on the different views. Both estimate a full 3D rotation matrix.

The distance of a feature to P is called its *straight depth*. Knowing $\mathbf{R}$, it can now be calculated from the feature points' depths in each view's depth map. If depth maps are not available, it is also possible to fix only the depth of one or more features, and deduce the rest from the relative scales of the different feature's disparities.

Using $\mathbf{R}$ and the *straight depth* of each feature, the algorithm now estimates the set of camera positions on P, once for each feature. The results are aggregated to find the final camera positions.

The resulting camera positions are in a coordinate system with the camera of the *reference view* at origin. If multiple reference views were used in the feature tracking step, the entire procedure is repeated for each reference view, and in the end the camera positions are stitched together.

2.2 Preliminaries

The 2D dataset consists of several *views*. A view v consists of an image, and optionally a depth map, taken from one camera position. The views are enumerated with two integer indices $v(x, y)$. Views with the same x index are (approximately) aligned vertically, views with the same y index horizontally. This is relative to the camera image planes.

The goal is to estimate the position and orientation of the camera for each view, i.e. to find the extrinsic camera matrices $\mathbf{Rt}_v$.

If depth maps are used, they need to be in the same coordinate systems as the images. For each image pixel (i_x, i_y), the value d of the same pixel in the depth map needs to indicate the orthogonal distance from the camera center to that object point, orthogonal to the camera image plane. The camera matrices $\mathbf{Rt}_v$ will be expressed in the same unit as these depths.

2.3 Image correspondences

First some features f are selected. They correspond to 3D points in the scene. This step aims to find for each feature f, the set of *feature points* $p(f, v) = (x, y, d, w)$, that is the pixel coordinates x, y where the feature is visible in each view v. This data is called the *image correspondences*.

A feature point optionally also contains a depth d, and a weight w. The depth is a distance orthogonal to the camere image plane of v. If $\mathbf{R} \neq \mathbf{I}$, then the depths $p_{f,*}$ of the same feature for different views will be different.

2.3.1 Choosing feature points

Features are chosen by choosing feature points on a *reference view*. By default the center view in the dataset is used as reference view, but there can also be multiple reference views (see later).

The chosen feature points need to be such that they are likely to remain *stable* when doing feature tracking. It means that when one looks for a similar-looking nearby point on an adjacant view, it is likely to be the same scene point. An example is shown on figure 2.1.

The *OpenCV* function `goodFeaturesToTrack` is used. Additionally, the image is first subdivided into 4 or more rectangular regions, and the best chosen features from each region are taken.

The chosen features should be well distributed across the image, and have different depths. There should be about 300 or more features, considering that many will be filtered out because their optical flow is unstable.

2.3.2 Optical flow tracking

Optical flow feature tracking is always done on adjacent views, for example $v(x, y)$ and $v(x + 1, y)$. Then sequentially, it uses the corresponding feature points on $v(x + 1, y)$ to estimate those for $v(x + 2, y)$, and so on. So there is an error accumulation, which gets worse the longer the path that the view indices take is.

The acquisition system moves line-by-line. So it is physically guaranteed that for any $v(x, y)$ and $v(x + 1, y)$, the camera only moves by small amount, whereas for $v(x, y)$ and $v(x, y + 1)$, there can be a larger deviation. So it is better to take most optical flow correspondences in x direction.

