Project

Depth Information

Lecture *Computer Vision*

Master Computer Science
Faculty for Electrical Engineering and Computer Science
Ravensburg-Weingarten University of Applied Sciences

Author: Bernhard Marconato
Matriculation Nr.: 33444

Submission Date: July 3, 2021
Supervisor: Prof. Dr. rer. nat. Stefan Elser

Contents

1. Task

For the third and final compulsory project of the lecture Computer Vision [5], the dimensions and distance of objects shot by a camera should be estimated. In the Moodle course, two videos are provided, with each containing a chessboard to do the camera calibration. Video 1 also contains a Schwalbe motor roller; video 2 a 3D printer. Additionally, a test video (video 0) is provided which contains the chessboard in known distance of 250 cm to the camera. This video can be used to verify the results of the calculation process.

Using the intrinsic camera matrix and the known size of the chessboard (50x50 cm), the sizes and distances of the real-world objects should be calculated.

Necessary goals of the task as stated in the provided exercise document are: [5]

- Provide the geometrical/mathematical reasoning how you did calculate the distance, width, and height.
- For the distance estimation, follow the approach above by using the two lines calculated with the help of the intrinsic matrix. Provide sketches where needed.
- Calibrate the camera for each video separately.
- Provide your distance estimation for the Schwalbe and the 3D metal printer.
- Provide your width and height estimation for the 3D metal printer.

The theoretical background is covered in the lecture and will not be explained in detail in this project documentation.

2. Implementation

As in the other projects, the camera calibration and object calculations will be implemented using Python and the OpenCV library that offers all necessary functionality to fulfill the task. [6] Additionally, the NumPy library is used for mathematical calculations. [1]

2.1. Calibrating the videos

To begin, the provided videos need to be used to calibrate the camera and obtain the intrinsic camera matrix. In general, the process that will be used has been described in the last project already [7], therefore only a shorter summary will be provided. However, wherever changes in comparison to the last project were made, they will be mentioned.

According to the exercise description, for each video a separate intrinsic matrix should be calculated. [5] This is done by first using the OpenCV function `findChessboardCorners()`, where the chessboard can be detected on each of the video frames. As a chessboard size to be detected by the function, 5x5 internal corners are used because this offered the best detection across the entire video in a short test. With bigger sizes, not enough frames were detected, especially in the parts of the video where the chessboard was moved to the edges of the image - where the important distortion information of the camera is visible.

Due to the large computational effort, not all frames where a chessboard has been detected can be used for the calibration however. Therefore, a selection of 40 frames for each video has been made. Like in the last project, 20 frames have been selected randomly using the Python function `random.sample()`. To further improve the detection results however, additionally 20 frames have been chosen manually, where the chessboard is held in the outer parts of the image - where the distortion of the camera is the strongest. This way, the camera calibration can work with more effective information available to undistort the edges of the image. In testing, this has slightly improved the undistortion, especially for video 1 in the top right corner.

Using the detected chessboard corners in the selected 40 frames and the OpenCV function `calibrateCamera()`, the intrinsic camera matrix of each video can be determined. Afterwards, each video frame can be undistorted with `undistort()` and then saved to the disk for further processing. In comparison to the original project, the intrinsic matrix has not been refined further this time, to avoid information loss in the image due to the cropping. This information is helpful for the next processing step.

To improve the undistortion results even more, unlike in the last project, the calibration is executed a second time, as recommended in the lecture. Therefore the once undistorted frames are taken and used as input for the identical calibration process again. Again, 20 random and the same 20 selected frames where the chessboard has been detected have been chosen for the calibration.

This has improved the undistortion noticeably, as can be seen in Figure 2.1. Especially in the corners, the parallel wall lines are less curved as before, so being better undistorted.

(a) After first calibration (b) After second calibration

Figure 2.1.: Undistortion result of video 1 frame

The amount of detected frames after the first and second calibration did not change significantly, as can be seen in Table 2.1. Around 60 % of all frames were detected in the videos.

