Computer Vision - Augmented Reality Part 2:

Assumptions:

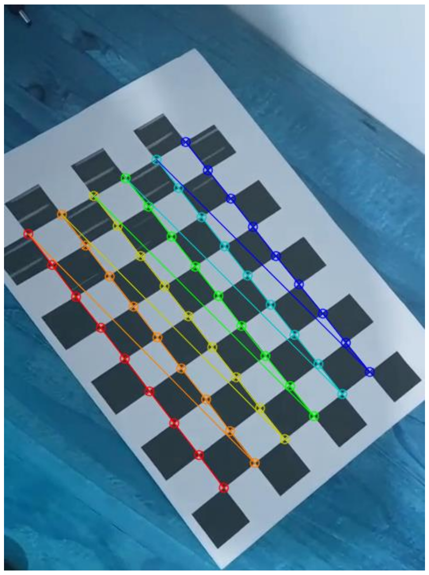
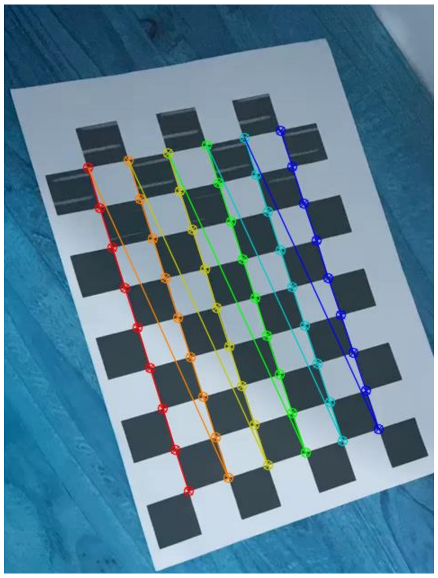
1. We printed a chessboard and used it to find the camera matrix, distortion coefficients rotation vector and translation vector, after which we saved them for later calibrations.
2. We used the first part of the project to find the homographic matrix and the mask, using the same picture and video from before.
3. Using the homographic matrix and mask we found the key points that are the same in the reference picture and frame picture.
4. We used cv2.SolvePnP with the camera matrix and distortion coefficients to find the rotation vector and translation vector of the camera to map the real world points to the pixels of the frame.
5. We render the mesh object using the rotation vector and translation vector.

On each step I will further elaborate:

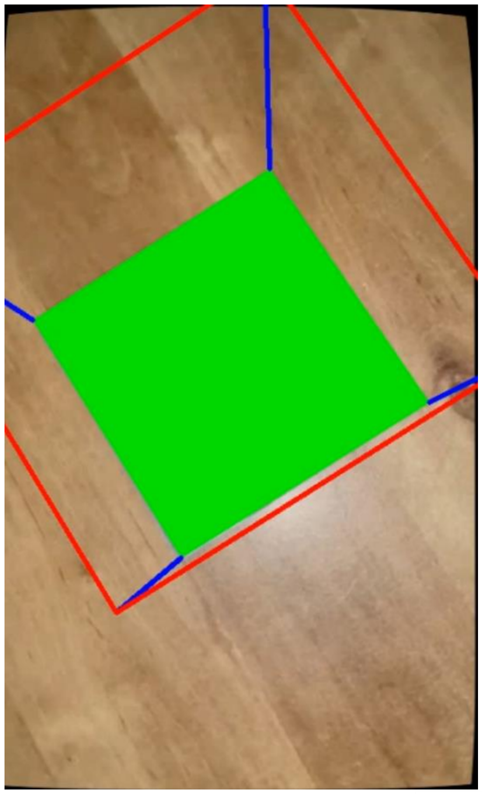
- For the camera calibration we used a video of a few seconds that shows the chess board from a few different angels and positions.
  As we know a video has about 60 frames a second so to make the calculation simpler we captured and tested every 5th picture. For each picture we collected the position of the chess board using cv2.FindChessboardCorners, And improved the capturing using cv2.cornerSubPix. Afterwards we use cv2.calibrateCamera on all the pictured we collected thus far to find the camera matrix, distortion coefficients rotation vector and translation vector. This parameters will help with camera distortion and with step 4.
- Step 2 is as described in part 1, we use the same logic to project the corners of the reference image to the frame coordinates using cv2.perspectiveTransform.
- The resulting variables of step 2 are homographic matrix and a mask. The homographic matrix tells us where a point in the reference picture should be in the video picture, and the mask gives information of which of the key points from the reference picture and from the video picture are perfect and which are not, the mask gives 1 when the key points are the same and 0 otherwise, with this logic we can remove the unwanted key points and return the perfect points to the next step.

- After finding a few good key points from the reference and from the video picture we call the method cv2.solvePnP.
  Cv2.solvePnP will give us two variables rotation vector and translation vector, this variables will determine the position and orientation of the object. Using this information we can render the object in the correct orientation on the image we found before.
- Lastly, we render the mesh to the frame using MeshRenderer class that uses pyrender and trimesh to render a 3d model, we have to provide the size of the video, camera matrix and to draw the object the class need the rotation vector and translation vector we found in the last step.

Here are a few picture from the first part, as we can see the model see the picture of the chess board and detected the chess board inner corners and draw lines between them. We can see that in both picture although the difference in lighting and orientation the detected corners and lines are the same.

Here are a few pictures of the rendered models, there is also a video attached



Where are model succeed and where it fails:

Our model succeed by detecting the relevant key points and solving for the camera orientation and by so render a 3d object at the center of the picture.

As we can see above we tried a few different 3d object to render on the picture, and by so we experimented with a few different file format like .obj and .stl, luckily the library trimesh can read them both but in the case of .stl files the model axis are inverted and by so we needed to find a solution to this problem. In the class mesh render there were a few lines that make the drill object right side up so if we comment this lines the .stl object will be in the right orientation (we can see a picture of this problem below).
Another problem that accrued in the making of the project was the size of the 3d cube, the size is determine by the object points we insert at the very end when we call cv2. projectPoints, and in the notebook we saw in class there were a few multiplication to the object points but we needed to change them because the size of the object was too small. We tried a few different version and numbers until we found the right one.

The right picture is the cube with its original size as we saw in the class.
On the Left we can see that the house is inverted.