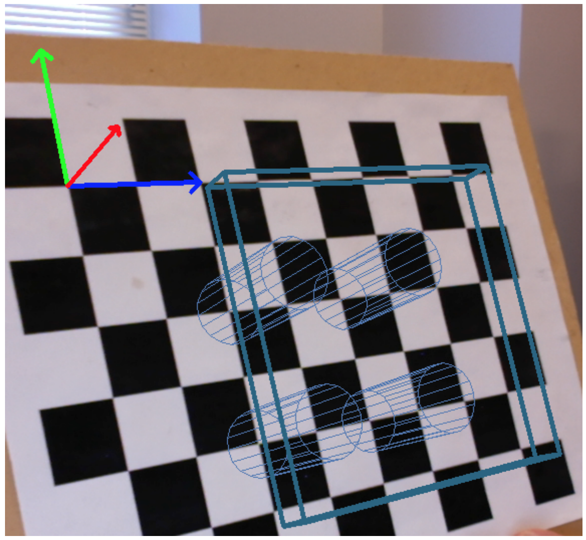
Project 4: Calibration and Augmented Reality

Due Mar 25 by 8:59pm **Points** 0



Overview

This project is about learning how to calibrate a camera and then use the calibration to generate virtual objects in a scene. The end result should be a program that can detect a target and then place a virtual object in the scene relative to the target that moves and orients itself correctly given motion of the camera or target.

Setup

Use this image **checkerboard.png** as a target. You can display it on a screen, such as a tablet or cell phone, or print it out on paper. This checkerboard pattern, when viewed with the long axis horizontally, has 9 columns and 6 rows of internal corners. You will want to use a pattern like this for both the calibration and VR parts of this project. When creating a coordinate system, it is easiest to consider the squares as being 1 x 1 and define any virtual objects using those units. There are alternative patterns available, such as the AR toolkit, that will also work.

Tasks

1. **Detect and Extract Chessboard Corners**

   The first task is to build a system for detecting a target and extracting target corners. Get this working cleanly before you move on to the calibration task.

The [calibration documentation (https://docs.opencv.org/master/d4/d94/tutorial_camera_calibration.html)](https://docs.opencv.org/master/d4/d94/tutorial_camera_calibration.html) for OpenCV is fairly good. The relevant functions for the first task are [findChessboardCorners (https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a)](https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a), [cornerSubPix (https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga354e0d7c86d0d9da75de9b9701a9a87e)](https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga354e0d7c86d0d9da75de9b9701a9a87e), and [drawChessboardCorners (https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga6a10b0bb120c4907e5eabbcd22319022)](https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga6a10b0bb120c4907e5eabbcd22319022).

For the corners argument, I suggest using a std::vector. If you have the definition:

```
std::vector<cv::Point2f> corner_set;
```

Then the number of corners is given by corner_set.size(), and the corner coordinates for the ith entry in the vector are corner_set[i].x and corner_set[i].y.

Have your video draw the chessboard corners when it finds them. It can also be helpful to print out how many corners it finds, along with the coordinates of the first corner. Note that the first corner will generally be in the upper left of the checkerboard as viewed in the image.

2. **Select Calibration Images**

The next step is to let the user specify that a particular image should be used for the calibration and save the corner locations and the corresponding 3D world points. For example, if the user types 's', then store the vector of corners found by findChessbordCorners into a corner_list. At the same time, create a point_set that specifies the 3D position of the corners in world coordinates. You probably want to use the following definitions for the corner_list, point_set, and point_list.

```
std::vector<cv::Vec3f> point_set;
std::vector<std::vector<cv::Vec3f> > point_list;
std::vector<std::vector<cv::Point2f> > corner_list;
```

To build a point set you can either try to get specific and figure out the size of a checkerboard square in mm, or you can just measure the world in units of checkerboard squares and give the corners coordinates of (0, 0, 0), (1, 0, 0), (2, 0, 0), and so on. Note that if the (0, 0, 0) point is in the upper left corner, then the first point on the next row will be (0, -1, 0) if the Z-axis comes towards the viewer.

There need to be as many points in the point_set as there are corners in the corner_set. You also need as many point_sets in the point_list as there are corner_sets in the corner_list.

You may want to store the images themselves that are being used for a calibration. **Include a calibration image with chessboard corners highlighted in your project report.**

3. **Calibrate the Camera**

If the user has selected enough calibration frames--require at least 5--then let the user run a calibration. Alternatively, you could continuously update the calibration each time the user adds a new calibration image (beyond some minimum number), telling the user the current per pixel error after each calibration.

Use the cv::**calibrateCamera (https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d)** function to generate the calibration. The parameters to the function are the point_list, corner_list (definitions above), the size of the calibration images, the camera_matrix, the distortion_coefficients, the rotations, and the translations. The function also has arguments for information about how well each parameter was estimated. You may want to use the flag CV_CALIB_FIX_ASPECT_RATIO, which specifies that the pixels are assumed to be square (not a bad assumption these days). Radial distortion is optional, and you may want to have it turned off at first. For cell phone cameras, though, it may be necessary to get a sufficiently low reprojection error.