Video	Calibr.	Total frames	Detected frames	Detection percentage
0	**I**	671	411	61 %
	II		430	64 %
1	**I**	1056	637	60 %
	II		572	54 %
2	**I**	870	612	70 %
	II		627	72 %

Table 2.1.: Amount of detected frames with 5x5 chessboard in first and second calibration

The resulting intrinsic camera matrices were saved locally as files using the Python tool `pickle` (serialization) for further usage, so they could be loaded afterwards again. The intrinsic matrices for videos 0, 1 and 2 of the second calibration step can be seen in Equation 2.1, Equation 2.2 and Equation 2.3. In theory, they should all be identical assuming the camera was not changed across the recordings. But there are differences up to 10 % in the individual values of the matrices.

The main reason is that the images were undistorted before already, in the two-step process, with differing camera matrices per video. Therefore, the video frames were already undistorted by differing factors. Additionally, the camera may have been changed in between the recordings slightly. Moreover lighting conditions and other error sources could influence the result, which is always only an estimation.

$$M_0 = \begin{pmatrix} 1285.1265 & 0 & 801.0578 \\ 0 & 1284.44665 & 445.72247 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

$$M_1 = \begin{pmatrix} 1160.63588 & 0 & 780.77692 \\ 0 & 1161.66796 & 477.22944 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

$$M_2 = \begin{pmatrix} 1304.50899 & 0 & 763.9381 \\ 0 & 1303.83037 & 474.96704 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

2.2. Determining object dimensions

Next, the task is to determine the size of the 3D printer. It is known the chessboard has the size 50 x 50 cm and is positioned perpendicular to the camera and 3D printer. The chessboard is in the same plane as the 3D printer. Also, an undistorted frame of the video is available where pixel values of the objects can be read from, as can be seen with the red and green lines in Figure 2.2.

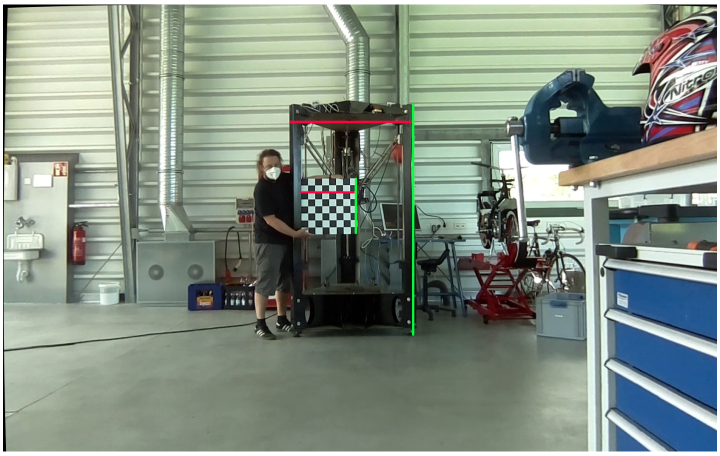

Figure 2.2.: Dimensions of objects in video 2

Using this information, it is possible to calculate the width and height of the printer. First, the pixel values of the chessboard C and printer P width and height need to be read off the image. The determined values can be seen in Equation 2.4 (width) and Equation 2.5 (height), based on the last frame of the undistorted video 2.

$$C^w_{px} = 833 - 700px = 133px \quad P^w_{px} = 968 - 674px = 294px \tag{2.4}$$

$$C^h_{px} = 539 - 406px = 133px \quad P^h_{px} = 778 - 233px = 545px \tag{2.5}$$

Using these values, it is now possible to set up a ratio between pixels and 'real-world' values in cm, as can be seen in Equation 2.6.

$$\frac{C_{cm}}{C_{px}} = \frac{P_{cm}}{P_{px}} \tag{2.6}$$

All values except for P_{cm} are available, which is why the equation can be solved for it, as seen in Equation 2.7, resulting in the real-world size of the printer.

$$\begin{aligned} P_{cm} &= \frac{C_{cm} * P_{px}}{C_{px}} \\ P^w_{cm} &= \frac{50cm * 294px}{133px} = 110cm \\ P^h_{cm} &= \frac{50cm * 545px}{133px} = 205cm \end{aligned} \tag{2.7}$$

The estimation yields that the 3D printer is 110 cm wide and 205 cm high.

