Calibration and Augmented Reality

Rahul Kumar
Northeastern University

1. SUMMARY

The purpose of this project is to implement Augmented Reality using camera calibration with a known target image and generating virtual objects in a scene in real-time. It is based on calculating the camera's intrinsic and extrinsic parameters and utilizing them to transform the given 3D world coordinates to the 2D pixel coordinate to project a virtual object in the image. To calibrate the camera, a chessboard pattern is used as a known target. After finding the intrinsic parameters, extrinsic parameters are calculated for each frame, and 3D to 2D transformation is done using the transformation matrix.

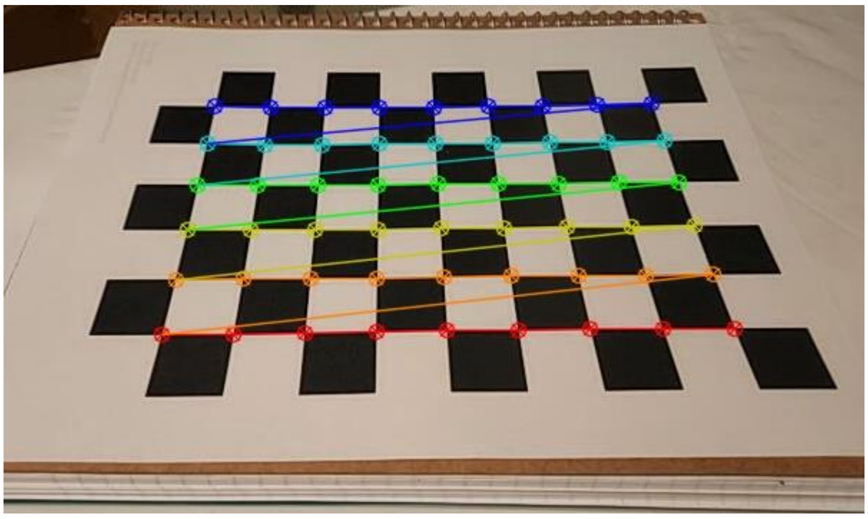
2. SETUP

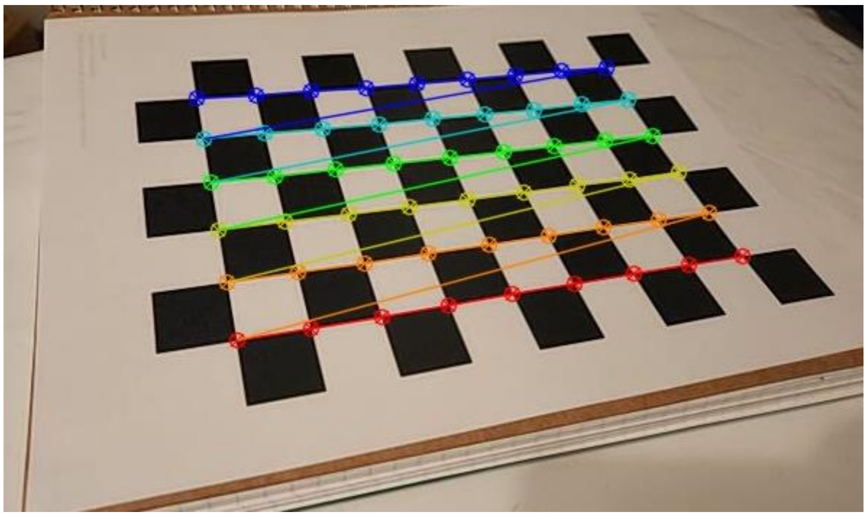
A known target i.e. chessboard pattern is used for camera calibration. The chessboard pattern has 9 columns and 6 rows of internal corners thus 54 corners are used in the calculation. One unit of world coordinate is set as equal to the side length of one square chessboard. The real-time video is streamed with the mobile phone camera, which is pointing towards the target.

