EyeTracking Project Report – AUS Lab 2

Al Sarsar Al Kodmani Mousallam

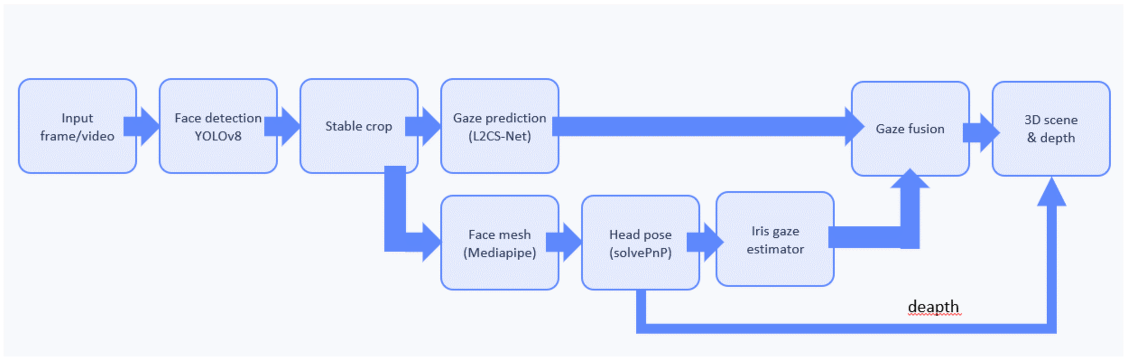
Introduction

Eye tracking is the process of estimating where a person is looking. Accurate gaze estimation enables a rich set of applications ranging from **driver monitoring** and advanced driver assistance systems (ADAS) to **virtual/augmented reality**, **assistive technologies**, and industrial automation. In the automotive domain, understanding where a driver's attention is directed allows the vehicle to detect fatigue, distraction or unsafe behaviours and to issue timely warnings. In AR/VR headsets or remote collaboration systems, eye tracking enables foveated rendering and more natural human-computer interaction. Finally, gaze information can improve process safety in manufacturing by ensuring that operators pay attention to critical control panels.

The EyeTracking project studied in this report aims to deliver a **real-time gaze-tracking system** that works with a single RGB camera. The system accepts a live video stream or a pre-recorded video and estimates the 3-D gaze direction of the subject's eyes. It uses state-of-the-art deep learning for face and gaze estimation, geometric computer vision methods for head pose and depth, and probabilistic filters to fuse information across frames. The following sections describe the methodology in detail and present sample results obtained from the implementation.

Methodology

The pipeline of the EyeTracking project can be broken down into several stages, each responsible for a specific task. **Figure 1** outlines the overall pipeline used by the system. The subsequent subsections explain the mathematics and the algorithms behind each stage.



Overview of the EyeTracking pipeline showing camera calibration, face detection, gaze estimation, landmarks and pose estimation, iris gaze, fusion, and 3-D scene reconstruction

1. Data input and camera calibration

The system supports two modes of operation:

1. **Live camera mode** uses a webcam or camera attached to the device. Frames are captured sequentially, processed in real time and displayed with overlay graphics.
2. **Video input mode** reads frames from a video file, processes them and writes results back to a video or image sequence. This mode is useful for offline analysis.

Camera calibration is required to recover metric and geometrically consistent information from image measurements. In this project, calibration is **not performed online using a calibration pattern**, but instead relies on the **factory and specification-based parameters of the Lenovo 500 FHD webcam**, which is the camera used throughout the application.

The camera is modelled using a **pinhole camera model** with radial and tangential distortion. The intrinsic matrix, containing the focal lengths and principal point, is defined according to the known resolution and optical characteristics of the Lenovo 500 FHD webcam. Likewise, the distortion coefficients (k1,k2,k3) for radial distortion and (p1,p2) for tangential distortion are taken from the camera's documented specifications or reasonable approximations consistent with consumer-grade webcams.

By using known camera parameters rather than performing a full calibration procedure, the system simplifies deployment and ensures consistent behaviour in both live and video-based modes. Although this approach may introduce small metric inaccuracies compared to a full chessboard-based calibration, it is sufficient for the project's objectives, particularly head pose estimation, gaze direction computation, and relative depth estimation using solvePnP.