2.3. Determining object distances

Now, the distances of the 3D printer and the Schwalbe need to be determined. Using video 0, the approach can be evaluated first because the distance to the chessboard is already known to be 250 cm. Therefore the upcoming calculations should yield a similar result for video 0.

As stated in the project notes [5], it is possible to assign every point $\vec{z}$ in the two-dimensional image plane a line L that maps them to a line in the three-dimensional room using the intrinsic camera matrix K, as seen in Equation 2.8.

$$L = \left\{ \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = dK^{-1} \begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} \middle| \, d \in \mathbb{R}; \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} \in \mathbb{R}^3 \right\} \tag{2.8}$$

The lines "begin" (or more exact, intersect) in the origin of the world coordinate system, which is in this case the camera itself. This determination is sufficient since only the relative distances to the camera are needed, and not the actual placement in the room or any other relative system. To better visualize the problem, in Figure 2.3, the situation of

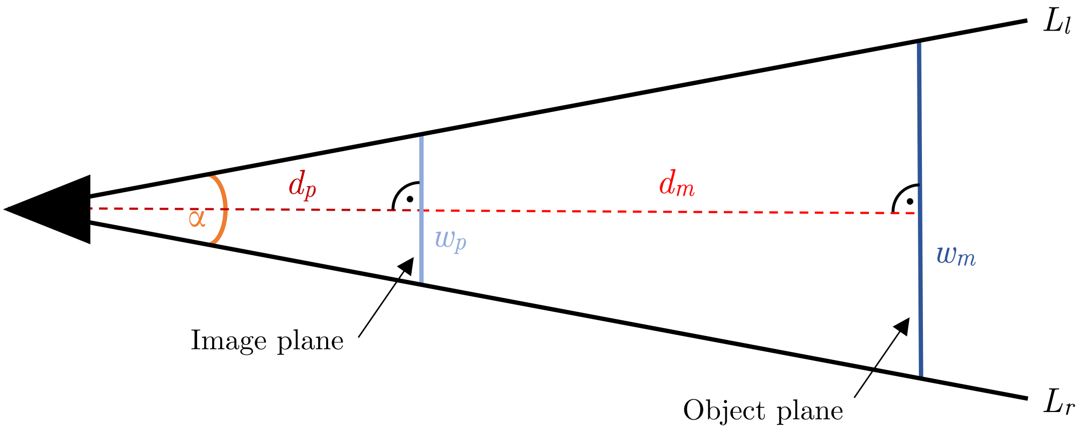

Figure 2.3.: Top-down view on the camera and chessboard in the room

the camera, the image plane and the object plane in the room is visualized in a top-down view.

On the left, the camera sends out the line rays capturing the borders of the chessboard. The image plane w_p shows the 2D image of the chessboard, where the upper point crossing the line L_l is the left side of the chessboard, while the lower point is the right side of the chessboard. The actual chessboard in the three-dimensional room is in further distance and marked with w_m. Both planes are perpendicular to the camera.

The values w_p (width of chessboard in pixels), w_m (width of chessboard in [centi]meters) and the lines L_l and L_r are known. Unknown are the values d_p (distance between camera and image plane in pixels) and d_m (distance between camera and object plane in [centi]meters). d_m is exactly the distance to the chessboard that needs to be found.

Since the real chessboard is placed perpendicular to the camera, it is possible to find two similar triangles in the image - one smaller one between camera and image plane, and one bigger one between camera and the actual object plane. This is simplified in Figure 2.4 with the green and the red triangles.