Figure 2.1: Chosen feature points

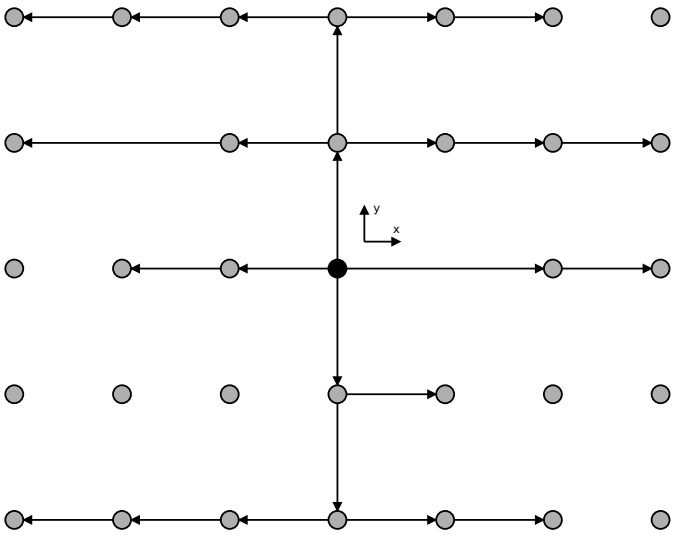

Figure 2.2: Optical flow paths

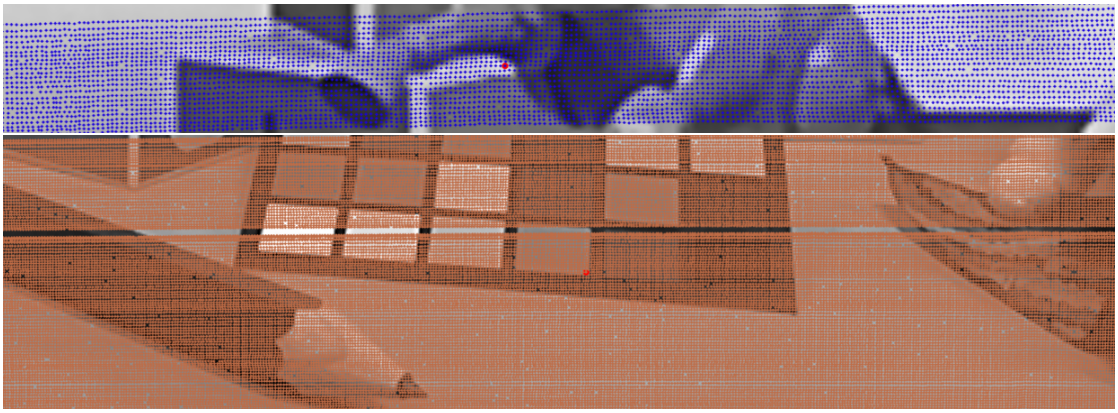

Figure 2.3: Good feature correspondences

The optical flow algorithm moves over the (x, y) view indices as shown on figure 2.2. The center view (black circle) is the reference view. As each step moving from (x, y) to (x', y'), for each feature f, all the feature points $p(f, v(x', y'))$ are computed from those of $p(f, v(x, y))$. If no feature point $p(f, v(x', y'))$ could be computed anymore, the algorithm stops for that line.

If the image for a view is missing, that view is skipped, and instead the correspondences are taken from the second-previous view, as shown. It is important that no view is missing in the column of the reference view, because then that entire line will be skipped.

The reference view is not one of the edges, but instead in the center, and the optical flow steps are done in all four directions. This minimizes the total path taken, and reduces the accumulated error.

The maximal number of steps to be taken in x and in y direction, called *outreach*, can be set to a maximal limit. Using a smaller *outreach*, and instead doing the calibration from multiple reference views, and combining the results in the end, can produce better final results.

To compute the optical flow, the *OpenCV* function `calcOpticalFlowPyrLK` is used. The parameters can be adjusted. It can also be set to use multi-scale image pyramids, so that larger features can be used.

Figures 2.3 are examples of good feature correspondences. The background image is a close-up of a view image. The red dot is the feature point on this view. The other dots are the corresponding feature points on the other views.

2.3.3 Filtering features

The next step is to filter the generated *image correspondences*. It is important, because incorrect feature points can have a large impact on the final results. The optical flow procedure tends to generate a large number of bad correspondences. There is an algorithm to automatically filter out bad correspondences, but they should also be verified by hand.

Figure 2.4: Bad feature correspondences

Figures 2.4 are examples of good feature correspondences. In the first example, the deviation occured because a foreground object with a curved border moved in front of the tracked feature. Such deviations can also occur because of specular reflections (for example on the metal sink), and because of badly chosen feature (such as on the furry objects). In the second example, the pattern appears regular but the correspondence is still incorrect.

The filtering algorithm removes all feature points for one a feature f entirely, if there are too little feature points, or if the pattern deviates too much from a regular lattice.

Properly filtering the correspondences is important: Having incorrect correspondences, and having too little correspondences, both have a large impact on the final result. In practice, for each view, there should remain feature points for about 100 features.

2.3.4 Reference views

If the range of motion of the camera is large, or the field of view is small, many feature points that are visible in the center view, will not be visible on the extremity views.

