PYTHON PROJECT REPORT

AIM

To create simple augmented reality using OpenCV and Aruco.

INTRODUCTION

Augmented reality (AR) is an interactive experience of a real-world environment where the objects that reside in the real world are enhanced by computer-generated perceptual information, sometimes across multiple sensory modalities, including visual, auditory, haptic, somatosensory and olfactory. AR can be defined as a system that fullfills three basic features: a combination of real and virtual worlds, real-time interaction, and accurate 3D registration of virtual and real objects.

ArUco markers have been used for a while in augmented reality, camera pose estimation, and camera calibration. ArUco markers were originally developed in 2014 by S.Garrido-Jurado et al., in their work Automatic generation and detection of highly reliable fiducial markers under occlusion. ArUco stands for Augmented Reality University of Cordoba. An aruco marker is a *fiducial marker* that is placed on the object or scene being imaged. It is a binary square with black background and boundaries and a white generated pattern within it that uniquely identifies it. The black boundary helps making their detection easier. They can be generated in a variety of sizes. The size is chosen based on the object size and the scene, for a successful detection. If very small markers are not being detected, just increasing their size can make their detection easier.

So my idea is to make use of this Aruco markers to create a simple Augmented reality. Libraries used for this are OpenCV and Numpy.

I have attached all the python code files along with this file.

EXPLANATION

For projecting the points in the real world to 2D on our screen we need to calibrate our camera. This is the important step to be followed. We know that all camera has distortion effect and this increases as one moves away from the centre to sides of the frame. This distortion is not noticeable but in projects which require high precision and accuracy we need to remove this distortion so that our result will be as expected in all situations

Removing Camera Distrotion

First, We need to get some real world points and compare it with the 2D points that we get in the frame. For this purpose I used checker board. OpenCV already has functions for finding the corners in the checkerboard pattern here I used 8*8 square checkerboard(Chess board). I took 28 images of checkerboard from my camera in each image the checkerboard is held at different positions and different angles. 2D points can be obtained directly from the image using OpenCV. Now, for the 3D points in the real world I took the plane of the board as X-Y plane and Z is perpendicular to the board. So that all the real world points on the checker board will be of the form (x,y,0). I assign all the corners on the chessboard with it's corresponding coordinates on the real world (I took each side of the square on the chessboard as 1 unit length).

One of the image which has been used for the project, after finding all the corners is as shown in figure 1. It is seen that it has successfully detected all the inside corners. The corners are detected in order shown by the lines starting from the corners marked as red.

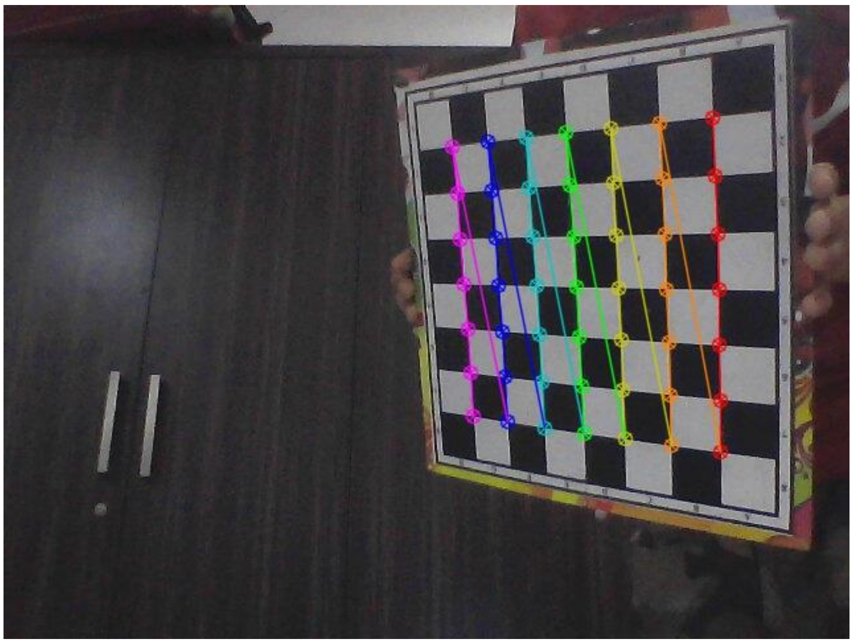


Figure 1

We give input in form of images taken from my camera with chessboard aligned at different angles to the function cv2.findChessBoardCorners(). It returns all the corner points in the image. Now after getting all the 2D points and it's corresponding 3D points we can calibrate the camera using the function cv2.calibrateCamera(). This function takes object points (Real world points) and image points (2D points) and returns the intrinsic camera matrix, distortion coefficients of the camera. It also returns translation vector and rotational vector for each of the images. The intrinsic camera matrix and the distortion coefficients are the one we are interested in. This can be stored in local files so that we can use these whenever we need. These values are different for different cameras.

Projecting the Points

Now after calibrating the camera, we can use the obtained camera matrix and distortion coefficients to project points in the real world to 2D points on the frame. For this we need to have some reference points on the real world.

We can use Aruco markers as reference. OpenCV has inbuilt function which can be used to detect corner points of the aruco marker we can thus use these points as reference. We will pass the detected corner points on the frame (2D points) and also the corresponding 3D points. As I said before, 3D points are obtained by taking the plane of aruco marker as the X-Y plane so that our 3D points can be calculated easily as they will have the form (x,y,0). I took each of length of the aruco marker as 1 unit. The 3D points is as shown in the figure below. Remember that the corner array returned by the aruco.detectCorners() is in order from top-left corner moving clockwise to other corners.

