CS 8395-01 Augmented Reality
Final Project: Head Pose Tracking and Interaction using Webcam

Mahrukh Tauseef
Professor: Robert Bodenheimer

Due Thursday 28 April 2022

Objective

The idea behind head pose tracking using webcams was inspired by the implementation of a simple pose tracker that can be installed in a car and tracker the pose of the driver (more specifically eye gaze) to see where they are looking on the road. The pose estimation from front camera can be mapped to the image place of a second camera (facing the road) to track where the person is looking. Such technology can be useful to evaluate driving skills. This can be very helpful for individuals with developmental disorders such as Autism Spectrum Disorder (ASD) to be able to track their driving skills and seek help if need be. Even though eye gaze trackers can do a great job at detecting eye gaze, running them in a car on a microcontroller might be computationally expensive. Hence, I wanted to take the first step and explore how well head pose can be detected in real-time.

Theoretical Background

Pose Estimation is a very common problem in the realm of Computer Vision. It has been successfully used for the detection of body pose, object pose, camera pose, or head pose. This has been achieved by the efficiency of linear algebra that allows for getting the location and orientation of a camera and use that information to project a vector in image place that tracks the location of the head pose. Below is a brief description of some of the algorithms that are commonly used in pose detection.

Perspective-n-point

Perspective-n-point (PnP) is a common algorithm used to estimate the orientation of the camera based on the known 3D points of an object as well as the 2D image points of the same object on the image place. In addition, this algorithms needs the camera's intrinsic information i.e., focal length, center of image, and distortion to be able to accurately estimate the pose of the camera.
Figure 1 below shows a depiction of the camera pose detection.
Figure 2 below shows a snapshot of the transformation function that can be used to find the rotation vectors as well as the translation vectors of the camera. The column vector on the left represents the 2D point in an image plane. The first matrix in the right side of the equation is the calibration matrix of the camera with focal length and center points of image plane. The matrix with r and t includes the pose of the camera and the last column shows the known 3D points in world coordinate system corresponding to the known point in image plane Once the position of the camera is known, the pose of the 3D object is real world can be easily projected to the image plane.
Fortunately, these computations can be efficiently and robustly done using a popular API in Computer Vision called *OpenCV*. Even though OpenCV is available multiple platforms, I decided to use python since I am also planning to employ some deep learning techniques for road image segmentation in the future and thus, I wanted to keep everything on one platform.

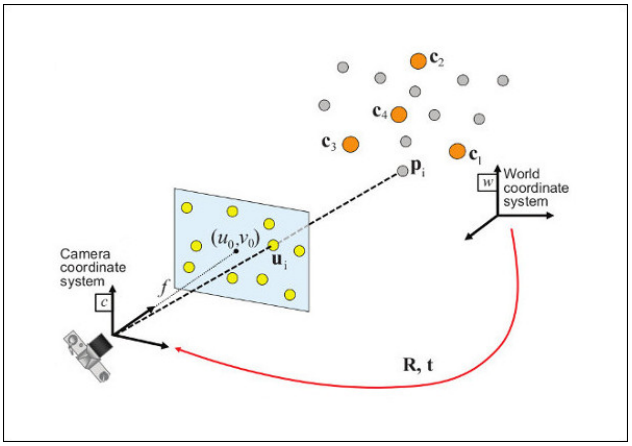

Figure 1: Diagram for the pose estimation problem. From [1]

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Figure 2: Perspective-n-point transformation. From [1]

Implementation

The implementation challenges can be divided into the following groups

Getting 3D point from World Coordinates

Different facial landmarks such as eyes, nose, ears, and chin can serve as useful features for the detection of head pose. However, in order to get these 3D points in world coordinates, a 3D reconstruction of the face needs to be generated to get accurate 3D vectors for these facial landmarks in a reference frame which can be tedious. An alternative to that is to use a generic facial model in an arbitrary reference frame that is able to keep the relative position of the 3D facial landmarks intact. Ahmed [2] and Mallick [3] both used the same 3D vectors of the tip of the nose, center of the chin, left corner of the left eye, right corner of the right eye, left and right corners of the mouth obtained from a generic 3D model of the head.

