Fast Depth Map Estimation from Stereo Camera Systems

Project Documentation SLAM For UAV's

Malik Al-hallak 90020
Hagen Hiller 110514

Winter term 2014/15

Abstract

The major goal of the project *SLAM For UAV's* is to implement a SLAM algorithm on an *AscTec Pelican*. Since this goal covers several topics, we split the project group and work on different subtopics. In this documentation we describe how a stereo camera system is used to create depth maps, which then can be used for further tasks like obstacle avoidance or 3D reconstruction. We will describe the complete pipe line, from stereo camera images to a depth map and the tools needed.

1 Introduction

2 System Description

In this section we will describe the developed system in detail. First we will give an overview of computing depth maps in general. Furthermore,we will list the dependencies of the system. Next we will explain the developed framework and how the single components work with each other.

Although we will outline the pipeline of generating a depth image from a stereo system, we expect the reader to inform oneself about single steps and conventional knowledge in computer vision.

2.1 Depth Map Estimation 101

In this section we will give a rough introduction on how one can generate depth maps from a stereo camera system. This process is organised in three main steps:

- image preprocessing
- similarity measurement
- 3D-coordinate calculation

2.1.1 Image Preprocessing

The preprocessing step is done on two levels; one is the single camera and the second is the stereo system. Each image has some distortions due to the camera's lens. This distortions need to be corrected for the further steps. With the help of a calibration pattern we can calibrate the cameras. This will detect and correct distortions. The process is called *undistortion*.

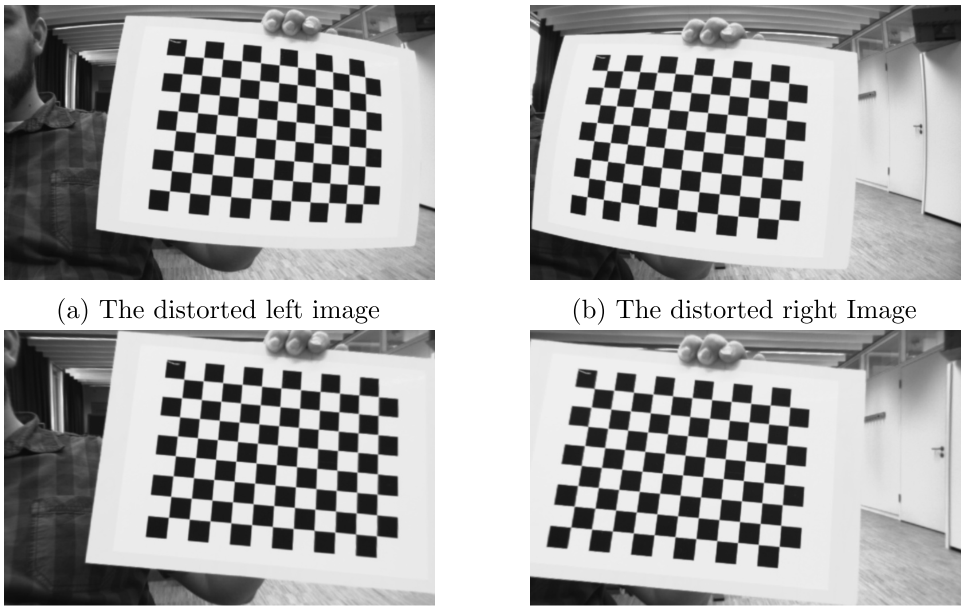

(a) The distorted left image (b) The distorted right Image
(c) The undistorted left image (d) The undistorted right Image

Figure 1: Camera calibration corrects distortions

In Figure 1 the result of the camera calibration is illustrated. It is obvious that the checkerboard edges are curved and the junctions do not have right angles. After the calibration the everything is straight and correct. At the same time it is visible that some information is lost during the calibration, which is due to "unwarping" the image.

During the process of depth map estimation we are going to take features in the left image and search them in the right image. But since this is a 2D search domain (x and y direction) it would be nice to ensure that the searched feature in the right image is at least in the same row as in the left image. This process is called *rectification*. After the rectification step, we can assume that the camera images have no horizontal shift anymore. This reduces the feature search problem to a one dimensional (vertical) search problem. Furthermore, under the precondition that there is only a vertical shift and both cameras are identical, we can say that a feature from column x_{Left} is at least at the same position in the right image so it holds that $x_{Right} \geq x_{Left}$. This also reduces the search problem, since the closer x_{Left} is to x_{Max} the smaller is the set of elements to search in.

