Distance and Movement Measurement of an Object based on Stereo Images

Lennard Rose, 5110000, FHWS Moritz Zeitler, 5118094, FHWS

Abstract

This paper explains how to compute distance to, and movement of an object within vision of a stereo camera. Included is the whole workflow consisting of all steps that have to be made to get such a recognition to work. These steps are the hardware setup, all calibration steps, object detection, object tracking and the measurements of the values themselves.

I. Introduction

From a high level standpoint the human eye is a very simple complex. It enables us to see things, even in color. But if we go deeper there is much more to our sight. For example we can see depth, which is very complex as a concept. This is based on the human eye having to different point-of-views. In this project the so called Stereo-Vision, will be translated to a computational level. To establish this we use a stereo camera setup to take photos/videos, create a depth map and try to compute the distance and movement of an object. The following chapters will describe this in a step by step fashion, starting with the theory behind it and the implementation afterwards.

II. Workflow

This chapter contains a detailed theoretical and practical description of each individual step in the workflow.

A. Stereo Camera Calibration

Cameras are projecting a three dimensional object to a two dimensional image. This projection is never alone determined by the original object, but also by all parameters of the used camera. These you can separate in two groups, external and internal. External parameters are the camera orientation in different planes. Internal parameters are determined by the cameras internal characteristics.
We can represent these internal parameters as an upper triangular matrix K. This matrix has the values fx and fy as the x

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Fig. 1. Intrinsic Matrix

and y focal lengths. cx and cy as the x and y coordinates of the optical center in the image plane. Gamma is the distortion between the axes, this value is replaced by a ”0” in OpenCV´s calibration function.
The external parameters are represented by an 3x3 rotation matrix, that indicates the rotation bias of the camera, combined

$$\left[R \mid t\right] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]$$

Fig. 2. Extrinsic Matrix

with an 3x1 translation vector.
Together these form the 3x4 Projection Matrix. This matrix and the 3D-coordinates of our point is used to calculate the image coordinates of the 2D projection. Calibration does nothing else than alter these matrices so our 2D projection matches the 3D object. This is usually done by giving the calibrating program/ function a set of 2D images with known 3D coordinates.
This Calibration consists of two steps removing of Distortion as well as rectifying the images[1].

$$\mathbf{P} = \overbrace{\mathbf{K}}^{\text{Intrinsic Matrix}} \times \overbrace{[\mathbf{R} \mid \mathbf{t}]}^{\text{Extrinsic Matrix}}$$

Fig. 3. Projection Matrix

1) Distortion: A camera lens always has some kind of curvature. This curvature is also visible in an image taken with this lens. This is called distortion. There are many different types of distortion which can be seen in the picture beneath. To

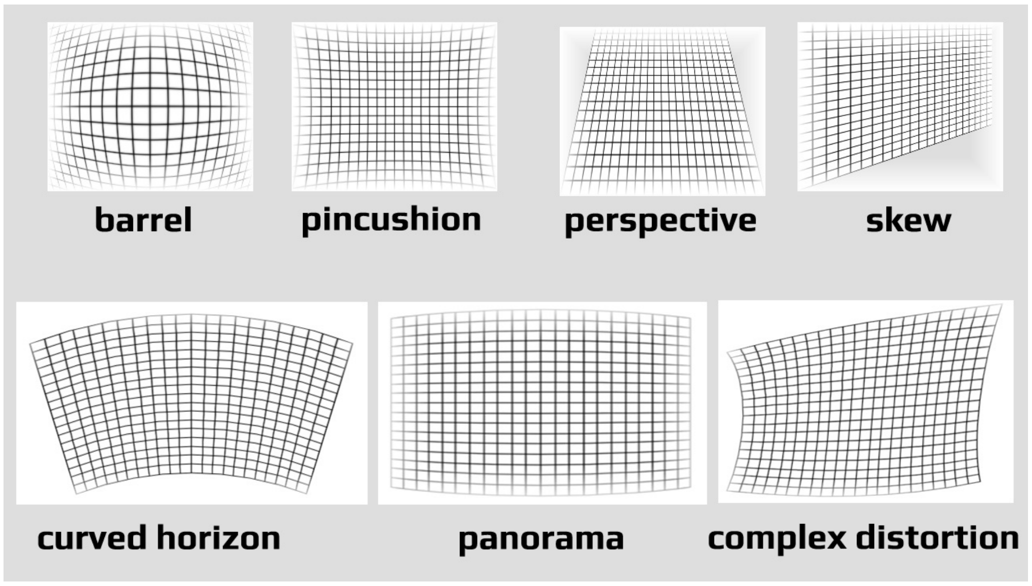

Fig. 4. Distortion types

remove this distortion we take pictures of a chessboard. OpenCV, the package we use to do all calibration steps, can easily detect key points using this chessboard pattern. This is based on the big difference in pixel values between white and black. In combination with the size of the squares as well as the amount of the squares, OpenCV is able to create a calibration matrix removing the distortion.
A very important part is to be very precise while taking the calibration pictures. To get a good result you need to take calibration photos, basically of all parts of the pov of the camera.
The python implementation looks like the following:
First of all we convert the input picture to grayscale and feed it into a detection function for chessboard corners.

```
ret, corners = cv2.findCirclesGrid(inputImage,
(chessboardColumns, chessboardRows),
None)
```

As input we have the chessboard picture, columns of the chessboard, rows of the chessboard and an output array we don't use. This function results in a return value whether a chessboard got detected and if so the 2D points of the chessboard corners. If the detection was successful the returned points get even more refined using:

```
cv2.cornerSubPix(inputImage,
corners,
(11, 11),
(-1, -1),
criteria)
```

The 'cornerSubPix' of openCV refines the corner locations [2] of the checkboard.

2) Rectification: With image rectification, the goal is to project the two different images taken by the camera onto a common image plane. Through this process we make sure that the individual lines of

B. Stereoscopy

C. Disparity Map

D. Depth Map

E. Object Detection

F. Object Tracking

G. Measurements

1) Distance:

H. Direction

I. Speed

III. Prerequisites

IV. Results

V. Conclusion

References

[1] C. Loop and Z. Zhang, “Computing rectifying homographies for stereo vision,” in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 1, 1999, pp. 125–131 Vol. 1.

[2] W. Förstner and E. Gülch, “A fast operator for detection and precise location of distinct points, corners and centres of circular features.”