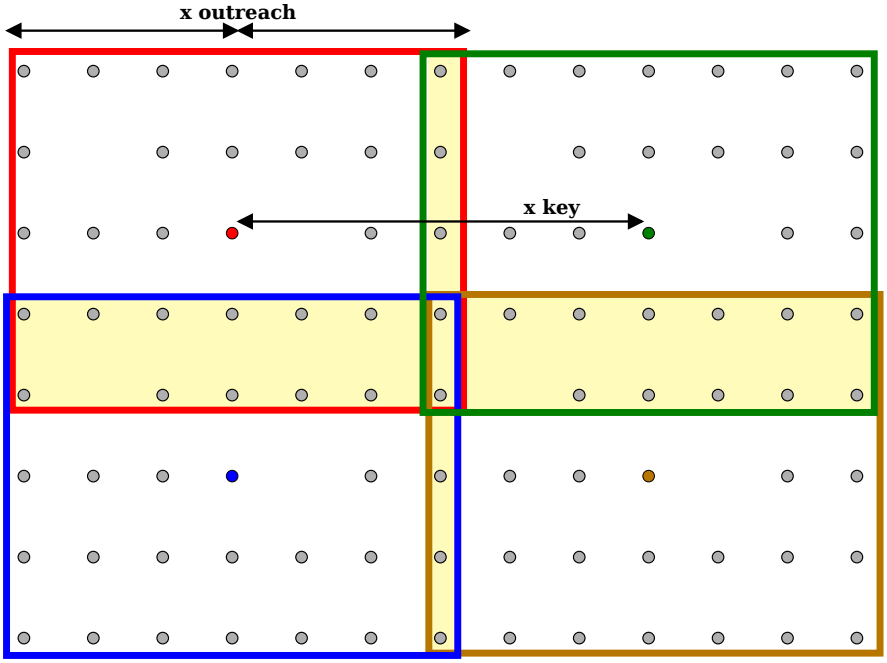

Figure 2.5: Reference views arrangement

Also as said before it can be good to limit the optical flow *outreach*, so as to reduce the accumulated error.

When multile reference views are used, the entire calibration will be done for each reference view independently (except for the rotation estimation, as $\mathbf{R}$ is constant).

Figure 2.5 shows how reference views can be selected. The four colored dots are the reference views. They must be selected so that they are in a grid, for the stitching algorithm to work later. It is called the *reference grid*. The x *key* and y *key* is the distance between reference view indices, in x and y direction.

In the figure, the outreach is set so that for feature points chosen for one reference view, correspondences can only be found for views within the same-colored rectangle. For the stitching to work, there must be some overlap in these rectangles, as shown by the yellow regions in the figure.

The overlap should be large enough so that there will remain many views with feature points from two references, after the filtering.

For example an *outreach* of 80, and a *key* of 100 can be a good choice. It would leave an overlap range of 60.

The algorithm chooses the *reference grid* so that there are no missing images on the columns of the reference views' x indices.

2.3.5 Feature point depths

If depth maps are available, they are used to attribute a depth value d to each feature point $p(f, v) = (x, y, d, w)$. The depth is read from the view's depth map, at pixel position (x, y).

However, features are often located on the border of foreground objects. Taking a single pixel value in the depth map could incorrectly take the depth of the background, or an incorrect intermediary value. Therefore the algorithm takes a small pixel window around (x, y), and retains the minimal value in it.

2.3.6 Feature point weights

Feature points $p(f, v)$ can have a weight value. If many feature points are clustered together on a small region of the image (for example a checkerboard), it is reasonable to given them a lower weight, and to give a higher weight to more isolated feature points. The weight could also depend on a confidence value calculated for the correspondence.

Especially for the rotation estimation from optical flow slopes (see later), it is important that all regions in the image are uniformly represented.

This is not implemented.

