MediaPipe Iris Tracking Methodology

Written by: Arnab Mazumder

Requirement

The basic requirement this document is addressing, is to fix and enhance iris tracking from a single user's eyes (from a webcam view) and replicating that onto our in-game avatar's eyes.

The eyes would need to be responsive enough to allow movements in any direction, such as diagonally up and rightward, eye rolling and looking down.

Previous Approach

- Using MediaPipe for Unity, tracking of iris was being performed by using 3D FaceLandmark points received from 'OnFaceLandmarkDetectionOutput' event function inside the '**FaceLandmarkerRunner**' class.

- Eye angles were determined initially by reading the angle subtended between the iris center point, and the two extreme (inner and outer) points, for each eye, on their instantaneous local X-Z planes. These calculations were attempted in both world space, as well as properly converted local space of the eyes.

- This 3D landmarks-based rotation approach resulted in unpredictable Z-value of iris center landmark in relation to the eye's inner or outer points, complicating our task of setting iris rotations to avatar's eyes.

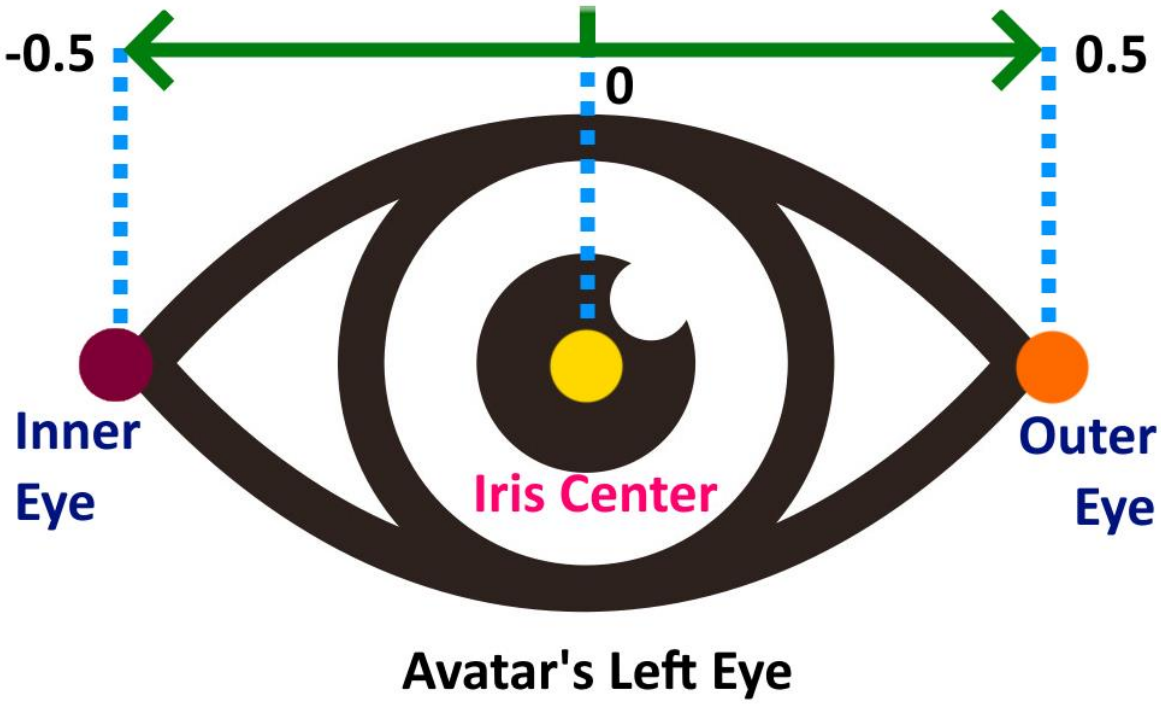
3D Normalized Range Approach

- We go with a new approach of measuring the normalized distance of the iris center between the two extreme (Inner and outer) points, for each eye. The normalized values are measured only on a single axis (The imaginary axis formed between inner and outer eye points).

- The iris center point is projected onto this imaginary line and its relative distance between the points is measured.

- Finally, this normalized value (Ranging between -0.5 to 0.5), is multiplied by a factor value, and this multiplied value is clamped between pre-defined min-max values (typically [-44, 44]), to get the final eye rotation in degrees.

- This means when this normalized value returned is -0.5, the iris is looking fully right within the range, and vice-versa for full left iris position. A 0 value indicates the iris is looking dead in the center. Both eyes use the same approach, although with inversed normalized values and ranges. This has been visualized as below for left eye:



- One problem with this approach was that head rotations caused the 3D landmark points for eye inner and outer, and the iris point, to unpredictably change their Z-positions relative to one another, resulting in deviated iris center point projections on the imaginary inner-outer line, consequently deviating resulting angles by a lot as the user's head rotated on Y-axis.

- To combat this, the avatar's head Y-rotation was tracked using the nose bridge's topmost and bottom-most two points, to determine whether the user was turning their head leftward or rightward, and by how much.