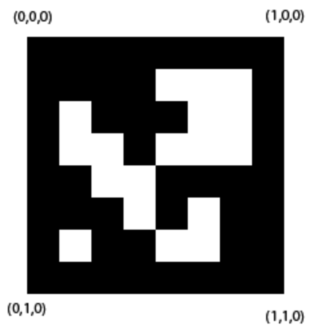


Figure 2: Aruco Marker

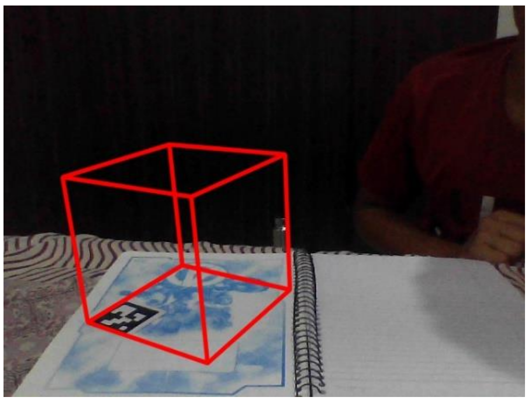
After getting 2D points and its corresponding 3D points we can pass these to the function cv2.solvePnPRansac() this function will return the rotational and translation vector of the points in the 3D world. This can then be used to project any points in the real world on the frame so that the user will have a feeling that the point is situated in the real world. We can pass the obtained rotational and translational vectors as input to the function cv2.projectPoints() along with the points in the 3D as arguments to get its corresponding projected points on the frame (i.e. image points).

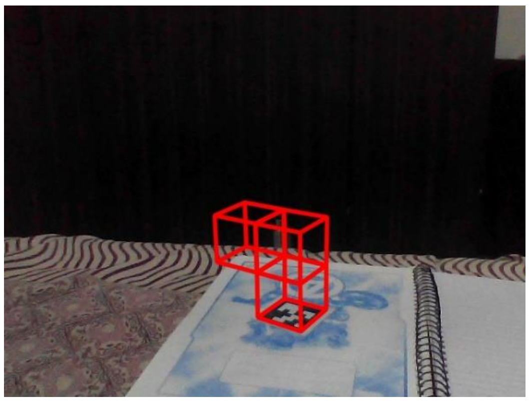
CameraPoseEstimation.py contains the code for the estimation of camera to get camera matrix and distortion coefficients.

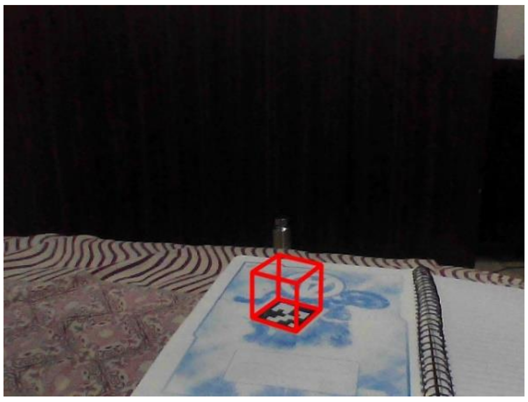
More Explanation is given as comments in the code files.

The sample outputs is as shown below,

Note: version of OpenCV used in the project is 4.1.2.3 There might be some problem if same version of OpenCV is not used.


a)


b)


c)

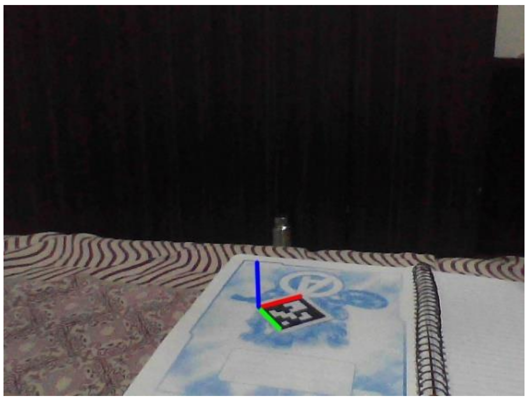

d)

Figure 3: Sample Outputs

Figure 3 d) is the coordinated axis projected onto the frame red line corresponds to X-axis, green line corresponds to Y-axis and blue line corresponds to Z-axis. Figure 3 a, b, c are some 3D objects drawn on the frame. Every 3D object rotates or translates when the aruco marker is changed and we can see the corresponding results in live video.

Note: The z axis is actually going into the plane so I flipped the z axis so that it comes pointing out of the plane and I changed z to -z in every draw functions so that user can give input as if the z axis is pointing out of the plane.

In ArucoAR.py file I only wrote functions to draw a cube, rectangle and the axis ,description of each function is given in the file. You can write additional functions to draw some complex objects or use the given functions to create some interesting objects eg: Figure 2 b) is created by calling drawCube() three time with different starting point. The entire images used for the project is attached with this file.

CONCLUSION

The program works perfectly in all cases irrespective of the orientation of the Aruco marker. If better camera or more sample images are given as input for calibration we can increase the accuracy of the projections. This project implements the basic AR using OpenCV in python. This finds uses in real world applications such as teaching especially during online lectures. You can see an application of it in the lectures of BYJU'S app.

--The End---

Submitted by:
Aswin Raj K (121801008)