Lenovo 500 FHD Webcam

2. Face detection and cropping

The first stage of processing each frame is detecting the face. The project leverages **YOLOv8**, a one-stage object detector that employs an advanced backbone and a split head design. YOLOv8 returns a set of bounding boxes and confidences for detected faces. The largest bounding box is selected as the face region of interest (ROI). To improve stability across frames, bounding boxes are fused using a **union bounding box**: the union of detections across a short temporal window is computed to provide a stable crop. The resulting ROI is resized to a fixed resolution (e.g., 224×224) and passed to the gaze estimator.

The YOLO model is anchor-free and uses convolutional layers to extract features at multiple scales. In the context of face detection, YOLO’s speed makes it suitable for real-time applications, while its accuracy ensures that the entire face is captured for subsequent processing. Alternatives could include two-stage detectors such as RetinaFace or MTCNN, which may offer higher precision but at the cost of computational complexity.

3. Gaze estimation with L2CS-Net

The **GazeEstimator** module employs **L2CS-Net**, a convolutional neural network designed for appearance-based gaze estimation. It takes the face crop as input and outputs the **pitch** and **yaw** angles of the gaze vector relative to the head. L2CS-Net uses a **ResNet-50** backbone with residual connections to extract features, followed by two separate fully connected heads – one for pitch and one for yaw. Each head formulates angle estimation as a classification–regression problem:

- **Classification:** the continuous angle range is discretized into (n) bins. The network predicts a probability distribution over bins via softmax, and the angle is recovered.
- **Regression:** a small offset is regressed within each bin to refine the angle estimate.

This hybrid loss encourages the network to learn coarse bin assignments while maintaining the ability to predict fine-grained offsets. Alternative approaches include fully regression-based models or purely classification-based gaze networks, but the combined method tends to achieve higher accuracy.

4. Face landmarks using MediaPipe Face Mesh

After obtaining a coarse gaze estimate, the system localises facial landmarks using **MediaPipe Face Mesh**. This model predicts 468 three-dimensional landmarks on the face in real time. Key landmark groups include the eyes, irises, nose, mouth and jawline. The face mesh provides the 2-D pixel coordinates and estimated depth of each landmark, enabling extraction of points such as the corners of the eyes and the tip of the nose.

Face Mesh comprises two models: a face detector that crops the face and a landmark regressor that outputs dense 3-D coordinates. The model uses a lightweight neural network and a **Procrustes analysis** module to map the landmarks into a metric 3-D coordinate system. Alternatives to MediaPipe include 3-D Morphable Models or techniques such as Dlib's facial landmark detector, but MediaPipe offers high speed and accuracy.

5. Head pose estimation (SolvePnP)

To compute where the head is oriented relative to the camera, the project uses the **Perspective-n-Point (PnP)** problem. Given a set of known 3-D points on the face (e.g., nose tip, chin, outer eye corners, inner eye corners, mouth corners) and their corresponding 2-D image coordinates, **OpenCV's `solvePnP`** returns the rotation vector and translation vector that align the 3-D model with the image.