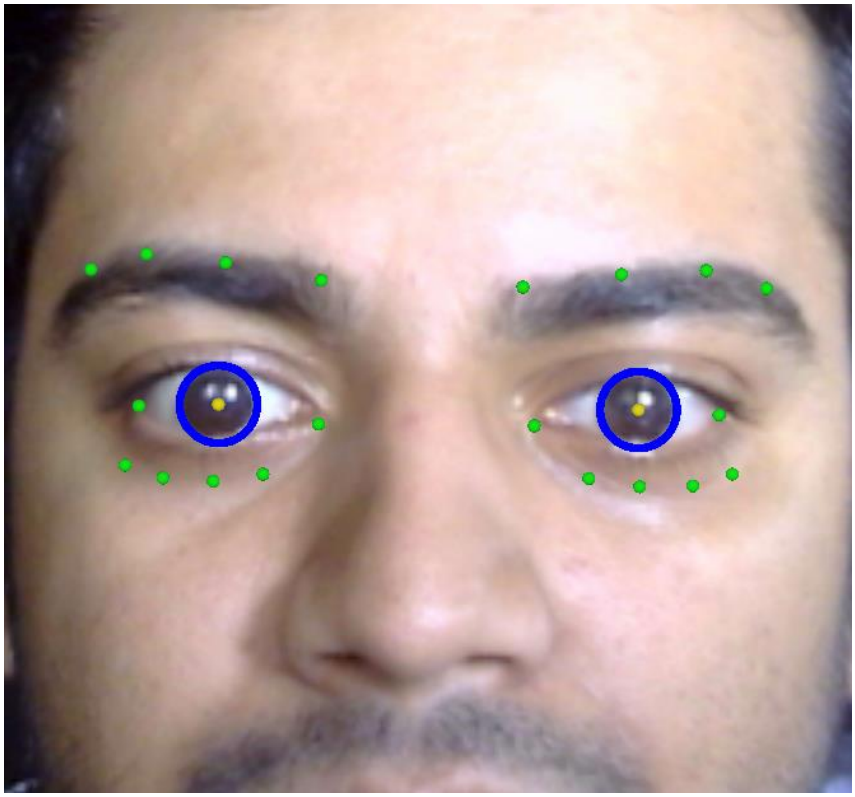
- A 'tracked' eye and 'follower' eye method was used, where, depending on the head Y-rotation, one eye (Which would be the most visible depending on how the head has been rotated) is set to be the tracked eye, from which all values are

calculated, while the other eye (which would not be very visible due to the head turn) was set to be the ‘follower’ eye. The Y-axis rotation of the tracked eye was copied onto the follower eye every frame, to avoid random cross-eyed or wide-eyed look due to Z-deviations. This created further issues where a sudden ‘jump’ was visible when tracked eye was just changed with very slight rotations of the head from Y-axis center 0 rotation.

- By knowing the head rotation on Y-axis, exponential offsets were temporarily used to observe and reverse the deviating effects of head turn based Z-deviations of all points using trial and error.

- Notably, it was also found that converting the same Landmark points to 2D space, by simply using the X and Y coordinates and discarding Z values, resulted in the same unpredictable behavior at facial rotation extremes.

2D Annotations Normalized Approach



- The final working approach was to completely discard the 3D Landmark points from ‘OnFaceLandmarkDetectionOutput’ event function and experiment directly with MediaPipe’s own Annotation points.

- MediaPipe annotates eye-points relative to a 2D Raw-image containing the user’s live webcam view, to demonstrate point positions relative to a user’s facial expression and eye movement.

- Although these annotations are mainly to demonstrate tracking capabilities, these annotations work beautifully to provide much more accurate results for eye movements.

- Annotation points completely ignore any 2D movement and directly lie on this webcam Raw-image’s flat plane, which means all annotation points being synced with user’s face are always going to provide reliable 2D space point positions, eliminating concerns with any Z-space deviations or any unnecessary projections or offsets needed for calculations.

- This method completely de-couples any user head rotation on any axis, from normalized values of iris between the inner and outer points for any eye. This means, no matter where the user looks or turns, MediaPipe will always annotate all points onto a single plane, meaning that normalized values returned are almost always going to be correct and calculating a rotation from this normalized value is more reliable than using 3D Landmark points.

