ORB-SLAM2 Report

Nikolaos Kourtzanidis

January 4, 2021

1 Introduction

In this example we run a custom rgb video sequence on a state of the art, feature based, monocular SLAM algorithm called ORB-SLAM2 [1]. OpenCV, which is a library of programming functions mainly aimed at real-time computer vision, was used for Camera calibration and to convert the video sequence into individual corresponding frame images. We begin by walking through the step by step procedures required to reproduce the results. The camera used for the implementation was the main back camera of the Samsung Galaxy S7 Edge.

2 Camera Calibration

The following implementations were performed on a computer with, Intel Core i7-6700 CPU @ 3.40GHz, and 31 GB of memory, and Ubuntu 20.04.1 LTS (Focal Fossa) operating system. To execute a monocular rgb sequence the command shown in Figure 1 must be executed in terminal. You should be in the ORB-SLAM2 directory for this command to work. The ./Examples/Monocular/mono_tum is the path to the executable. The Examples/Monocular/TUMX.yaml points to a .yaml file containing the settings such as the cameras intrinsic parameters and distortion coefficients.

```
./Examples/Monocular/mono_tum Vocabulary/ORBvoc.txt
Examples/Monocular/TUMX.yaml PATH_TO_SEQUENCE_FOLDER
```

Figure 1: Execution command for a monocular rgb example. [2]

The PATH_TO_SEQUENCE_FOLDER should consist of a folder named "rgb" which contains the images used for the video sequence, and a rgb.txt file, which is a text file and contains the corresponding order of the picture frames using their timestamps as shown in Figure 2.

The first step required is to determine these parameters specifically for the camera that is going to be used. Cameras typically introduce significant distortion to images. Two major distortions are: Radial and Tangential. The

```
# color images
# file: 'rgbd_dataset_freiburg1_xyz.bag'
# timestamp filename
1305031102.175304 rgb/1305031102.175304.png
1305031102.211214 rgb/1305031102.211214.png
1305031102.243211 rgb/1305031102.243211.png
1305031102.275326 rgb/1305031102.275326.png
1305031102.311267 rgb/1305031102.311267.png
1305031102.343233 rgb/1305031102.343233.png
1305031102.375329 rgb/1305031102.375329.png
1305031102.411258 rgb/1305031102.411258.png
1305031102.443271 rgb/1305031102.443271.png
1305031102.475318 rgb/1305031102.475318.png
1305031102.511219 rgb/1305031102.511219.png
1305031102.543220 rgb/1305031102.543220.png
1305031102.575286 rgb/1305031102.575286.png
1305031102.611233 rgb/1305031102.611233.png
1305031102.643265 rgb/1305031102.643265.png
1305031102.675285 rgb/1305031102.675285.png
1305031102.711263 rgb/1305031102.711263.png
```

Figure 2: Example of an rgb textfile. [2]

distortion caused by the optical lens itself is called radial distortion. The camera lens makes a straight line become a curve in an image, and it becomes more clear when getting closer to the border of the image [3]. Radial Distortion can be corrected by using a polynomial function:

$$\begin{aligned} x_c &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_c &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned} \tag{1}$$

Where $(x, y)^T$ is the coordinate before correction, $(x_c, y_c)^T$ is the coordinate after correction, and $r = x^2 + y^2$. Tangential Distortion is mainly caused by the optical lens not being parallel with image formation plane during camera assembling [3]. The parameters p1 and p2 can be used for correction given:

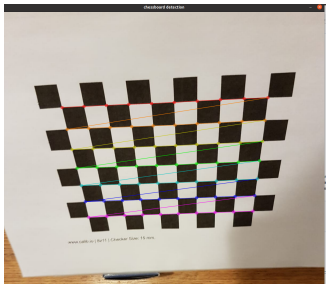
$$\begin{aligned} x_c &= x + 2p_1 xy + p_2\left(r^2 + 2x^2\right) \\ y_c &= y + p_1\left(r^2 + 2y^2\right) + 2p_2 xy \end{aligned} \tag{2}$$

$$\text{Distortion coefficients } = \begin{pmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{pmatrix} \tag{3}$$

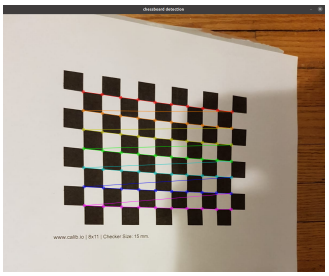
The extrinsic parameters of a camera are used to transform the points of an object in world frame to camera frame and usually represented by to a translation vector and a rotational matrix. The intrinsic parameters are used to project the points from the image plane to the pixel plane coordinates after the distortion corrections are applied and take the form of a 3x3 matrix:

$$\text{Camera Matrix } = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

These parameters are required to be input into the TUMx.yaml file used in the command for execution. The parameters were determined by using functions



(a) Image 1.