Detecting Facial Landmarks from Image Plane

The advancement in deep learning has made this problem quite easy since there are a lot of models that can efficiently and accurately detect several facial landmarks from images of the face. They do so by first detecting the face and using a cropped image of the face to run through a landmark detection model. Below are some of the state-of-the-art models for landmark detection:

- Dlib (detects 68 landmarks)
- LBFModel (detects 68 landmarks)
- Mediapipe Face Mesh (detects 468 landmarks)

During the initial stage of the project, I used Dlib abd LBF model to detect the facial landmarks are noticed that LBF model was more accurate. Figure 3 show the detection of all facial landmarks

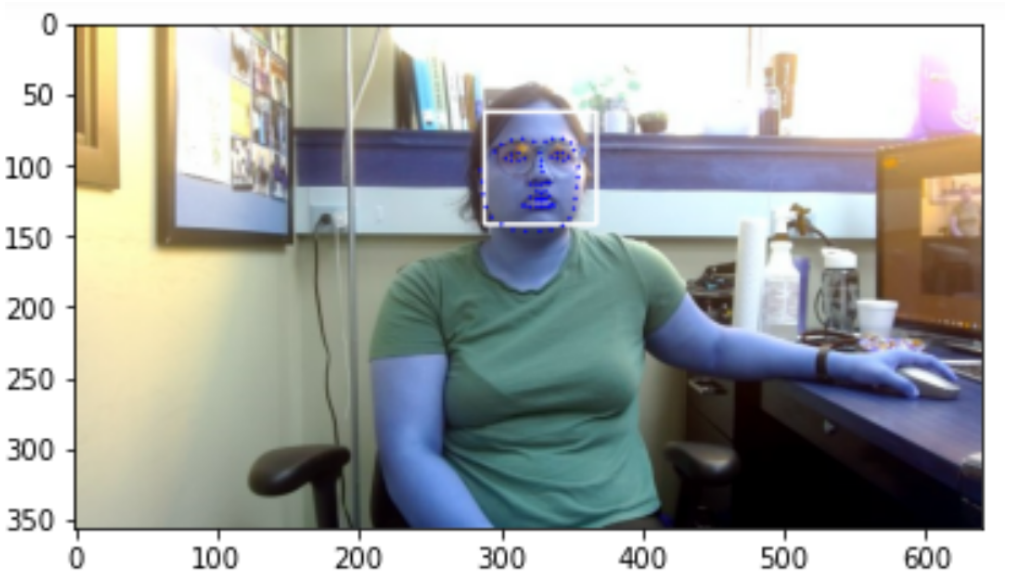

Figure 3: Image of a person with face as well as facial landmark detection using the LBF model. Note: The color distortion was caused by plotting the images using *Matplotlib* as opposed to *OpenCV* due to the issues encountered while creating images on Jupyter Notebook using the latter

Once, the landmarks were obtained, I needed to detect the 2D vectors for the facial landmarks chosen in the section above (i.e., the tip of the nose, center of the chin, left corner of the left eye, right corner of the right eye, left and right corners of the mouth). Figure 4 shows the plot of the facial landmarks in 2D. Using this plot, the six aforementioned landmarks were correctly identified.