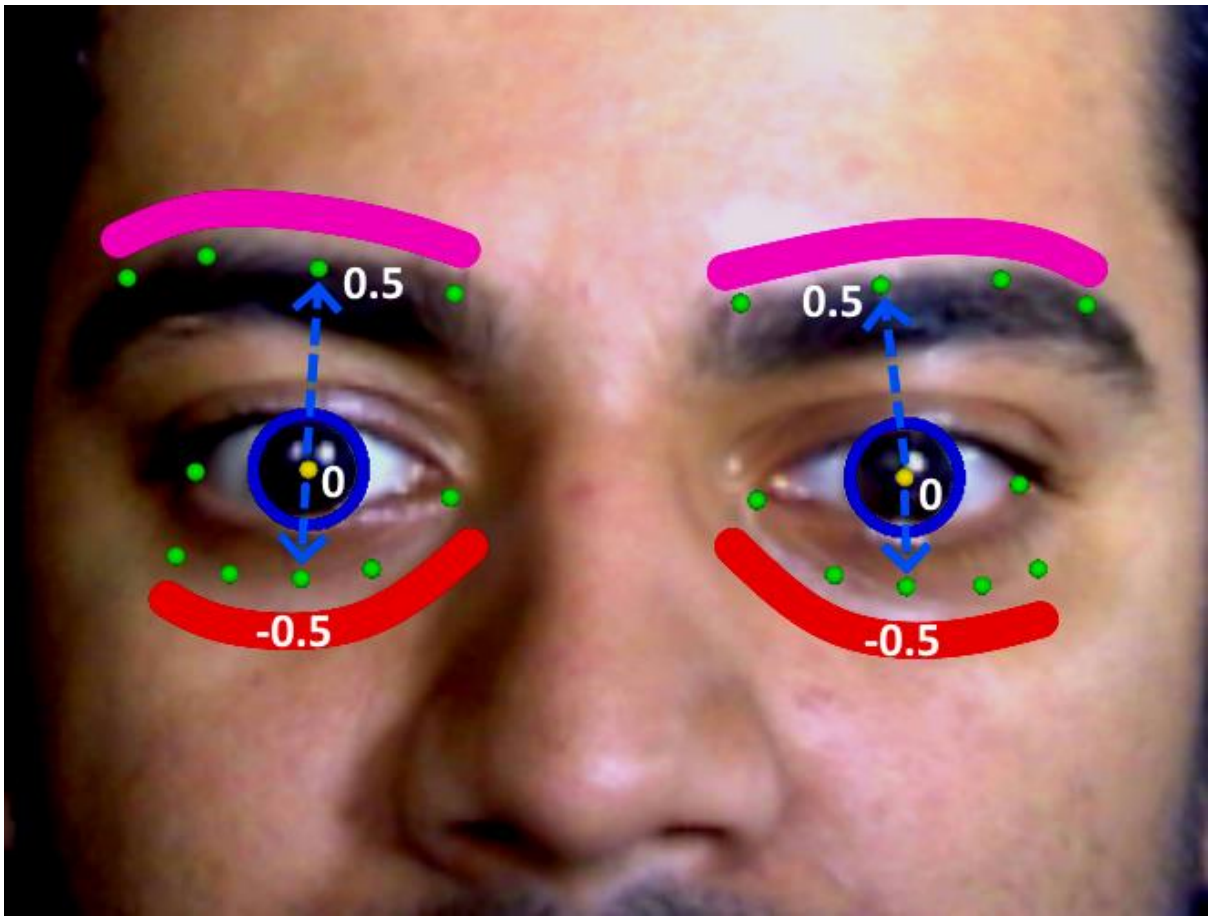
- From here, we continue with our previous normalized value approach to get rotation values.

- One major benefit of this approach is it eliminates the need to provide Z-deviation based exponential offsets, any dead-zones needed. It also eliminates the need to track head rotations and nose bridge points.

- Another benefit is that this approach allows us to independently use rotations values from both eyes without any tracked and follower eye concept from our previous complex approach. This permits unique movements such as deliberate cross-eyed look by user, due to eliminating the need copying rotations from one eye to the other. This feature is missing in some competitor eye tracking applications.

Enabling Vertical movement with similar Approach

- Using the same approach also easily allows us to enable vertical iris movement, albeit with a slightly different mechanism:



- From the above visual, it is evident that two different imaginary ‘curved’ lines are formed for each eye, above and below it. For any single eye, the pink curved line is formed by four tracked upper eyebrow annotation points by MediaPipe, while the bottom red curved line is formed by 5 cheekbone-to-eye annotation points.

- This multi-point approach forming a ‘curve’ is important to account for the tapering end shape of the eyes, which are not uniform and would thus result in deviated vertical rotation results if only one top (center) and one bottom (center) points were used, as the iris move horizontally across the eye’s taper-ended shape.

- The iris vertical rotation is determined using the same method as horizontal rotation, but by using a top and bottom point at any instant. We use an efficient squared magnitude based nearest point method to get the top point, which is closest to the iris center point, from the set of points on the top line. The same is done to retrieve the nearest bottom point too. Hence, in every frame, the iris uses these closest top and bottom points to calculate the normalized values.

- Importantly, these points were carefully picked out by index, using trial and error, as these points proved to not move vertically by much, when user's iris moves upwards or downwards. The eyelid points (which are closer to the iris) move by a large vertical differential when user iris looks up or down, resulting in very weak normalized values. Hence, we go with the upper eyebrow and lower cheek points as shown.

- This means we can determine the normalized value of iris center between these two curved top and bottom 'lines', to get vertical eye rotations.

Conclusion

We successfully use MediaPipe 2D annotation points to get higher fidelity iris tracking results from even very low resolution webcams, allowing a multiplatform deployment without relying on platform specific fidelity.