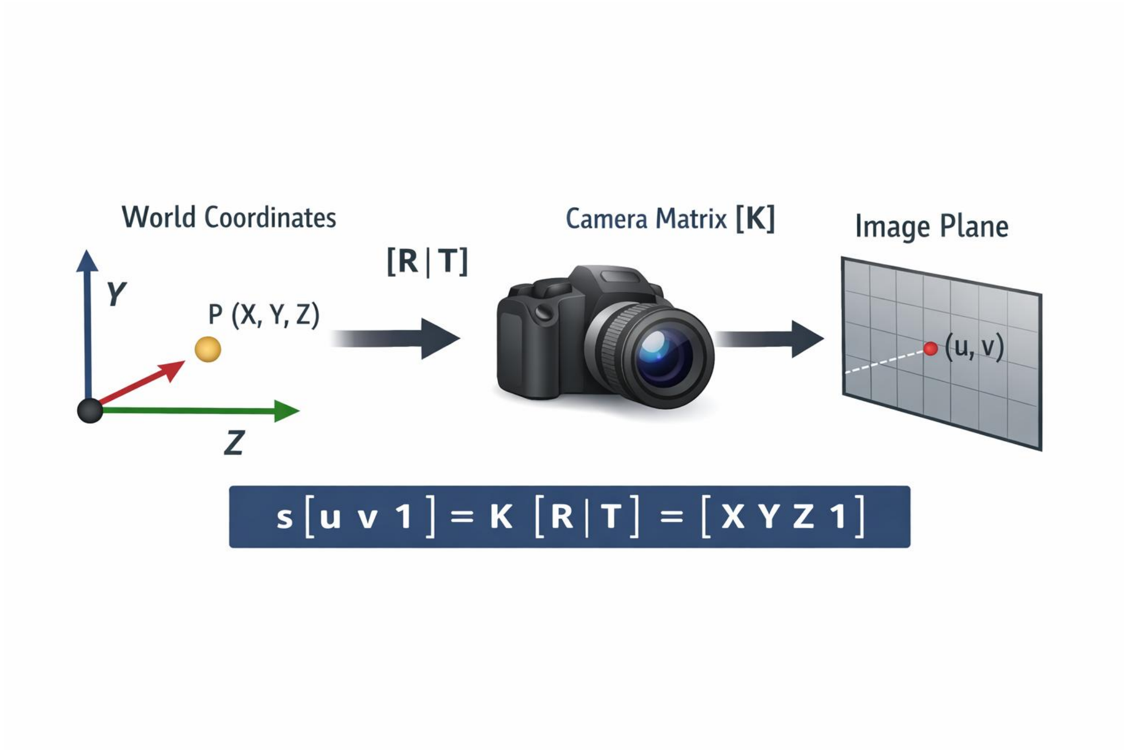
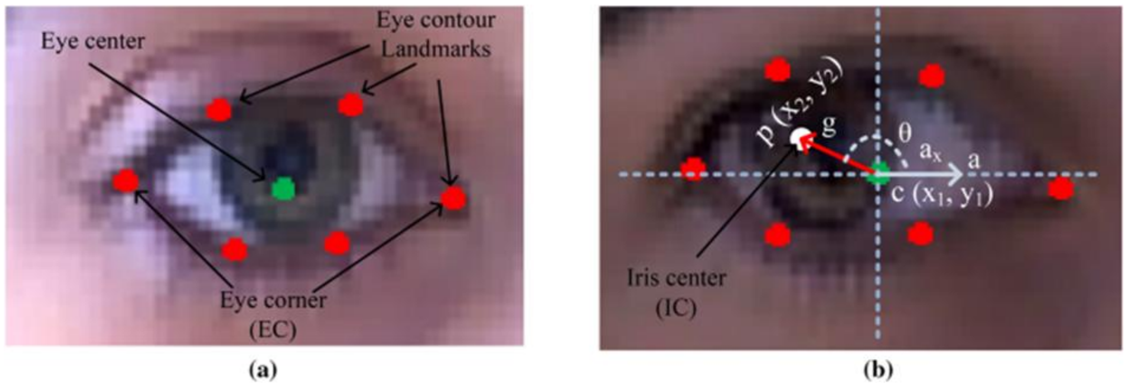


Illustration of the Perspective-n-Point problem: 3-D facial points are projected onto the image plane through camera intrinsics and extrinsics

6. Iris gaze estimation and vergence

While L2CS-Net provides a global gaze direction from the face crop, precise gaze requires analyzing each eye separately. The **IrisGazeEstimator** focuses on the left and right eyes. It proceeds as follows:

1. **De-rotation:** the input eye images are warped to compensate for head roll using the previously estimated rotation matrix. This ensures that the eye region is upright.
2. **Region of interest:** bounding boxes around each eye are extracted based on facial landmarks.
3. **Iris segmentation:** the iris is segmented using grayscale intensity thresholding and morphological operations. Erosion removes noise and dilation fills gaps. The centre of the iris is computed from the segmented region.
4. **Vector computation:** a 2-D vector from the centre of the eye to the centre of the iris is calculated. From this vector, the gaze amplitude and direction (angle) are derived.
5. **Vergence:** the gaze vectors of both eyes are combined to compute vergence – the inward or outward rotation of the eyes used for depth perception.