3. CAMERA CALIBRATION

3.1. Detecting Chessboard and Extracting corners

The very first step in camera calibration is to detect the target, which is a chessboard pattern, and extract the corners as feature points of the target. To detect the corners in a chessboard, 'findChessboardCorners', which is an OpenCV function, is used. This function determines the presence of a chessboard in an image and if found, it provides the approximate position of the internal corners of the chessboard. To get the accurate coordinates, 'cornerSubPix' is used, which provides the sub-pixel accurate location of corners. Finally, lines are drawn connecting corners using 'drawChessboardCorners'. The below images show the extracted corners of the chessboard.





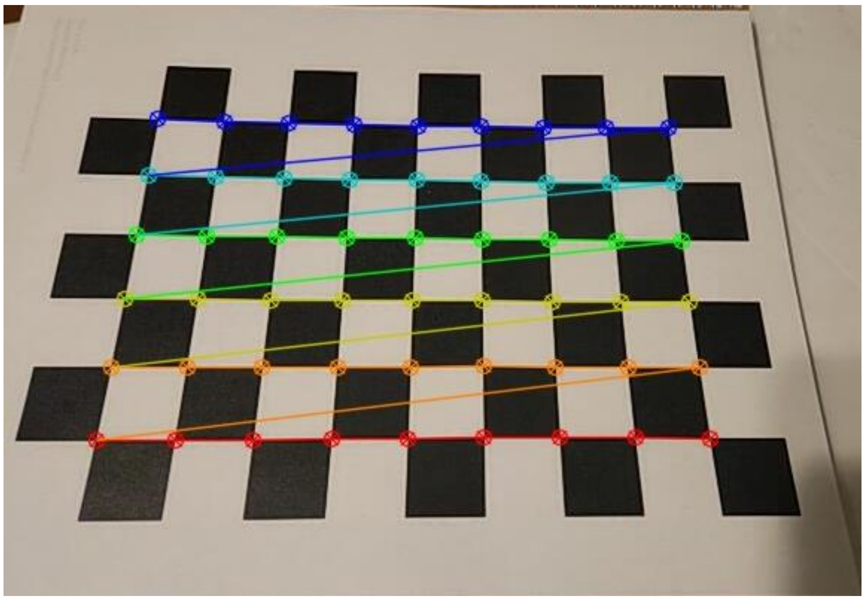


Fig. 1. Calibration Images

3.2. Calibrate camera

After extracting the corners, to calibrate the camera, multiple calibration images are required along with correspondence between pixel and world coordinates. Since the target is known, the top left corner is selected as the origin for the world coordinate and one unit along the axis is kept equal to the side of the square on the chessboard. Now, multiple images of the target are taken at different orientations, and pixel coordinates of corners in each image are saved. The pixel coordinates and corresponding world coordinates are passed onto the 'calibrateCamera' function which returns the camera's intrinsic parameters. In this project, Samsung Galaxy S22 is used for calibration, and below are the results of the camera calibration.

Intrinsic camera matrix:

$$\begin{pmatrix} fx & 0 & u \\ 0 & fy & v \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 455 & 0 & 312 \\ 0 & 455 & 244 \\ 0 & 0 & 1 \end{pmatrix}$$

Distortion coefficient:

$$(k1 \quad k2 \quad p1 \quad p2 \quad k3) = (0.074 \quad -0.37 \quad 0.00006 \quad -0.0017 \quad 0.4)$$

Reprojection error = 0.12 pixels

4. AUGMENTED REALITY

4.1. 3D Axis projection

Once camera calibration is completed, the camera matrix along with the distortion coefficient is found. But with a change in the pose of the camera, the extrinsic camera parameters are computed for each pose. So, for each new frame, 'solvePNP' is used to calculate the camera pose i.e. rotation and translation matrix given the camera intrinsic parameters and 2D-3D correspondence (correspondence between chessboard corner pixel coordinates and corners world coordinates). The intrinsic camera matrix and extrinsic matrix form a transformation matrix which is used to project any world coordinates to the pixel coordinates. In applying this concept, the 3D axis is projected on the image at the origin. The below image shows the projection.