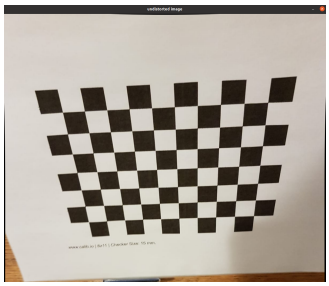
(b) Image 2.

Figure 3: Two checkerboard images before the distortion corrections are applied and a pattern was found.

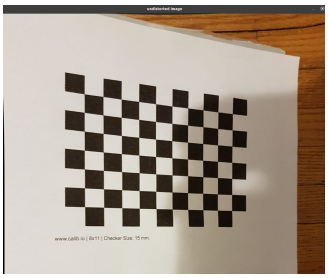
from OpenCV. For the camera calibration program, we must provide some sample images of a well defined pattern, for example, a chessboard. We find some specific points of which we already know the relative positions (e.g. square corners in the chess board). The pattern used in this example was an 8x11 checkerboard with each square having a dimension of 15mm as shown in Figure 3. A custom checkerboard pattern of your choice can be printed by using the link: `https://calib.io/pages/camera-calibration-pattern-generator`.

We load several images of the checkerboard in different orientations and positions. In the code we provide a 2D matrix of the coordinate location in the 3D world frame, by ignoring the Z coordinate, assuming the checkerboard only moves within the XY plane. The function cv::findChessboardCorners is used to determine whether or not the pattern that we input to the program (in this case an 8x11 grid) was found. This function returns a boolean true or false, and if a pattern is found the corresponding 2D image coordinates are returned by using a conditional "if" statement and the cv::cornerSubPix function. In the example code provided 218 images were used for calibration. The cv::drawChessboardCorners function, was used to display the pattern on the images provided, if found, as shown in Figure 3.

Once we can determine the corresponding image coordinates from the pattern found, and we already know the dimensions of our checkerboard (3D world space coordinates) , we can solve for the distortion coefficients and camera parameters (intrinsic and extrinsic). For the solving step, the cv::calibrateCamera function was used for each image coordinates found from the pattern, and the 2D world coordinates provided to return the extrinsic and intrinsic camera parameters, the distortion coefficients, and the overall RMS re-projection error. For 218 images this function took approximately 26 minutes to execute. The extrinsic parameters were used to re-project each image and apply the distortion corrections to measure the norm distance between the original image from the corner finding algorithm and the transformed image. The cv::remap function was used to provide the newly transformed images which can be displayed as shown in Figure 4. Figure 5 shows the Camera matrix , Distortion coefficients



(a) Image 1.



(b) Image 2.

Figure 4: Two checkerboard images after the distortion corrections and corresponding transformations are applied.

and RMS re-projection error obtained.

```
Reprojection error = 1.02493
K =
[1799.9097, 0, 539.5;
 0, 1799.9097, 959.5;
 0, 0, 1]
k=
[0.307483, -0.666124, 0, 0, 0]
```

Figure 5: Camera matrix, Distortion coefficients and RMS re-projection error obtained.

3 Video To Files

Referring back to the execution command, in Figure 1, the "PATH_TO_SEQUENCE_FOLDER" should consist of a folder named "rgb" which contains the images used for the video sequence, and a rgb.txt file, which is a text file that contains the corresponding order of the picture frames using their timestamps as shown in Figure 2. Each image was named by the corresponding timestamp in seconds, that the frame was captured from the beginning of the video until the end.

A python script was used to perform this function of converting a video to the corresponding sequence of images for the hallway video and also to convert the video of the checkerboard movements into images that were used for the camera calibration procedure, as mentioned in the previous section. The OpenCV libraries were imported, and the script was ran by executing the following command in terminal:python Videotofiles.py videopath imagefolder

orb-slam2. Where videopath is the path to the directory which contains the .mp4 video sequence and imagefolder is the path to the directory where the rgb folder and rgb.txt file should be returned in, which should be replaced by "PATH_TO_SEQUENCE_FOLDER" argument in the ORB-SLAM2 command when attempting to perform execution on the custom video sequence.

