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 pipeline, from stereo camera images to a depth map and the tools needed.

1 Introduction

2 Related Work

3 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.

3.1 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

3.1.1 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*

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 cameras 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.

3.1.2 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 integers. 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 since it has the best tradeoff between time and accuracy. The block matching mechanism is a little bit faster but very noisy and the template matching strategy takes far to long to evaluate a single picture pair.

3.1.3 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:

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

with

- $C_{x\ left}$– 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 of both left and right cameras
- f_x– focal length in x direction of the rectified image in pixels
- T_x– translation in x direction from the left camera to the right camera

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.

3.2 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.

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

For our testing setup we attached the cameras to a magnetic stand which cn be moved onto a steel plate. In order to get exact real world measurements we also used a laser operated distance measurement device (DISTO NAME ETC).

4 Evaluation

In order to evaluate our system we used the tools described before in Hardware and Dependencies. The Disto was aligned beyond the left camera sensor. Using this setup we could measure our distances using the written Framework and check the values using the laser measurement. Basically we tested our Setup in three different ways:

1. Absolute measurements from the Disto to an object placed in the room
2. Relative measurements between two objects placed in the room
3. Obstacle size measurements to determine the minimum reckognizable size of an object

These tests will be explained and evaluated in the following sections.

4.1 Absolute measurements

For the absolute measurements we placed obstacles in the room starting from 8.4 meters in 0.6m steps. The results are visualized in the following figures:

BAR CHART

TABLE

When taking a look at the outcomes of the absolute test it is clear to see that the drift is quiet large. The maximum difference between camera and laser data is just about $0,172$ m in just one case the minimum value was about $-0,0058$ cm. Although the maximum drivt seems quiet high, the final mean of $-0,03$ m shows that the accuracy of our system is quiet good.

4.2 Relative measurements

For the relative test we measured the distance from our system to two different obstacles in different distances to each other. The difference of the two measured distances is our final value:

BARCHART

TABLE

4.3 Obstacle size measurements

In order to measure the size of a reckognizable object we used cardboard circles attached to a translucent plastic rod.

die tests muessen wir nochmal machen, sone tabelle dafuer waere schon echt nice.

5 Discussion

6 Future Work