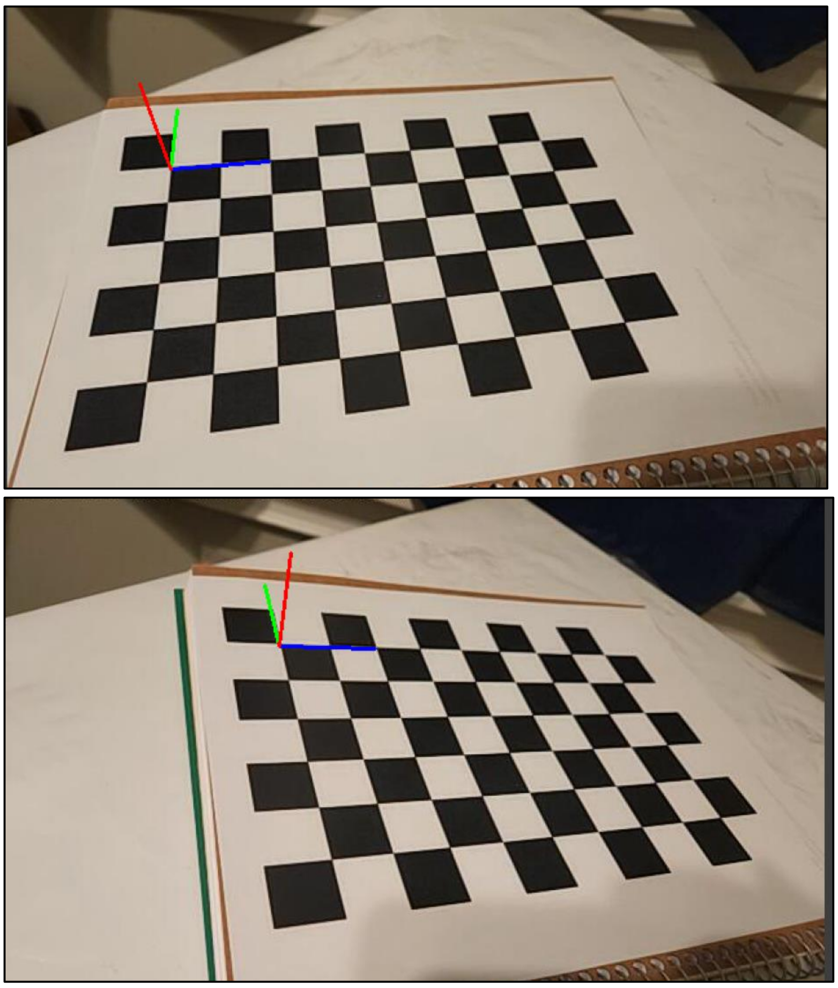


Fig. 2. 3D axis projection on the image

4.2. Virtual object projection

As a virtual object, I tried to form a robot on the image. It is attained by projecting corner points of robots on the image and connecting each corner to form a wireframe of the robot. The below image shows the projection of a virtual object i.e. robot.