Epipolar geometry We will explain the outcome of the rectification with the help of the epipolar geometry. Each 3D object point is mapped into the 2D image of both cameras. The object point O defines an image point $P(x_l, y_l)$ in the left image. The question is now, where is the corresponding point in the right image? To find the corresponding point one would have to search the epipolar line of that image point in the right image. This line is determined by the projection of the line O–$P(x_l, y_l)$ on the image plane of the right image. But this would mean that we would have to calculate the epipolar line for each pixel and search along this line for the corresponding point. The rectification simplifies the search such that the image planes are "corrected" in such a way that all epipolar lines are parallel and horizontal. This ensures that the corresponding point is on the same row as the source point. The process is illustrated in figure 2

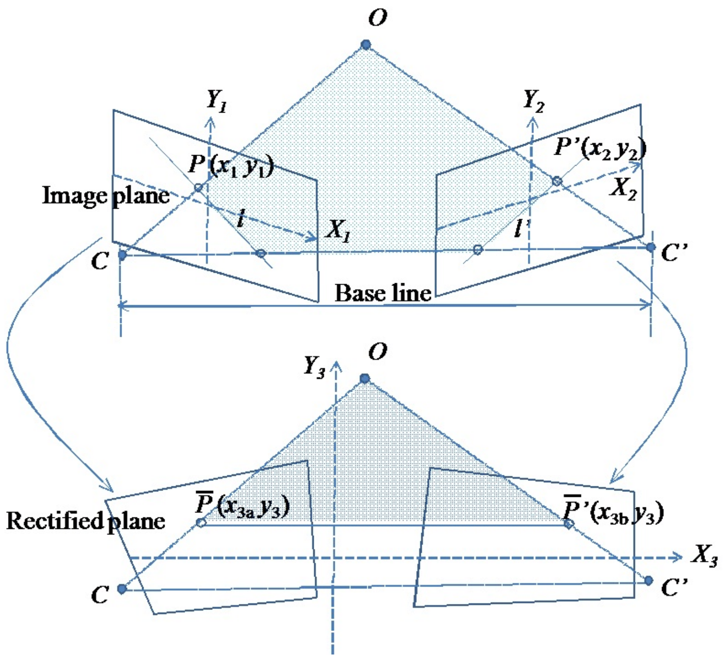
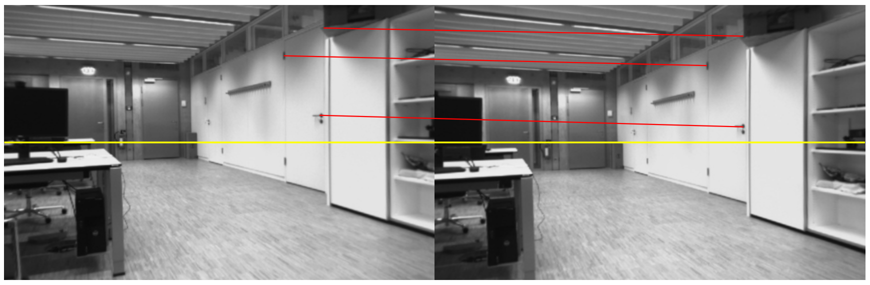


Figure 2: Epipolar geometry and image rectification[1]