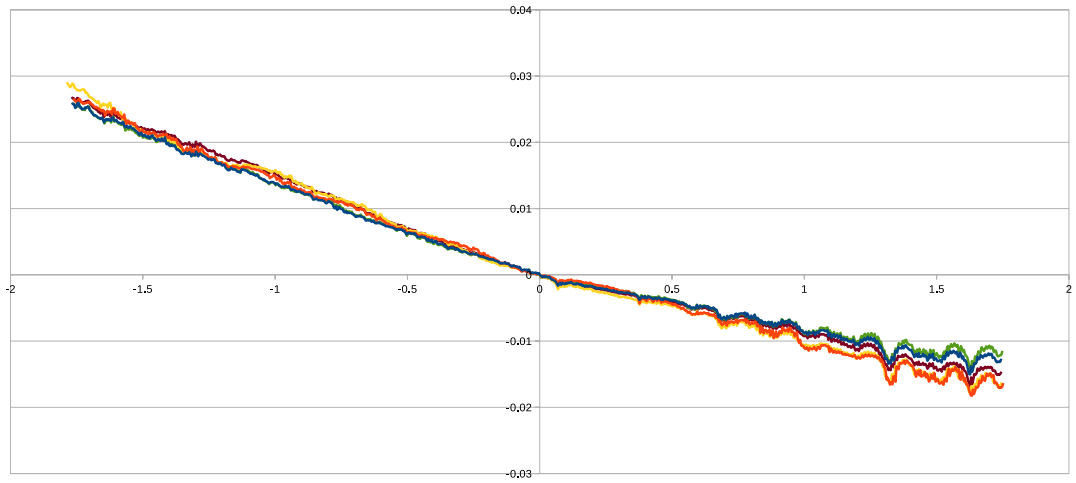

Figure 2.6: Overlayed feature points in 1D case

2.4 Observations

Looking at the image correspondences on figures 2.3, it can be seen that the arrangement of the feature points roughly corresponds to the (inverted) camera positions on P. (The vertical gap on the second figure is a result of the acquisition system: It did not take the y-step properly at that height.)

The feature points for every feature f will be arranged in the same pattern, just at different places in the image, and with different scales (i.e. disparities). The basic idea of this calibration method is to overlay the feature points for several features, make their scale uniform, and take the averages, to get a good estimate of the camera positions.

2.4.1 Rotation

However this would only be correct if $\mathbf{R} = \mathbf{I}$. In reality, the rotation $\mathbf{R}$ distorts the feature points. For figure 2.6, a 1D optical flow was taken, with the camera moving on a horizontal axis only. The feature points were then overlayed, centered on one feature point, and given uniform scale. The figure shows these transformed feature points, one color for each feature. It can be seen that they form lines with different slopes. The scaling does not affect the slope. The different slopes are caused by the camera's rotation $\mathbf{R}$. In the 2D case, it is possible to calculate $\mathbf{R}$, from these slopes alone. This is done in section 2.5.

2.4.2 Depth

Another problem is that if more than two sets of feature points should be given a common scale, one set of feature points would need to be chosen as reference. This would amplify the error in the correspondences of that particular feature. So it is better to calculate

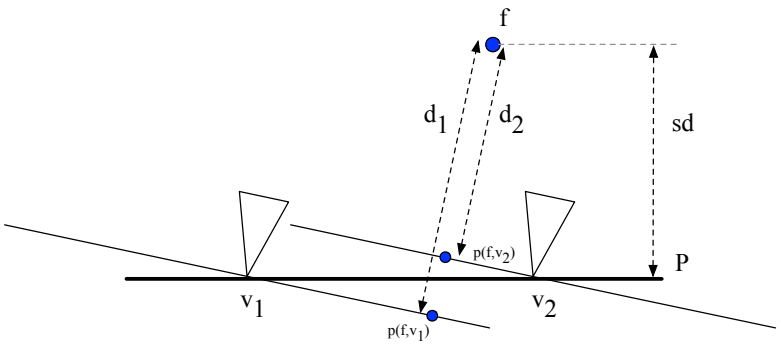

Figure 2.7: Differing feature point depths

scaling factors in a more global way. This is done with the *straight depths*, in section 2.6.

Figure 2.7 shows how because of the rotation $\mathbf{R}$, the feature points depths d_1, d_2 for one feature f are not all equal. In section 2.6 all of these feature points depths are aggregated together, to calculate the straight depth sd, using $\mathbf{R}$ and the camera intrinsic matrix $\mathbf{K}$.

2.5 Camera rotation

The next step is to determine the camera rotation $\mathbf{R}$. This is the rotation of the cameras relative to the plane P on which the camera centers are placed. It is a 3D rotation matrix, with 3 degrees of freedom.

If $\mathbf{R}$ is known to be very small, this step can be skipped entirely, by simply setting $\mathbf{R} = \mathbf{I}$.