Schematic of iris segmentation and gaze vector calculation

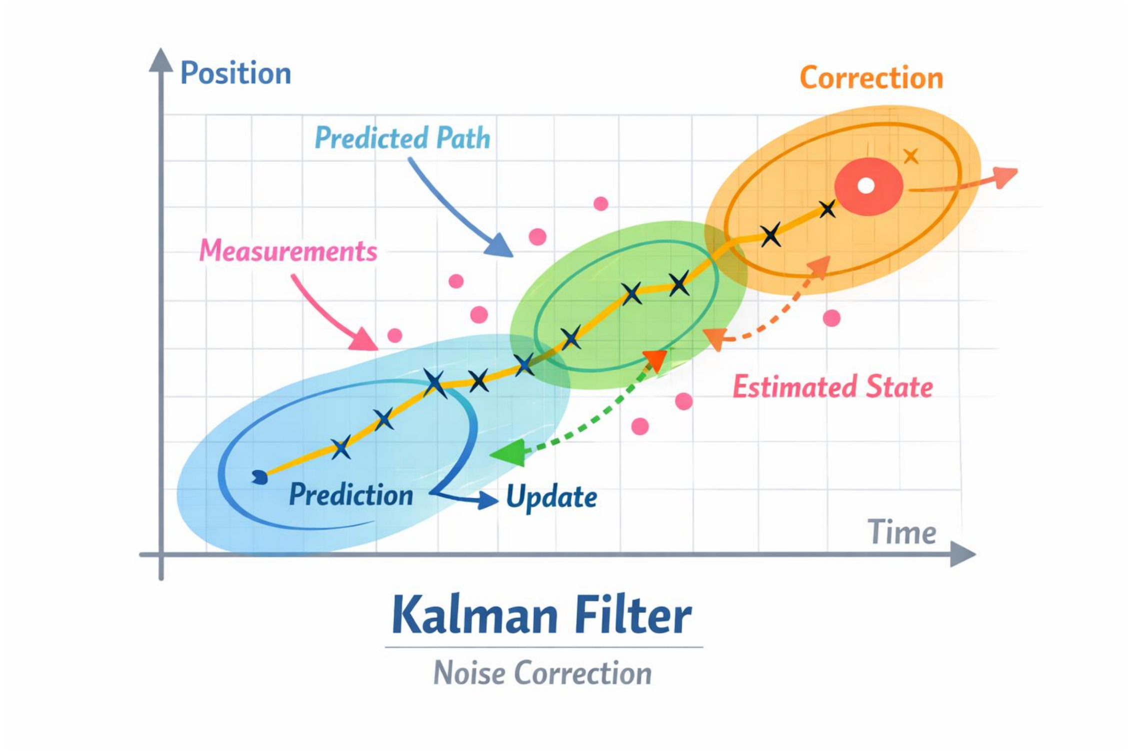
This step relies on simple image-processing operations rather than deep learning, which makes it computationally cheap. Alternatives include deep-learning–based iris segmentation networks; however, the threshold-based method suffices when lighting conditions are controlled.

7. Gaze fusion and temporal smoothing

The system produces two gaze estimates: one from L2CS-Net and one from the iris module. To combine these estimates and reduce noise, a **Kalman filter** is used. A Kalman filter is a recursive Bayesian estimator that maintains a state vector consisting of position and velocity of the gaze point in image space. It alternates between **prediction** (propagating the current state forward using a constant-velocity model) and **update** (correcting the prediction using a new measurement).

The Kalman filter also smooths the **depth estimate** obtained from the translation vector. This step is critical because depth estimates vary across frames due to noise in the landmark positions and solving PnP. The filter reduces jitter and yields stable gaze targets.

A fused gaze vector yields more accurate pointing directions than either L2CS-Net or iris-based estimates alone. Alternative fusion techniques include particle filters or exponential smoothing, but the Kalman filter strikes a good balance between simplicity and performance.



Conceptual diagram of the Kalman filter showing prediction and update phases

8. 3D scene reconstruction

Finally, the project renders a **3-D scene** showing the face, head pose and gaze direction. The **Scene3D** module transforms each facial landmark into a common reference frame and projects them back into the image. The 3-D rotation matrices derived from Euler angles rotate the points, and the camera intrinsics project them. A coloured gaze arrow and axes are drawn over the scene to visualise where the subject is looking. The scene can also depict an object plane (e.g., a virtual screen) so that gaze intersection points can be estimated.

