Tutorial 1: Camera Calibration

ME 555.14 Advanced Robotics

February 29, 2024

1 Objectives of the Tutorial

Upon completion of this tutorial, one should ideally be able to -

1. Understand the basics of camera calibration.
2. Implement one popular method of calibration (Python with OpenCV).
3. Troubleshoot errors and differentiate between successful and unsuccessful calibrations.
4. Be confident in calibrating other cameras in the future (in preparation for the end-semester project and after completing the Adv. Robotics course)

2 What is Camera Calibration?

Camera calibration is the process of estimating the parameters of a camera using images of a known pattern. To do so, 3D world points and their corresponding 2D image points need to be estimated.

It involves determining the camera's intrinsic and extrinsic parameters which describe the camera's internal characteristics, such as its focal length and lens distortion, as well as its position and orientation in space.

2.1 Camera Parameters

- **Internal Camera Parameters:** Intrinsic calibration determines the camera's internal parameters - focal length, optical center, and lens distortion coefficients. These parameters are used to model and correct the lens distortion and convert between image coordinates and normalized camera coordinates.
- **External Camera Parameters:** Extrinsic calibration establishes the camera's position and orientation in relation to a known scene or world coordinate system – represented by a rotation matrix and a translation vector. These tell us where the camera is located in the given space and at what angle it is pointing, which is essential for placing the camera within a broader context, such as the map of a room or a robot's coordinate system.

These parameters are used to correct for lens distortion, measure the size of an object in world units, or determine the location of the camera in the scene.

3 Camera Pinhole Model

A great explanation of the world coordinate system, camera coordinate system and the mathematis behind the pinhole model can be found here!

4 Zhengyou Zhang's Method of Calibration

The implementation of OpenCV's `cameraCalibraton` method is based on Zhengyou Zhang's paper "A Flexible New Technique for Camera Calibration". The technique only requires a camera to observe a planar pattern from at least 2-3 different orientations. This can be done by either moving the camera or the planar pattern, without the motion being known. The procedure consists of a closed-form solution, followed by a nonlinear refinement based on the maximum likelihood criterion. This approach is notable for its practicality and the ease with which it can be implemented, making it accessible for various applications in computer vision and image processing.

One aspect to note from the results of this paper is that the relative error of the camera's calibration reduces as the number of images being used increases. However, between images 2 to 3, the error reduces significantly, and any images added after that contribute comparatively little to the overall reduction of error.

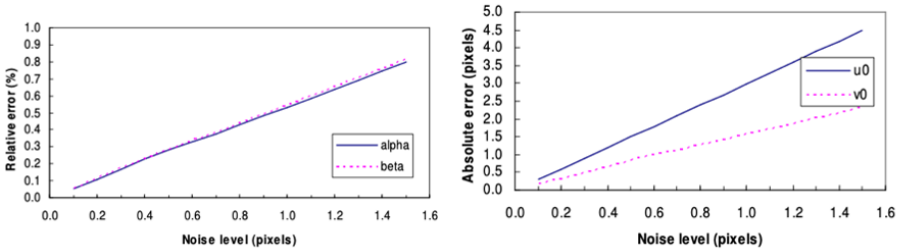

Figure 1: Errors vs. the noise level of the image points

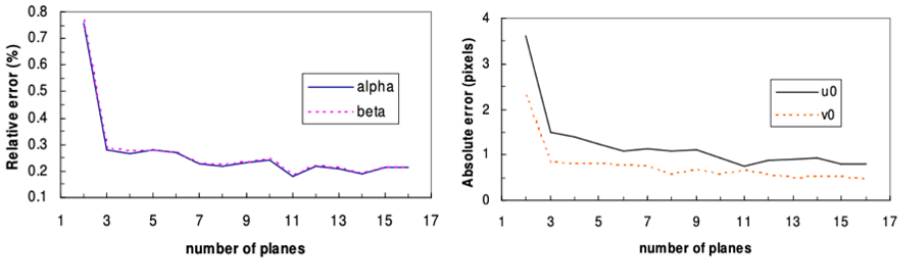

Figure 2: Errors vs. the number of images of the model plane

For a detailed exploration of this method, you can review the paper here.

5 Camera Calibration in python using OpenCV

One of the most widely methods of calibrating a camera is to use a checkerboard pattern. This is because it provides a clear, high-contrast set of points that can be easily detected in images. Also, the corners of squares on the checkerboard are ideal for localization and serve as fixed points in space.

The code for completing steps 1-4 has been provided in the addendum of this tutorial, with helpful comments for easy implementation.

5.1 Step 1: Generate a checkerboard pattern of known size

Use any of the following links to generate checkerboard patterns of different sizes, based on the size of your printing paper. You can choose to print this pattern out or use your computer/iPad to display the image. A rigid structure is preferred because there is no chance of distortion/bending.

1. Calibration Checkerboard Collection
2. ARToolkit

Note the numbers of rows and columns and the dimensions of the squares. We'll be using that to define the checkerboard in the code.