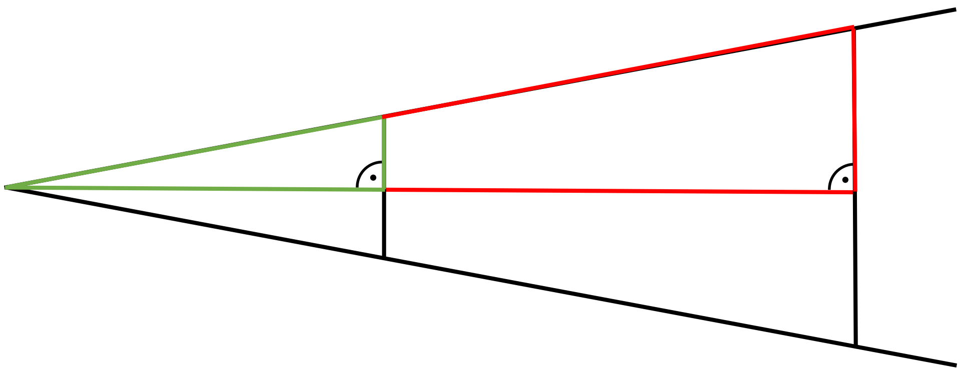

Figure 2.4.: Simplified view of similar triangles

In similar triangles, the ratio between the two adjacent/opposite sides of the triangles is constant. Using this idea, it is possible to define a ratio between the two triangles as in Equation 2.9.

$$\frac{w_p/2}{d_p} = \frac{w_m/2}{d_m} \tag{2.9}$$

As seen in the images, d_m is the actual distance between camera and the chessboard in the “real-world” unit (here centimeters due to $w_m = 50cm$). Therefore, Equation 2.9 can now be solved for the sought d_m, like in Equation 2.10.

$$d_m = \frac{w_m * d_p}{w_p} \tag{2.10}$$

To be able to calculate d_m, the only missing value is d_p. All other values are known to this point. However, this missing value can be determined using trigonometry if the angle α would be known (seen in Figure 2.3). But the angle between the lines is also not an unknown value, since it can be calculated as the angle between the two lines L_l and L_r at the intersection point. This can be done using the dot product of two vectors, here the direction vectors of the lines L_l and L_r. The formula can be seen in Equation 2.11. [4]

$$\alpha = \arccos\left(\frac{L_l \cdot L_r}{\|L_l\|\|L_r\|}\right) \tag{2.11}$$

Using the calculated angle, with the trigonometric function tan the distance d_p can be calculated, as seen in Equation 2.12. [3]

$$d_p = \frac{w_p/2}{\tan(\alpha/2)} = \frac{w_p}{2\tan(\alpha/2)} \tag{2.12}$$

Equation 2.9 and Equation 2.12 can now be put together and simplified, to be able to calculate the actual distance d_m like in Equation 2.13.

$$d_m = \frac{w_m}{2\tan(\alpha/2)} \tag{2.13}$$

With this formula, it is possible to calculate the distance estimation for each video using its intrinsic camera matrix, two image points $\vec{z_1}$ and $\vec{z_2}$ of chessboard and the real-world width of the chessboard $w_m = 50cm$. The upcoming calculations have been made using the NumPy library for Python.

Video 0 (Reference)

$$\begin{aligned} \vec{z_1} &= \begin{pmatrix} 668 & 499 & 1 \end{pmatrix}^T \qquad & \alpha &= 0.1891218436272016 \\ \vec{z_2} &= \begin{pmatrix} 912 & 499 & 1 \end{pmatrix}^T \qquad & d_m &= 263.59cm \end{aligned} \tag{2.14}$$

The error compared to the expected distance value of $250cm$ is $\approx 5\%$.