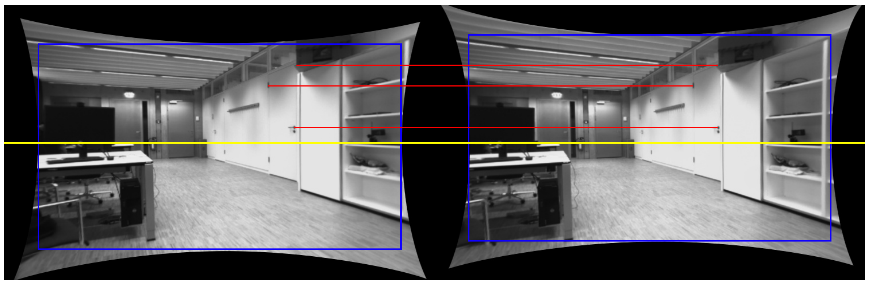
The search problem is shown in figure 3. The red lines connect related features in the left and right image. The yellow line shows the medial pixel row; therefore, it is perfectly vertically. One can see that the red lines and the yellow line in Figure 3a are not parallel. This indicates that the related features are not in the same row. Thus, several rows has to be searched for corresponding features; a 2D search problem. Figure 3b shows the rectified images. During the rectification the image is stretched, warped etc. to parallelize and horizontalize the epipolar lines. The blue rectangle indicates the *valid* area of the rectified images. This is the area where no black pixels, which are introduced during rectification, are part of the image. Figure 3c shows the cropped and scaled rectified image. All corresponding points are now on the same row.

After the rectification process the

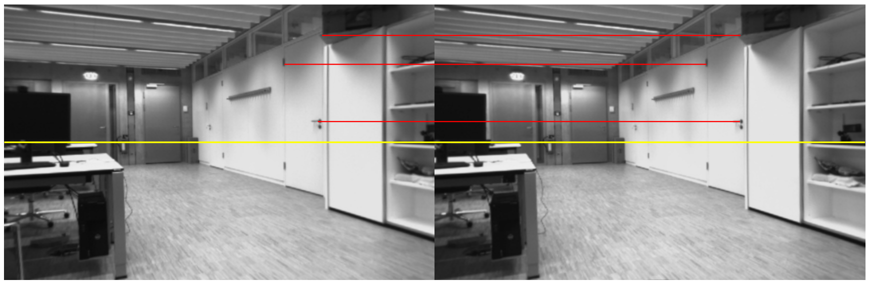
[1]Image courtesy http://www.intechopen.com/source/html/16684/media/image4.jpg



(a) Corresponding points in the undistorted image



(b) The corresponding points in the rectified images



(c) The corresponding points in the cropped and scaled rectified images

Figure 3: Image rectification result

2.1.2 Similarity Measure

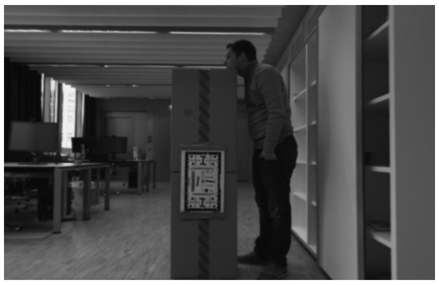
At this step we assume that we have undistorted and rectified images. Now we compare the left and right images. If the horizontal shift of an object from the left to the right image is large, the object is near the cameras. If the horizontal shift moves towards zero, the object is far away from the origin (the cameras).

But how can one determine the horizontal shift of an object? There exist plenty possibilities to do this. We tried three different approaches: template matching with different similarity measures, block matching and semi-global block matching. The result of each is a map with integer values. The map has the same size as the two images and contains for each pixel $p(x, y)$ a value (the disparity) which expresses the horizontal shift in pixels from the left to the right image. In our framework we use the semi-global block matching (SGBM) since it has the best tradeoff between time and accuracy. The block matching mechanism is a little bit faster but very noisy at the moment. The template matching strategy takes far to long to evaluate a single picture pair.

The SGBM tries to minimize a global energy function through different paths to the current pixel/block. Therefore different parameters has to be passed to the algorithm. Please have a look on [1] and [2] for further reading on the semi global matching. The SGBM extends the explained semi-global matching to a faster semi-global block matching. The SGBM is implemented in the OpenCV framework.

Furthermore the SGBM guarantees smooth disparity maps with closed shapes (depending on the parameters passed). Figure 4 shows the input and the result of the SGBM disparity calculation.