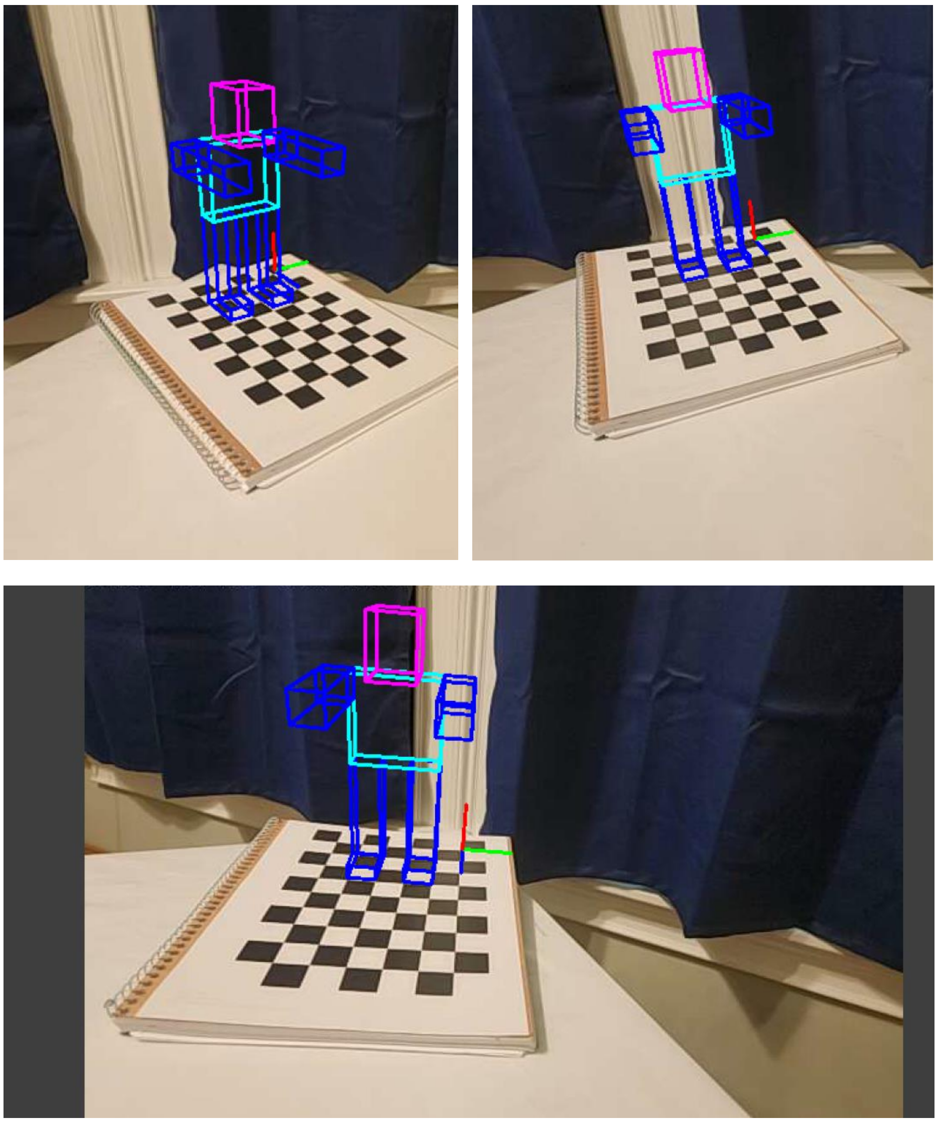


Fig. 3. Virtual object projection on the image

It is implemented such that for every camera pose, the projection does not change and remains at the same position with respect to the origin. A real-time video is recorded which can be found at the below location which shows the stability of projection even in the case of changing camera pose.

https://drive.google.com/file/d/1KCg4I6mhURJqHn97EiUTl9rEgfWufU53/view?usp=sharing

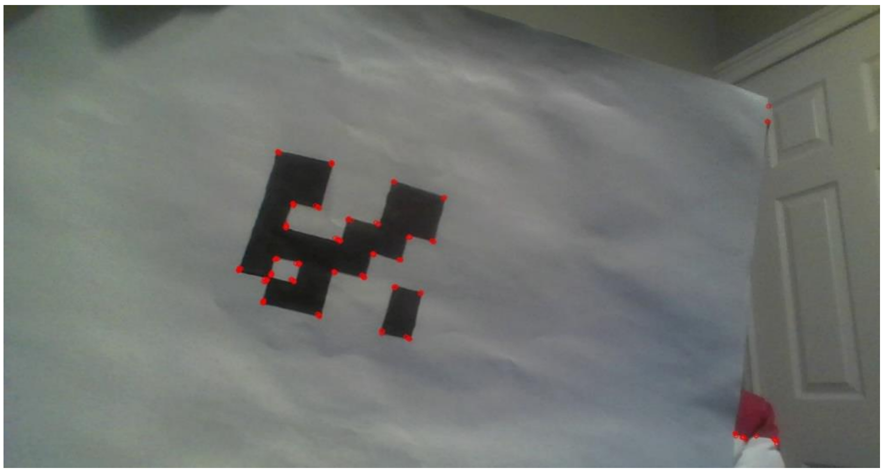
As an extension, I tried to give the effect of flying to the robot, and a video is recorded that shows the flight of the robot in the z-axis (perpendicular to the paper). The video link is below:

https://drive.google.com/file/d/1Ipoqah3Y-KShaykOpLqO8n1sadRxS-wh/view?usp=sharing