Video 1 (Schwalbe)

$$\vec{z_1} = \begin{pmatrix}691 & 439 & 1\end{pmatrix}^T \qquad \alpha = 0.16825217564450048$$
$$\vec{z_2} = \begin{pmatrix}886 & 420 & 1\end{pmatrix}^T \qquad d_m = 296.47cm \tag{2.15}$$

Video 2 (3D Printer)

$$\vec{z_1} = \begin{pmatrix}700 & 410 & 1\end{pmatrix}^T \qquad \alpha = 0.10173949799169739$$
$$\vec{z_2} = \begin{pmatrix}833 & 410 & 1\end{pmatrix}^T \qquad d_m = 491.03cm \tag{2.16}$$

As the intrinsic matrixes have been calculated for each video only, they should not be interchangeable between the videos. Regardless, out of curiosity, the matrices of video 1 and 2 have also been used for the estimation of the distance in video 0. For matrix 1, this has yielded a value of $237.89cm$ (error $\approx 5\%$), while for matrix 2 the value $267.47cm$ (error $\approx 7\%$) has been determined.

As the error for all values is relatively similar, it can be expected the matrices for the other videos also seem to be accurate enough for video 1 amd video 2. But without knowing the actual distances in video 1 and 2, it is not possible to evaluate these results further.

3. Conclusion

The low error rate of around 5 % shows that it is generally possible to estimate distances relative to the camera without detailed knowledge of the camera sensor - only with camera calibration using a chessboard with known size. While this is a working approach and very cost-effective, it can be prone to errors if the calibration and the intrinsic camera matrix are not accurate enough. Therefore, it is important to have high-quality calibration videos to reduce the error as much as possible.

The precondition of having a chessboard perpendicular to the camera cannot be fulfilled in every case, however. Also the results of the camera calibration approach can vary significantly, as discovered in lecture discussions.

That's why for more serious applications like autonomous driving, more advanced distance estimation methods should be researched. Following a similar approach, the OpenCV method `solvePnP()` could be used for the pose estimation of the chessboard. According to the documentation, this method can even be used with a non-perpendicular chessboard to estimate distance and other pose information of the chessboard. [8] Additionally, calculations could be simplified if the focal length of the camera sensor would be available.

As covered in the lecture, the problem could also be solved with more (advanced) hardware - by using two cameras for example. Even more expensive solutions like using Lidar sensors, which are already built into current iPhone and iPad Pro models and can be used in the preinstalled "Measure" app, could also be a possibility. [2]

A. List of Figures

B. List of Tables

C. Bibliography

[1] Travis E. Oliphant et al. *numpy*. Python Package Index. 2021-06-22. URL: https://pypi.org/project/numpy/ (visited on 2021-06-23).

[2] Apple. *Use the Measure app on your iPhone, iPad, or iPod touch*. URL: https://support.apple.com/en-us/HT208924 (visited on 2021-07-01).

[3] Paul Dawkins. *Trig Cheat Sheet*. 2005. URL: https://tutorial.math.lamar.edu/pdf/Trig_Cheat_Sheet.pdf (visited on 2021-06-29).

[4] Paul Dawkins. *Calculus II - Dot Product*. 2021-06-14. URL: https://tutorial.math.lamar.edu/classes/calcii/dotproduct.aspx (visited on 2021-06-29).

[5] Stefan Elser. *Computer Vision Compulsory Projects*. 2021-06-16. (Visited on 2020-06-26).

[6] Olli-Pekka Heinisuo. *opencv-python*. Python Package Index. 2021-06-07. URL: https://pypi.org/project/opencv-python/ (visited on 2021-06-24).

[7] Bernhard Marconato. *Project: Camera Calibration*. 2021-05-28.

[8] OpenCV. *OpenCV: Camera calibration with square chessboard*. URL: https://docs.opencv.org/3.4/dc/d43/tutorial_camera_calibration_square_chess.html (visited on 2021-06-29).