(a) The recitified left image


(b) The rectified right Image

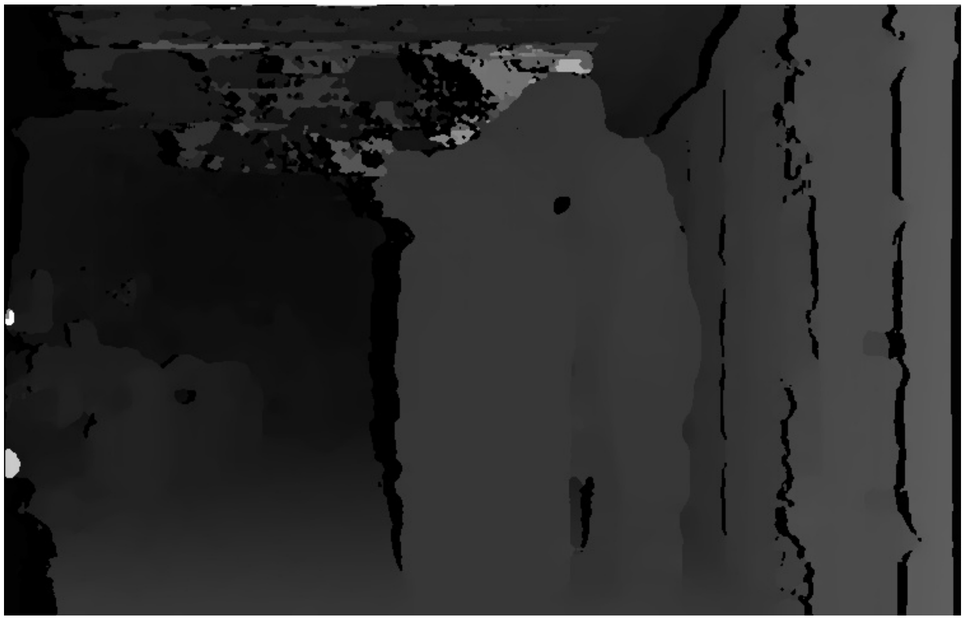

(c) The calculated disparity map using the SGBM

Figure 4: Disparity calculation using SGBM

2.1.3 3D-Coordinate Calculation

The last step is the reprojection from the image plane to the real world. We therefore use the reprojection matrix Q, which is constructed as follows:

$$Q = \begin{pmatrix} 1 & 0 & 0 & -C_{x\,left} \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 0 & f_x \\ 0 & 0 & -\frac{1}{T_x} & \frac{C_{x\,right} - C_{x\,left}}{T_x} \end{pmatrix}$$

$C_{x\,left}$ – x coordinate of the principal point of the left camera
$C_{x\,right}$ – x coordinate of the principal point of the right camera
C_y – y coordinate of the principal point in both cameras
f_x – focal length in x direction of the rectified image
T_x – the baseline

So to reproject a point to 3D coordinates we need the x and y position in the left image, the disparity value $d(x, y)$ and Q. We construct a vector $\vec{o} = (x\ y\ d\ 1)^T$ and multiply Q with it so we obtain 3D coordinates $\vec{c} = (X\ Y\ Z\ W)$ which we have to divide by W.

$$\vec{c} = Q \cdot \vec{o}^{\,T} = (X\ Y\ Z\ W)^T$$
$$\frac{\vec{c}}{W} = \left(\frac{X}{W}\ \frac{Y}{W}\ \frac{Z}{W}\ 1\right)^T$$

This can be done with every pixel (x, y) which has a valid disparity value d. It is possible to determine the coordinates of single points, e.g. for obstacle avoidance, or to produce point clouds from the images, e.g. for 3D reconstruction tasks.

This three steps form the rough pipe line to generate depth maps from stereo camera systems. In the further sections we will explain the developed framework.

2.2 Hardware and Dependencies

The AscTec Pelican is equipped with two *mvBlueFOX – MLC 200w* cameras from *Matrix Vision*. To use the cameras one need to download and install the mvIMPACT Acquire SDK. This SDK comes with the required drivers and a API for several programming languages such as C/C++, C# etc.

The system uses the SDK just for image acquisition, either in full resolution (752x480) or in binning mode with half resolution (376x240).

After the images are acquired they are transferred to OpenCV matrices. Therefore, OpenCV is the second dependency one will need to use the framework. We used version 2.4.9 to develop and test the framework. OpenCV is used for nearly every image processing and computer vision task in the framework.

The last dependency is a C++ compiler with C++11 support, such as the gcc-4.9.2.