You will have already generated the point_list and corner_list vectors. Make the camera_matrix a 3x3 cv::Mat of type CV_64FC1. You probably want to initialize the 3x3 camera_matrix to something like.

```
[1, 0, frame.cols/2]
[0, 1, frame.rows/2]
[0, 0, 1           ]
```

Print out the camera matrix and distortion coefficients after the calibration, along with the final re-projection error. The two focal lengths should be the same value, and the u0, v0 values should be close to the initial estimates of the center of the image. Your error should be less than a half-pixel, if everything is working well. **Include the error estimate in your report.**

The calibrateCamera function also returns the rotations and translations associated with each calibration image. If you saved the calibration images, you might want to also save these rotations and translations with them.

Enable the user to write out the intrinsic parameters to a file: both the camera_matrix and the distortion_ceofficients.

4. **Calculate Current Position of the Camera**

   For the remaining tasks you can write a completely separate program or continue to enhance your original one.

   Write a program that reads the camera calibration parameters, then starts a video loop. For each frame, it tries to detect a chessboard. If found, it grabs the locations of the corners, and then uses **solvePNP (https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d)** to get the board's pose (rotation and translation).

   Have your program print out the rotation and translation data in real time, as you are testing this task.

5. **Project Outside Corners or 3D Axes**

   Given the pose estimated in the prior step, have your program use the **projectPoints (https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c)** function to project the 3D points corresponding to the four outside corners of the chessboard onto the image plane in real time as the chessboard or camera moves around. Alternatively, put 3D axes on the board attached to the origin. **Include at least one image from this step in your report.**

6. **Create a Virtual Object**

   Construct a virtual object in 3D world space made out of lines. Then project that virtual object to the image and draw the lines in the image. Make sure the object stays in the right orientaiton as the camera moves around. Note, you may need to create an asymmetrical target in order for this to work properly from all angles. You may want to create an asymmetrical virtual object to aid in debugging.

   Have fun. **Take some screen shots and/or videos of your system in action for your project report.**

7. **Detect Robust Features**

   Pick a feature (e.g. Harris corners) and write a separate program that shows where the features are in the image in a video stream. Make a pattern of your choice and show where those features show up on the pattern. In your report, explain how you might be able to use those feature points as the basis for putting augmented reality into the image.

   Take some screen shots and/or videos of the system working.

Extensions

- Especially creative virtual objects and scenes are a good extension. It is possible to integrate OpenCV with OpenGL to render shaded objects (big extension).
- Check out the site **ArUco (http://www.uco.es/investiga/grupos/ava/node/26)**, which has a nice set of targets and code to detect them using OpenCV. Integrate these into your system.
- Get your system working with multiple targets in the scene.
- Test out several different cameras and compare the calibrations and quality of the results.
- Get your AR system working with a target other than the checkerboard, like a photo, painting, or object of your choice that is not a checkerboard. Make it something where it is easy to find 3-4 points on the object and determine which point is which. Place an AR object somehwere in the world reletive to the object.
- Enable your system to use static images or pre-captured video sequences with targets and demonstrate inserting virtual objects into the scenes.
- Not only add a virtual object, but also do something to the target to make it not look like a target any more.

Report

When you are done with your project, write a short report on the **Khoury wiki (https://wiki.khoury.northeastern.edu)** that demonstrates the functionality of each task. You will need to adjust the permissions of your home space and your report wiki pages so that other people can see it (by default, no one but you can view them). Your report should have the following structure. Please **do not** include code in your report.

1. A short description of the overall project in your own words. (200 words or less)
2. Any required images along with a short description of the meaning of the image.

3. A description and example images of any extensions.
4. A short reflection of what you learned.
5. Acknowledgement of any materials or people you consulted for the assignment.

Submission

When you are ready to submit, upload your code and a readme.txt (or readme.md) text file to **Gradescope (https://www.gradescope.com)** . The readme file should contain the following information.

- Links/URLs to any videos you created and want to submit as part of your report.
- The URL for your wiki report for this project.
- What operating system and IDE you used to run and compile your code.
- Instructions for running your executables.
- Instructions for testing any extensions you completed.
- Whether received an accommodation for the project or you are using one or more time travel days.

For project 4, submit your .cpp and .h (.hpp) files, and readme.txt (readme.md). Note, if you find any errors or need to update your code, you can resubmit as many times as you wish up until the deadline.

As noted in the syllabus, projects submitted by the deadline can receive full credit for the base project and extensions. (max 30/30). Projects submitted up to a week after the deadline can receive full credit for the base project, but not extensions (max 26/30). You also have eight time travel days you can use during the semester to adjust any deadline, using up to three days on any one assignment (no fractional days). If you need to make use of the "stuff happens" clause of the syllabus, contact the instructor as soon as possible to make alternative arrangements.

Receiving grades and feedback

After your project has been graded, you can find your grade and feedback on Gradescope. Pay attention to the feedback, because it will probably help you do better on your next assignment.