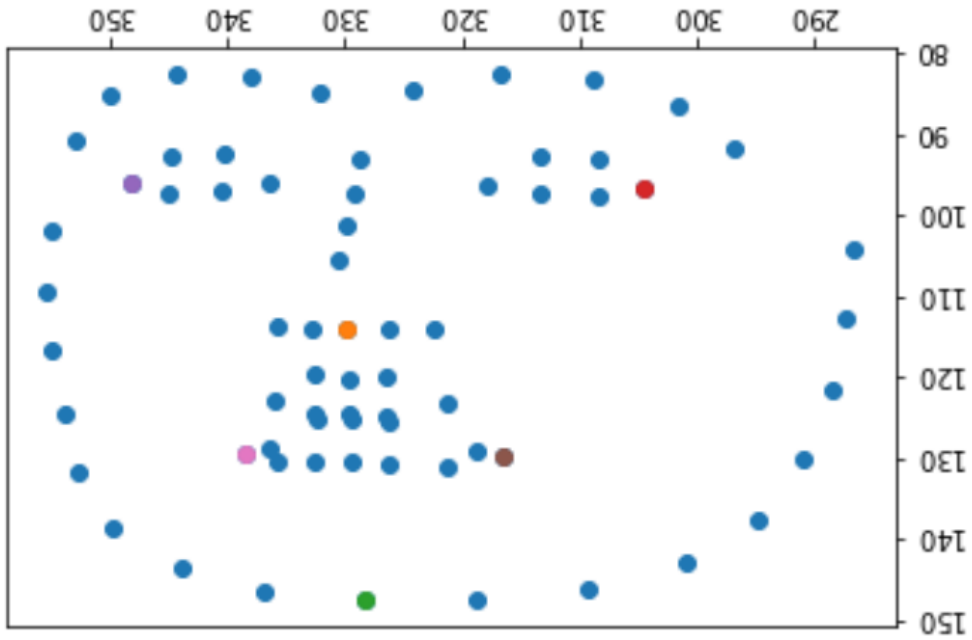

Figure 4: Plot of the facial landmarks. The blue points show all the landmarks whereas all the other colors highlight the ones that were chosen for the pose detection

Camera Calibration

The task was to calibrate the camera. OpenCV allows for the calibration using the Chess Board camera calibration method. The idea is to take several pictures of a chess board printed on a piece of paper and attached to a smooth surface from different rotations and translations. These images can be used to detect important features (such as the corners) of the chessboard to calibrate the camera using *cv.calibrateCamera*. This function outputs the calibration matrix that includes focal length, center of the image, and distortion coefficients. One way to validate the calibration is to view if the algorithm is able to detect the grids of the chessboard accurately for all the orientations of the chessboard. An example of this can be seen in Figure 5 below.