2.3 Framework Components

The three steps which we described earlier in section 2.1 are reflected in the framework composition. The framework consists of the two classes `Camera` and `Stereosystem` and the namespaces `disparity` and `utility`. In Figure 5 the data flow between the components is shown.

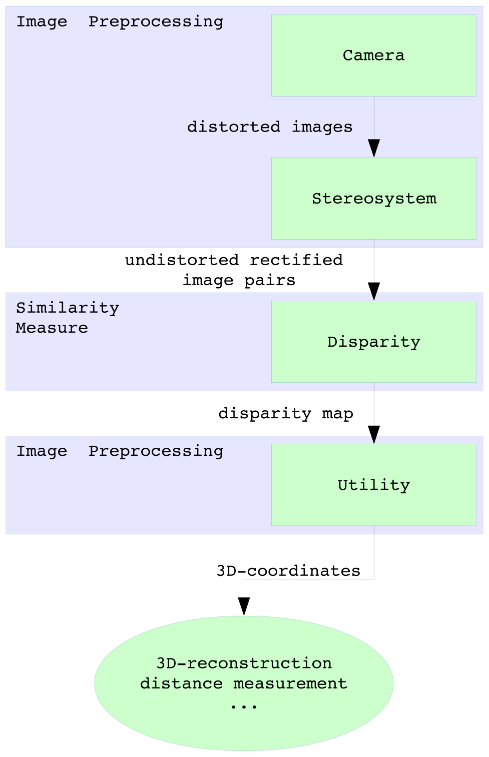


Figure 5: Framework components

The `Camera` class encapsulates the complete Matrix Vision API from the rest of the framework. On the one hand it realizes the image acquisition, on the other hand it allows to set some camera specific parameters like binning mode, exposure time or gain.

The images from the left and right cameras are then passed into the `Stereosystem` class. This class handles the complete stereo setup. It is responsible for calibration, undistortion and rectification.

The `Stereosystem` class offers functions to calibrate the cameras with a given set of calibration images, save and load former calibration parameter and return different types of image pairs, namely distorted, undistorted and rectified image pairs.

We use the OpenCV function `calibrateCamera` to get the so called *intrinsic paramters*, which are the *camera matrix* and the *distortion coefficients*. The camera matrix A contains the focal length and the principal point.

$$A = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

The framework searches for some object points in some calibration images which contain a calibration object. The calibration object is a checkerboard with known dimensions and known number of rows and columns. The algorithm then calculates the camera matrix and the distortion coefficients such that the object points (the checkerboard intersections) are collinear and hence, the checkerboard edges are straight.

The calibration function returns the root mean square reprojection error (RMS). For a good calibration it should be between 0.1 and 1 pixel. With the Matrix Vision cameras and the wooden calibration object we expect the RMS to be less then 0.2. We made the experience that 10 to 15 images are sufficient for a good calibration. Fewer or more images could downgrade the calibration result.

For the stereo setup the intrinsic parameters are not enough. We need further information about placement and orientation of the cameras to each other for correct rectification. This information is called *extrinsic parameters*. The extrinsic parameters describe the translation and rotation of the right camera to the left camera. By the way, we always assume the left camera to be the reference system. With the help of the OpenCV function `stereoCalibrate` we can determine the extrinsic parameters ([R]otation matrix and [T]ranslation matrix), the [F]undamental and [E]ssential matrix.

Both together, the intrinsic and extrinsic parameters, are then passed to the `stereoRectify` function to get the rectified images.

For a detailed description of the math implemented in OpenCV please have a look at the 3D reconstruction reference from OpenCV.

After the `Stereosystem` class returns the rectified image pairs, they are passed to disparity calculations. At the moment the `disparity` namespace offers semi-global block matching, block matching and template matching for disparity calculation.

The generated disparity maps can then be used to calculate 3D coordinates. Since this is just a single function (which is in fact just a single multiplication) we decided to put it into the `utility` namespace.

For some examples how the framework us used for image acquisition and disparity calculation please have a look into the `example/` directory.

3 Evaluation

4 Discussion

5 Future Work

References

[1] Hirschmuller, H. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (2005), vol. 2, IEEE, pp. 807–814.

[2] Hirschmuller, H. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 30*, 2 (2008), 328–341.

[3] OpenCV. Camera calibration and 3d reconstruction. http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.