Assignment 3 - Augmented Reality

Krishanu Das Baksi 2019ANZ8274

Deepak Raina 2019MEZ8497

COL-780 Computer Vision
October 23, 2019

Approach:

1. **Camera Calibration:**
   The task is to find intrinsic and extrinsic parameters of the webcam. We have used a checkerboard of 3x5 grid having a size of 30 mm. Assuming chessboard at Z=0 plane, the object points can simply be (0,0,0), (30,0,0), (60,0,0)..., which denotes the location of points in 3D space. To find image points in chessboard, we have used the function, `cv2.findChessboardCorners()`. Using object points and image points, the function, `cv2.calibrateCamera()` is used to calibrate the camera. This function returns the camera matrix, distortion coefficients, rotation and translation vectors.

2. **Estimating pose of the camera:**
   In this task, two markers (as shown in Fig. 1) have been selected that can be tracked in a video input from a webcam.

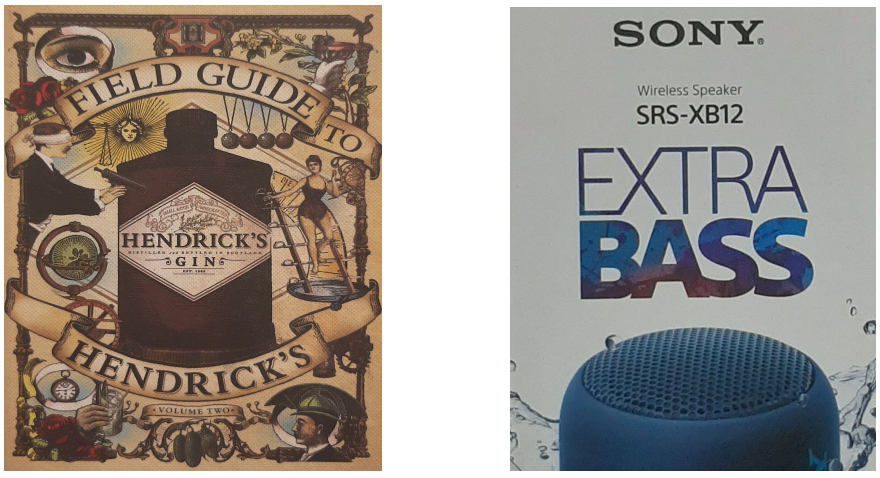

Fig. 1. Visual markers 1 and 2

For tracking the visual markers, first, the keypoints have been extracted using SURF/SIFT for both the marker image and video frame. Every pair and their corresponding key points were taken and used to compute the matches among the

keypoints. The image features are matched using `cv::FlannBasedMatcher` interface in order to perform a quick and efficient matching. The good feature matches have been identified using Lowe's criteria. Using these image pairs, we estimate the homography using `cv2.findHomography`.

In order to smooth out the fluctuations in the homography estimation, the exponentially weighted smoothing of the homography matrix is done in the consecutive frames. This is done to ensure some temporal relationship among the different matrices obtained. This was done by updating the current matrix as the weighted average of the matrix (estimated in the current frame) with the estimated matrix in the previous frame. Once the homography has been estimated, the bounding box has been drawn around the marker in the video stream.

Using the estimated homography and camera calibration matrix, the camera pose has been estimated. The camera pose with respect to a marker is the 3d transformation from the marker coordinate system to the camera coordinate system. It is specified by a rotation and a translation vector. The logic of calculating the camera projection matrix has been taken from [1, 2]. This matrix will have 3x4 dimension, having 3x3 rotation matrix and 3x1 translation vector.

3. **Rendering a 3D-Model:**

To render a 3D model, first, the 3d object is downloaded from online sources and loaded using Pygame library. Once the model is loaded, a function has been written to read this data and projects it on top of the video frame with the projection matrix that we obtained in the previous section. To do so we take every point used to define the model and multiply it by the projection matrix. After this, the faces of the model are filled with color. The feature matching and 3D model rendering using the selected markers is shown in the figure below. Please note that only few of the matches (best 30) are shown in the frame.

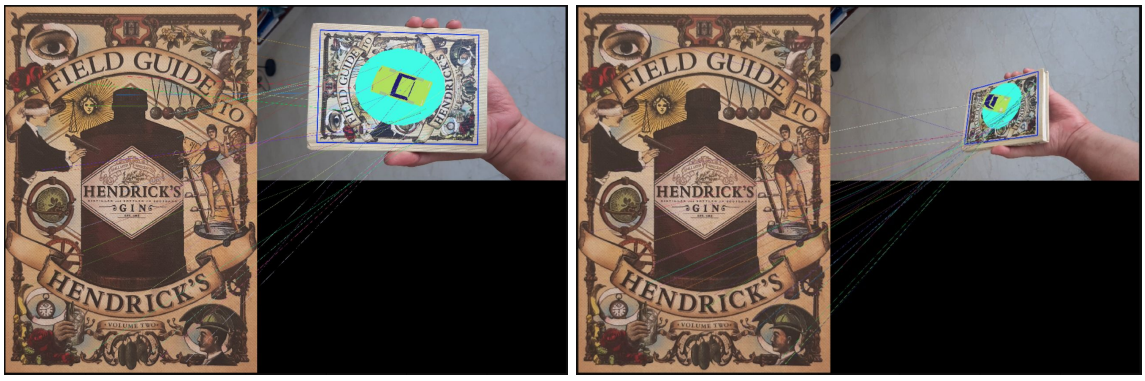

Fig. 2: Feature matching and 3D model rendering on a video stream

4. **Moving an augmented object:**

To move a 3D object, the center of two different markers is projected onto the camera coordinate system using the world/marker to camera projection matrix. Now the task is to move the 3D object from one marker to another marker. The object is rendered relative to the center of the second marker, by using the following formulation.

Let O_1 and O_2 be the projected centers of markers in the camera coordinate. Let pos_{obj} is the current position of 3D object with respect to center O_2 in the camera coordinate system. Initially the value of pos_{obj} is [0, 0, 0]. The distance vector of the object from O_1 in the current frame is then given by

$$d = O_1 - O_2 - pos_{obj}$$

The 3D object needs to move along the direction of distance vector in the camera coordinate. The unit step of its movement is defined by

$$s = \frac{d}{k}$$

where k is the parameter controlling the speed of movement. Larger the value of k, slower will be the movement and vice-versa. Once the object reaches the first marker, the distance will be zero. So, it's position will remain static. The new position of the object is then given by

$$pos_{new} = pos_{obj} + s$$

Once new position of 3D object with respect to O_2 is obtained in the camera coordinates, all the positions of the vertices of the object in the camera coordinate are added to the new position. The object is then rendered with respect to the center of O_2. So effectively, we get the positions of all vertex points of the object after proper translation in the camera coordinate. They are then rendered in a similar manner at the corresponding position in the image coordinates as described in section 3. Once the object has reached the marker position 1 i.e O_1, d will go down to zero and hence the movement stops.

5. **How to run the code**

Question 1: python cam_calibration.py image_location/foldername

Question 3: python renderer1.py marker_locaton/filename video_location/filename object_location/filename output_locaton/filename feature_detector(surf/sift)

Question 4: python renderer1.py marker_locaton/filename marker2_location/filename video_location/filename object_location/filename output_locaton/filename

Output:

The link of the output videos is as given below:
<videos>

References

[1]https://bitesofcode.wordpress.com/2017/09/12/augmented-reality-with-python-and-opencv-part-1/
[2]https://bitesofcode.wordpress.com/2018/09/16/augmented-reality-with-python-and-opencv-part-2/