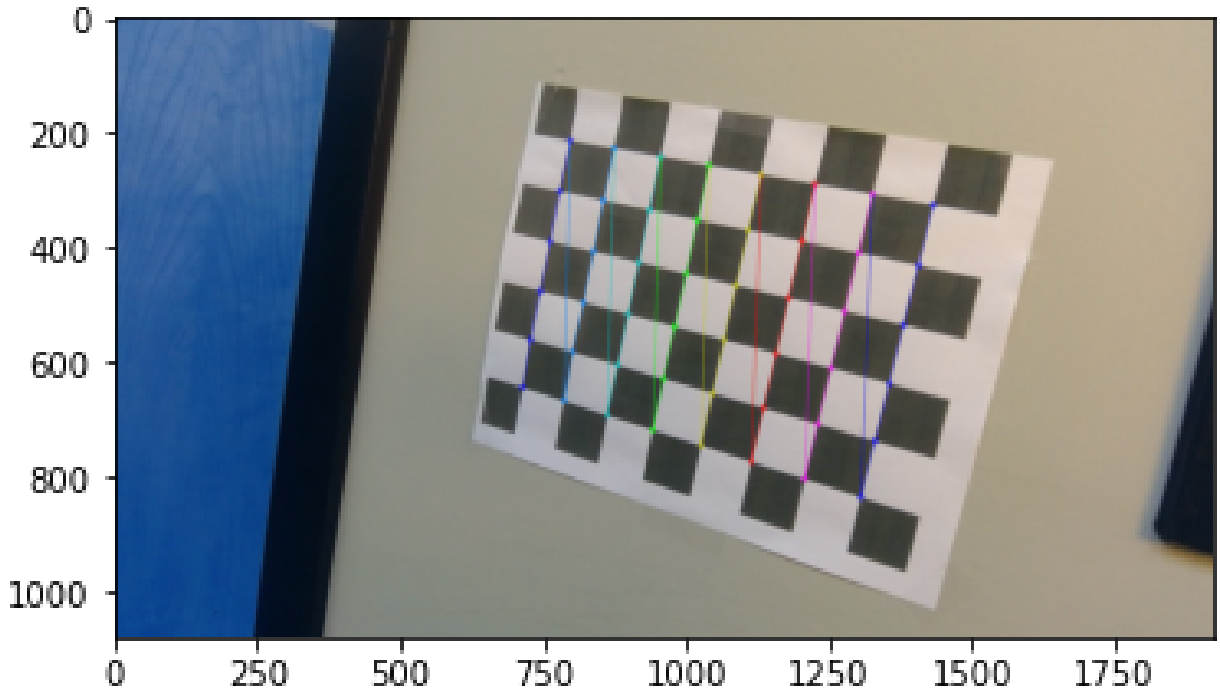

Figure 5: Chessboard grid detection after calibration of the camera

Perspective-n-point and Pointer Projection

As mentioned before, *OpenCV* includes functions that can solve the Perspective-n-point algorithm. However, there are a variety of solvers ranging fron generic *cv2.solverPnP*,*cv2.solverP3P*,*cv2.solverENP* to *cv2.solverPnPRansac*. All of them utilize slightly different techniques to solve the algorithm. However, PnPRansac has been known in the community for its robustness. Hence, I decided to utilize it.
Once the rotation and translation vectors for camera pose were retrieved, they were used to project the points that inform the pose in the image plane. This was done by using *cv2.projectPoints* function. These points were then used to generate a virtual line vector from the nose to the screen to show where the nose is pointing. This can be seen in figure 6
Even though, the projection of the point seems quite accurate, there is a slight offset that might be the result of error introduced by generic 3D facial model. This can be corrected by adding a constant.

Real-time Implementation

Even though, the methods mentioned above worked well with static images, the error of the projection significantly increased when implemented on a video stream. In addition, the projection also showed instability

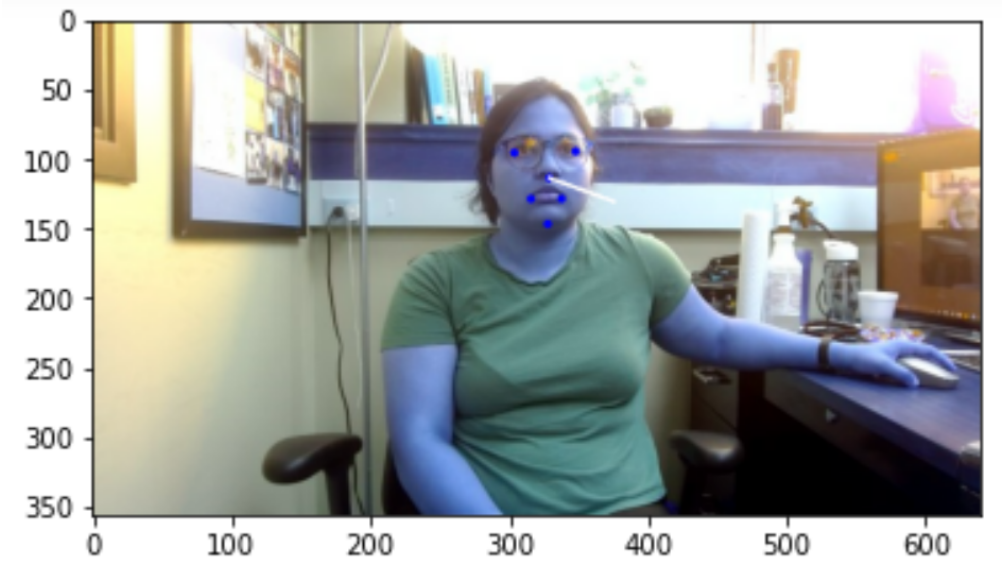

Figure 6: Projecting a point that depicts the head pose and facial landmarks used to estimate it

when the frames were changing.
I went through different tutorials as well as guides for head pose estimation to find a fast robust head pose detection. I adopted the technique to generate the point based on the angle offset detected due to the pose of the head from tutorial [4]. This angle can be obtained by using converting the rotation vectors obtained from PnP into a rotation matrix and then decomposing it using *cv2.RQDecomp3x3* into angles.
In addition, the *LBF model* was replaced by a more accurate option provided by *MediaPipe* that allows for the 2D landmark detection in image place as well as 3D points for these landmarks based on a generic model. They showed improvement in accuracy.