5.2 Step 2: Capture multiple pictures of the checkerboard

Place the checkerboard on a suitable wall or surface and use the camera (that needs calibrating) to capture images from various angles and depths.

Once the images are generated, save them in a folder within the same working directory as your code and make note of the image's pixel dimensions. Save the file path.

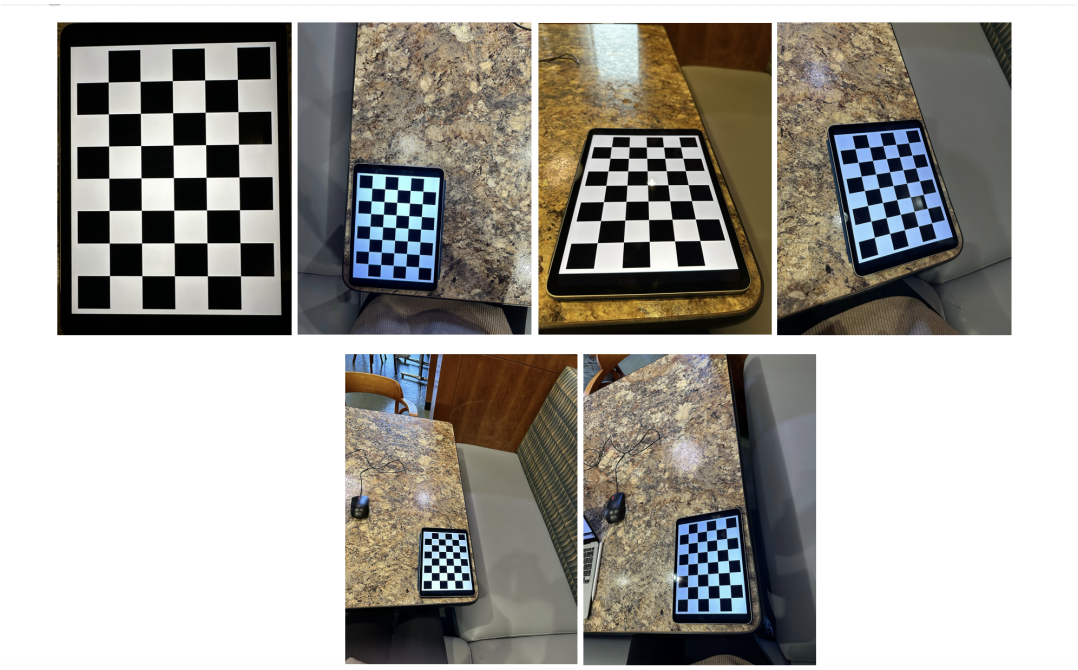

Figure 1: Partial image set used for calibration

Note:

- You could choose to keep the camera still in a stationary location and move the checkerboard around, while it captures images at certain time intervals (or) place the checkerboard in a fixed location and move the camera around. Either way, the calibration process remains the same.
- You can experiment with the number of images you use and how it affects the relative calibration error and computation time.

5.3 Step 3: Find the checkerboard corners

The OpenCV function `findChessboardCorners` is used to detect and return the pixel coordinates (u,v) for each 3D point in different images. It is important to note that this function detects the inner corners of the pattern, and hence the *patternSize* should be defined as [(rows-1), (columns-1)].

For example: For a board containing 10 rows and 8 columns, the *patternSize* to be detected by the functions would be 9 points per row and 7 points per column.

The output of this function is Boolean True/False depending on whether corners were detected in the image or not.

```
retval, corners = cv2.findChessboardCorners(image, patternSize, flags)
```

Inputs	
`image`	Input image
`patternSize`	Number of inner corners per a chessboard row and column (patternSize = cvSize (points_per_row, points_per_column) = cvSize(columns, rows)).
`flags`	Various operation flags. You have to worry about these only when things do not work well. Go with the default.
Outputs	
`corners`	Output array of detected corners.

Documentation for this function can be found here: OpenCV

5.4 Step 4: Refine checkerboard corners

In order to obtain the location of the corners with sub-pixel level accuracy, the OpenCV function `cornerSubPix` is used. The function takes the original image with the location of the corners found in `findChessboardCorners` and looks for the best corner locations within a smaller sub-area. Since this function is iterative, a termination criterion needs to be specified in terms of number of iterations or accuracy.

```
cv2.cornerSubPix(image, corners, winSize, zeroZone, criteria)
```

Inputs	
`image`	Input image
`corners`	Initial coordinates of the input corners and refined coordinates provided for output.
`winSize`	Half of the side length of the search window.
`zeroZone`	Half of the size of the dead region in the middle of the search zone over which the summation in the formula below is not done. It is used sometimes to avoid possible singularities of the auto-correlation matrix. The value of (-1,-1) indicates that there is no such a size.
`criteria`	Criteria for termination of the iterative process of corner refinement. That is, the process of corner position refinement stops either after `criteria.maxCount` iterations or when the corner position moves by less than criteria.epsilon on some iteration.