5. ORB FEATURE DETECTION

Instead of using a chessboard pattern, a random pattern is used and, in this case, features are detected using ORB (Oriented Fast and Rotated BRIEF) feature detector. ORB works on the FAST keypoint detector and BRIEF descriptor. FAST computes the keypoints by comparing the brightness of any given pixel to surrounding pixels in a circle around that pixel and if more than half of the surrounding pixels are brighter/darker than the center, then the central pixel is selected as a keypoint. It is made scale invariant and rotation invariant using a multiscale image pyramid and least moment axis with intensity centroid. Then BRIEF is used to generate the descriptor about each keypoints. The below images show the ORB feature detected in a random pattern.

For Augmented reality in the image, ORB features can be used as a known point in the different frames containing this known pattern. Then these features are matched using descriptors in 2 frames and based on the best matches, the Fundamental matrix can be calculated. Once the Fundamental matrix is known, then the Essential matrix is calculated and then the depth of the pattern in the image is estimated. Now we have world coordinates and corresponding pixel coordinates, using these we can solve for extrinsic matrix in each frame, given the intrinsic parameters are known and we can project virtual objects into the image.



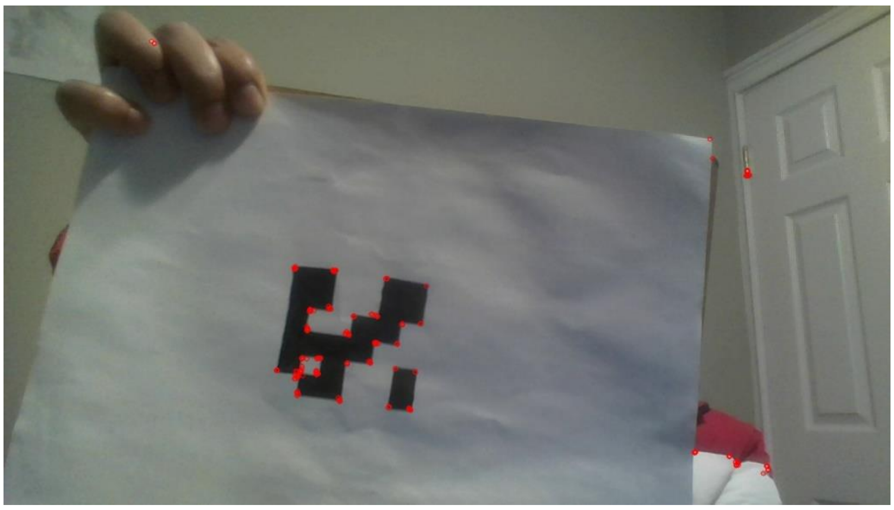


Fig. 4. ORB Feature detection

6. EXTENSION (FACE AUGMENTATION)

In the extension part, I tried something different and implemented Face augmentation (similar to Snapchat) using the Haar cascade classifier. In this implementation, the objective is to track the human face and eye and augment 2D sunglasses in real-time. For this, OpenCV 'CascadeClassifier' class is used to detect faces and eyes. Sunglass image is downloaded from the internet in .png format and using its alpha channel, a mask is created which is used while augmenting sunglasses on the face. Below is the link to the video which demonstrates the face augmentation in real-time.

https://drive.google.com/file/d/1tW-9KuiomNLzTYe3ehq607cMXVs2hawB/view?usp=sharing

7. LEARNING

After completing this project, I learned about camera calibration using a chessboard pattern and the implementation of virtual reality. In-camera calibration, I learned to calculate intrinsic and extrinsic parameters which can then be used to project 3D world points on the image plane. In Augmented reality, I experimented with projecting a robot in an image and provided the motion to it. As an extension, I tried face augmentation and successfully implemented the Haar features to detect the face and eye and augment sunglasses on the face in real-time.

ACKNOWLEDGEMENT

1. https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html
2. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a
3. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d
4. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d
5. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c
6. https://docs.opencv.org/3.4/db/d95/classcv_1_1ORB.html
7. https://github.com/javier-arango/opencv-face-detection
8. https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html