Otherwise, there are two methods to estimate $\mathbf{R}$: One is based only on the *slopes* of the image correspondences, and does not need depth maps. It needs the cameras to be aligned on a regular grid. It seems to produce an accuracy of around 0.5° for the three rotation angles. It is described in the next section 2.5.1.

The other uses the differing depths of the feature points to estimate the rotation, by doing a least squares plane fitting operation, followed by an adjustment of the roll rotation. It is described in the next section 2.5.2. It seems to give better results. Its implementation is only experimental, and there are probably some errors in it.

2.5.1 Optical flow slopes

2.5.1.1 Flow equation

The camera intrinsic matrix $\mathbf{K}$ projects points in the camera view space (v_x, v_y, v_z), to pixel positions on the image (i_x, i_y) in homogeneous coordinates, according to

$$w \begin{bmatrix} i_x \\ i_y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{2.1}$$

$$i_x = f_x \frac{v_x}{v_z} + c_x \qquad i_y = f_y \frac{v_y}{v_z} + c_y \tag{2.2}$$

$$v_x = \frac{v_z}{f_x}(i_x - c_x) \qquad v_y = \frac{v_z}{f_y}(i_y - c_y) \tag{2.3}$$

As shown on figure 2.7, the camera moves such that the camera center is on a plane P, and the camera has a constant rotation $\mathbf{R}$ relative to P.

The *world space* coordinate system is set such that its $z = 0$ plane is P, and its origin is any point $\vec{O}(0, 0, 0) \in P$.

Let $\vec{i}(\vec{O})$ be some point in the image of the camera when it is placed at O. Let $\vec{v}(\vec{O})$ be the same point in the camera's view space, calculated with formula 2.3, with a given value $z = v_z(\vec{O})$. Let $\vec{w}$ be the same point in *world space*. For any camera center position $\vec{Q} \in P$, the relation is

$$\vec{v}(\vec{Q}) = \mathbf{R}(\vec{w} + \vec{Q}) \tag{2.4}$$

To get from world space to view space, the coordinate system is first translated by $\vec{Q}$ (where $Q_z = 0$), then rotated by $\mathbf{R}$. In particular,

$$\vec{v}(\vec{O}) = \mathbf{R}\vec{w} \tag{2.5}$$

hence

$$\vec{v}(\vec{Q}) = \vec{v}(\vec{O}) + \mathbf{R}\vec{Q} \tag{2.6}$$

Using formula 2.2, $\vec{i}(\vec{Q})$ can now be calculated from this. So one has the function

$$\text{flow}_{\mathbf{R}} : \langle \vec{i}(\vec{O}), \vec{Q}, z \rangle \mapsto \vec{i}(\vec{Q}) \tag{2.7}$$

2.5.1.2 Horizontal and vertical camera movement

The following section will analyze how $\vec{i}(\vec{Q})$ evolves when $\vec{Q}$ moves on P. Most importantly, it is shown that when $\vec{Q}$ moves horizontally or vertically on P, then the **slope** at which $\vec{i}(\vec{Q})$ moves on the image does not depend on z.

Let $\vec{H} = (\epsilon, 0, 0)$ and $\vec{V} = (0, \epsilon, 0)$. They represent a horizontal and vertical displacement of the camera on P, by some magnitude ϵ.

Because of the transformation between cartesian and homogeneous coordinates in formulae 2.2 and 2.3, the function $\text{flow}_{\mathbf{R}}$ cannot be expressed directly as a matrix equation. $\mathbf{R}$ is decomposed:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.8}$$

To simplify the expressions, the coordinates of the chosen image point $\vec{i}(\vec{O})$ are simply denoted (i_x, i_y). Also, $z = v_z(\vec{O})$.

For the horizontal camera movement by $\vec{H}$, one gets:

$$i_x(\vec{H}) = \frac{i_x z + f_x r_{11}\epsilon + c_x r_{31}\epsilon}{z + r_{31}\epsilon} \quad \text{and} \quad i_y(\vec{H}) = \frac{i_y z + f_y r_{21}\epsilon + c_y r_{31}\epsilon}{z + r_{31}\epsilon} \tag{2.9}$$

2.5.1.3 Slopes

It can be shown that as ϵ varies, $\vec{i}(\vec{H})$ moves on a straight line. Its slope is

$$s_H = \frac{i_x - i_x(\vec{H})}{i_y - i_y(\vec{H})} \tag{2.10}$$

This expression simplifies to

$$s_H = \frac{i_x - i_x(\vec{H})}{i_y - i_y(\vec{H})} = \frac{f_y r_{21} + c_y r_{31} - i_y r_{31}}{f_x r_{11} + c_x r_{31} - i_x r_{31}} \tag{2.11}$$

The variables ϵ and z both vanish. s_H depends only on $\mathbf{R}$, $\mathbf{K}$ and $\vec{i}(\vec{O})$.