Interaction

In order to make the set up more interactive and to show one of the applications of this pointer, I decided to add three buttons to the video screen. Each button plays a certain sound clip every time the pointer touches it. Figure 7 shows a snapshot of what the view looked like with the pointer and the buttons. A video is added to the GitHub repository that showcases a demo.

Limitations

- Error was introduced because the 3D facial points were obtained from a generic model as opposed to the face of the user.
- Camera calibration itself is an estimation since the intrinsic features of the camera are not available to public
- A lot of trial and error was needed to tweak the angle in order to the mapping accurate

Future Work

Originally, this project was proposed to include a second part i.e., mapping of this pointer in the image plane of a second camera that capturing the view of the person. However, upon exploration, I discovered that such mapping needs advanced understanding of computer vision techniques that was beyond the scope

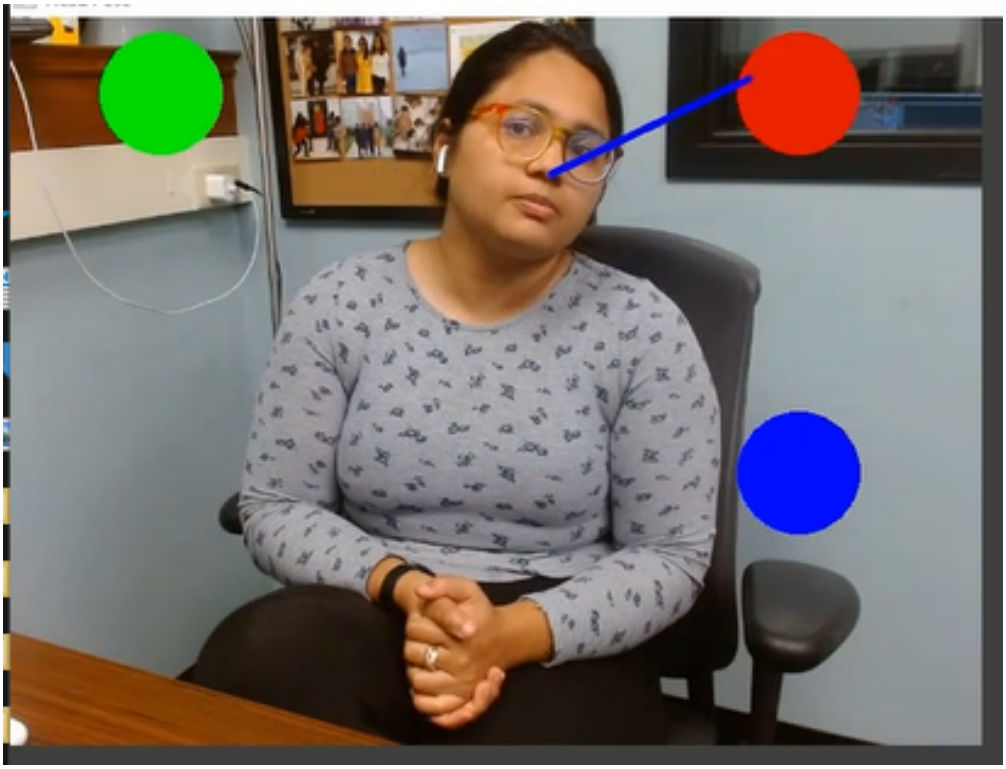

Figure 7: Video with a participant interacting with the buttons using the nose pointer

of this class. However, I hope to continue this project over the summer to see how the pointer projects into the image plane of a second camera.

References

[1] "Perspective-n-point (pnp) pose computation." [Online]. Available: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

[2] S. Ahmed, "Head pose estimation using opencv solving pnp." [Online]. Available: https://github.com/by-sabbir/HeadPoseEstimation

[3] S. Mallick, "Head pose estimation using opencv and dlib: Learnopencv ," May 2021. [Online]. Available: https://learnopencv.com/head-pose-estimation-using-opencv-and-dlib/

[4] N. Nielsen, "Head pose estimation with mediapipe and opencv in python - over 100 fps!!!" [Online]. Available: https://morioh.com/p/b683d0d9d3ef