5.5 Step 5: Calibrate the camera

The final step of the calibration process is to use the function `calibrateCamera` and pass the images' 3D points in world coordinates and their 2D locations. This will find the camera's parameters, 3D points, and pixel coordinates.

```
retval, cameraMatrix, distCoeffs, rvecs, tvecs = cv2.calibrateCamera(objectPoints, imagePoints, imageSize)
```

Inputs	
`objectPoints`	A vector containing vectors of 3D points
`imagePoints`	A vector containing vectors of 2D points
`imageSize`	Size of the image
Outputs	
`cameraMatrix`	**Camera Matrix:** A 3x3 matrix that contains the intrinsic parameters of the camera. It includes the focal lengths (fx, fy) along the X and Y axes (in pixels) and the optical centers (cx, cy) of the camera (also in pixels). $$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$
`distCoeffs`	**Distortion Coefficients:** An array that contains the parameters of distortion caused by the camera lens. It usually has five, sometimes more, parameters: k1, k2, p1, p2, k3, with k parameters modeling radial distortion and p parameters modeling tangential distortion.
`rvecs`	**Rotation Vectors:** For each image, this list contains a rotation vector that, together with a translation vector, maps the points from the 3D world coordinate system to the 2D coordinate system of the image. The rotation vector can be converted to a rotation matrix or Euler angles if needed.
`tvecs`	**Translation Vector:** Similar to the rotation vectors, for each image, this list contains a translation vector that represents the translation of the camera relative to the world coordinates.

Note that the camera matrix and distortion coefficients are intrinsic parameters of the camera, and do not change. However, the rotation vector and translation vector are recorded for each image.

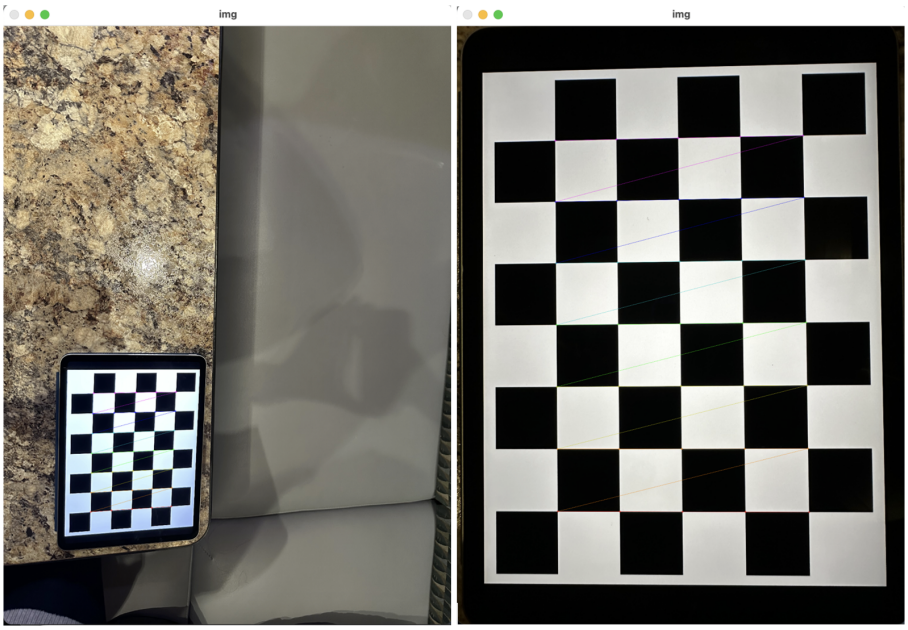

Figure 2: Result of the Refinement process will generate images with the detected corners and lines connecting them. Depending on the pixel resolution of your image, these points may be hard to see.

6 What's Next?

The next steps include -

1. Select one image from your folder of checkerboards that are oriented in an angle and generate an undistorted image of the same.
2. Visually inspect the before and after images to see if your camera was calibrated successfully and if the expected output was obtained.
3. Calculate the re-calibration error. Justify why this error is good/bad. If your error is higher than expected, troubleshoot what went wrong and try calibrating the camera again.

7 Instructions for the Exercise

This tutorial was created using an iPhone camera. In real world scenarios (and in the case of the final project) you will be using other cameras (for example: Intel RealSense RGB-D Camera with the UR5e).

For this exercise, use any live-feed camera (ESP-32, RealSense, a webcam, a laptop camera, etc...) fixed in one location and move the checkerboard pattern around.
You are also free to any other calibration pattern of your choice -

- Symmetrical circular grid pattern
- Asymmetrical circular grip pattern
- Customized patterns

... and a calibration method of your choice (for example: MATLAB Camera Calibrator, Mrcal Calibration).

7.1 Deliverables:

1. How many images did you use for camera calibration? Share 3-4 images of the detected corners from various angles.
2. Share one distorted image (before) and the corresponding undistorted image (after). If you encountered some bizarre outputs - share those images and explain what went wrong with it and how you fixed the issue.
3. What was your re-projection error? Justify whether it's good/bad.
4. Share your code (with appropriate comments) and the output (camera parameters).

8 Cited work

1. Camera Calibration OpenCV
2. A flexible new technique for camera calibration - Zhengyou Zhang.
3. Geometry of Image Formation