For the vertical camera movement by $\vec{V}$, one gets the similar expression

$$s_V = \frac{i_y - i_y(\vec{V})}{i_x - i_x(\vec{V})} = \frac{f_x r_{12} + c_x r_{32} - i_x r_{32}}{f_y r_{22} + c_y r_{32} - i_y r_{32}} \tag{2.12}$$

Note that s_H is a slope x/y, whereas s_V is a slope y/x. This is because $\mathbf{R}$ is expected to be near $\mathbf{I}$, and in that case $\vec{i}(\vec{H})$ and $\vec{i}(\vec{V})$ move almost horizontally and vertically on the images respectively, and so both slopes approach zero (and not infinity).

2.5.1.4 Samples from image correspondences

In order to apply this to estimate $\mathbf{R}$, the dataset must be such that the camera centers of views $v(x-1, y), v(x, y), v(x+1, y), \ldots$ must be in an approximatly straight line ("horizontal"). For views $v(x, y-1), v(x, y), v(x, y+1), \ldots$ they must also also be an approximatly straight line ("vertical"), which is perpenticular.

After the *image correspondences* were computed, for each *feature* f, a horizontal and a vertical slope $s_H(f), s_V(f)$ are estimated using line fitting on the feature point correspondences for those view indices.

This gives for each feature f a sample

$$S_f = \langle p(f, v), s_H(f), s_V(f) \rangle \tag{2.13}$$

$p(f, v)$ is the feature point position of f for the reference view v. This corresponds to the (i_x, i_y) from the previous formulae. If multiple reference views were used, the samples from different reference views can be put together here.

2.5.1.5 Estimating camera rotation

It is possible to estimate $\{r_{11}, r_{21}, r_{31}, r_{12}, r_{22}, r_{32}\}$ from these samples by solving two linear homogeneous least squares systems. From this $\mathbf{R}$ could probably be completed knowing it is an orthogonal matrix with $\det(\mathbf{R}) = 1$.

But this is not done in the algorithm. Instead a parameterization of $\mathbf{R}$ with three Euler angles (X, Y, Z) is optimized with an iterative method. The error to minimize is the mean squares sum of the predicted slopes for a given $\mathbf{R}$, minus the measured slopes.

The parameterization (X, Y, Z) of $\mathbf{R}$ is $\mathbf{R}^\intercal = \mathbf{R_z}(Z)\mathbf{R_y}(Y)\mathbf{R_x}(X)$. $\mathbf{R}^\intercal$ is the orientation of the camera in world space. So the *roll* rotation Z (around the optical axis) is performed last.

The three angles are interdependent: Say X is adjusted to minimize the error. Then Y is adjusted to reduce the error even more. Now X is no longer at the optical setting, and needs to be readjusted.

The roll rotation has the most impact. Three golden-section searches are performed sequentially which optimize Z, X and Y, in that order. The entire process is repeated iteratively until a certain error threshold. With each (outer) iteration, and tolerance and search interval of the (inner) golden-section searches are reduced.

2.5.1.6 Accuracy

On an artificially generated test dataset with a known camera rotation of $(10°, 20°, 5°)$, the estimated rotation was $(10.5289°, 20.6345°, 5.36933°)$. This artificial dataset has some random noise and outliers, 200 features, and 30×30 views. It is hard to estimate the accuracy for real datasets because the real rotation is unknown and typically very small.

2.5.2 Feature point depths

2.6 Straight depths

Knowing $\mathbf{R}$, the *straight depth* of each feature f can now be calculated. It is the orthogonal distance of the feature point in the scene, to the plane P.

2.6.1 Aggregating feature point depths

2.6.2 Depth from disparity

2.7 Camera positions

Finally, having *image correspondences*, the *camera rotation*, and the *straight depths*, the camera center positions can be computed.

2.7.1 Relative camera positions

2.7.2 Stitching

2.7.3 Final camera extrinsics

3 Implementation