4 Results

After the calibration procedure is finished and we have segmented our custom video sequence into their corresponding image frames, we must just modify the TUMx.yaml file and then we are in the position to execute the ORB-SLAM2 monocular example command, and view the point cloud map of our video sequence and the camera poses. The TUM7.yaml file was modified in my case according to the calibration results obtained in the previous section as shown in Figure 6. An snippet of the ORB-SLAM2 algorithm running the customized video sequence is shown in Figure 7. When attempting to run various custom sequences, the INITIALIZING portion would sometimes take very long, and in some instances would never even pass this INITIALIZING phase. This problem was resolved by changing the default ORBextractor.nFeatures: 1000 to 3000. Lastly, the features of the camera used in this implementation are shown in Figure 8.

```
# Camera calibration and distortion parameters (OpenCV)
Camera.fx: 1799.9097
Camera.fy: 1799.9097
Camera.cx: 539.5
Camera.cy: 959.5

Camera.k1: 0.307483
Camera.k2: -0.666124
Camera.p1: 0.0
Camera.p2: 0.0

# Camera frames per second
Camera.fps: 30.0

# Color order of the images (0: BGR, 1: RGB. It is ignored if images are grayscale)
Camera.RGB: 1

#--------------------------------------------------------------------------------------------
# ORB Parameters
#--------------------------------------------------------------------------------------------

# ORB Extractor: Number of features per image
ORBextractor.nFeatures: 3000

# ORB Extractor: Scale factor between levels in the scale pyramid
ORBextractor.scaleFactor: 1.2

# ORB Extractor: Number of levels in the scale pyramid
ORBextractor.nLevels: 8

# ORB Extractor: Fast threshold
# Image is divided in a grid. At each cell FAST are extracted imposing a minimum response.
# Firstly we impose iniThFAST. If no corners are detected we impose a lower value minThFAST
# You can lower these values if your images have low contrast
ORBextractor.iniThFAST: 20
ORBextractor.minThFAST: 7

#--------------------------------------------------------------------------------------------
# Viewer Parameters
#--------------------------------------------------------------------------------------------
Viewer.KeyFrameSize: 0.05
Viewer.KeyFrameLineWidth: 1
Viewer.GraphLineWidth: 0.9
Viewer.PointSize:2
Viewer.CameraSize: 0.08
Viewer.CameraLineWidth: 3
Viewer.ViewpointX: 0
Viewer.ViewpointY: -0.7
Viewer.ViewpointZ: -1.8
Viewer.ViewpointF: 500
```

Figure 6: Example of TUM7 file modified including Camera matrix, Distortion coefficients obtained.

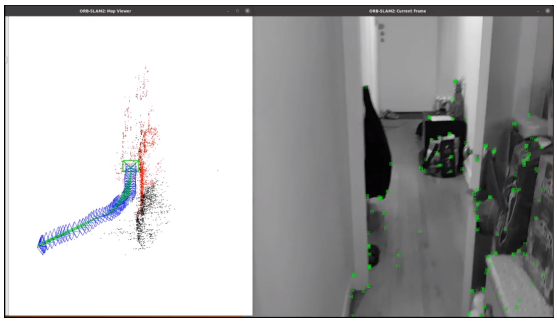


Figure 7: Snippet of ORB-SLAM2 algorithm running a custom video sequence.

Camera	Samsung Galaxy S7 Edge
Back camera (main camera)	✔ 12.2 megapixels
Main camera resolution	4032x3024 pixels
Video recording (primary)	✔ 4K UHD (3840x2160) 30 fps
Flash	✔ LED flash
Focal aperture	f/1.7 (aperture)
Focal length	26mm lens
Sensor size	1/2.5" inches
Pixel size	1.4µm pixel
Autofocus	✔ PDAF: phase detection autofocus
Touch focus	✔ Supported
Image stabilization	✔ OIS: Optical stabilization
Zoom	✔ Only digital zoom
Face/smile detection	✔ Face detection, Smile detection
BSI sensor	✔ Supported
HDR	✔ HDR photo/video on both cameras

Figure 8: Camera features.

References

[1] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[2] J. Zijlmans, "Orb-slam2: Implementation on my ubuntu 16.04 with ros kinect," https://medium.com/@j.zijlmans/orb-slam-2052515bd84c.

[3] Y. Chen and G. Wang, "Bundle adjustment revisited," *ArXiv*, vol. abs/1912.03858, 2019.