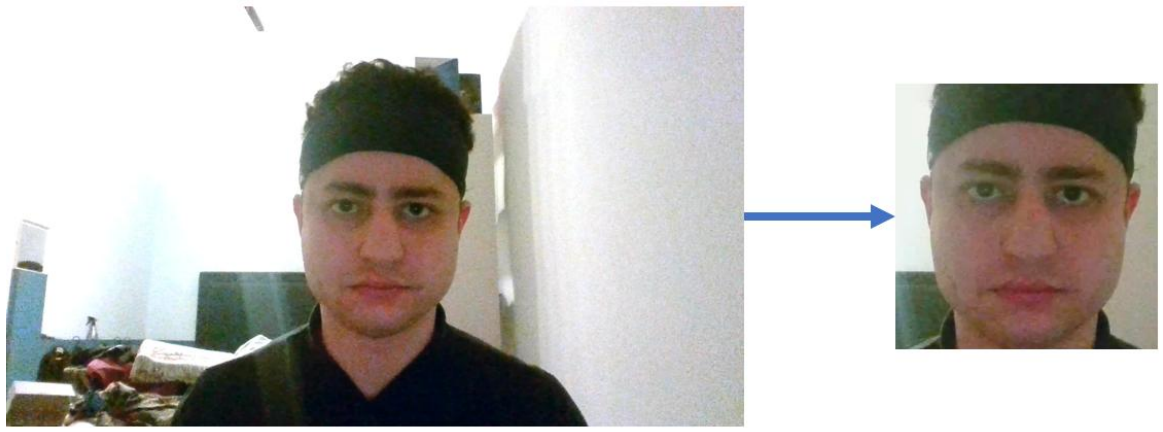
The 3D rendering illustrates how the gaze vector interacts with the environment and helps debug the system. Alternatives include overlaying gaze on the 2D image alone; however, the 3D scene provides depth context.

Results

The following subsections illustrate the output produced at each stage of the EyeTracking pipeline. Screenshots are taken and showcase how the algorithms perform on sample frames.

Face detection and cropping

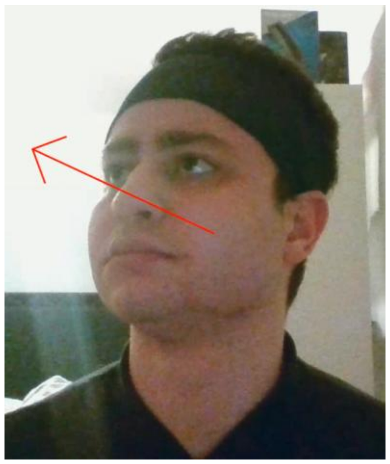
The YOLOv8 detector reliably identifies the subject's face and defines a stable cropping region. The union bounding box over successive frames ensures temporal stability and prevents jitter. The cropped face is resized and normalised before passing it to subsequent modules.



Face detection and cropping

Gaze estimation (L2CS-Net)

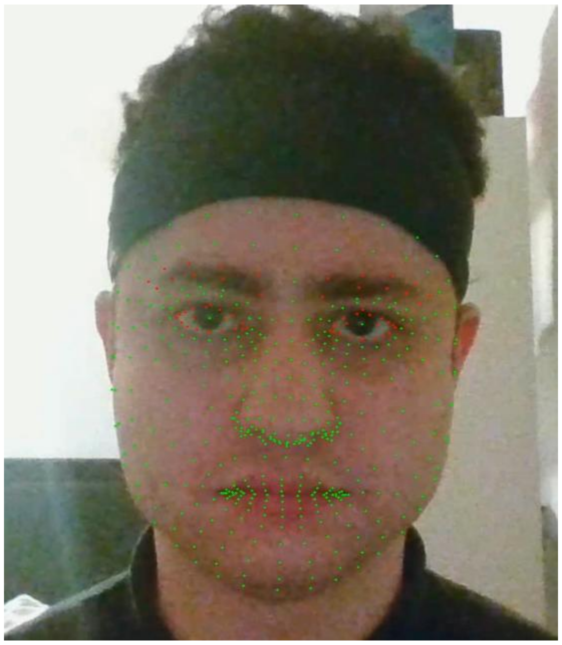
L2CS-Net predicts pitch and yaw angles that roughly match the subject's gaze direction. The network handles variations in head pose and lighting conditions and produces an initial gaze vector relative to the head.



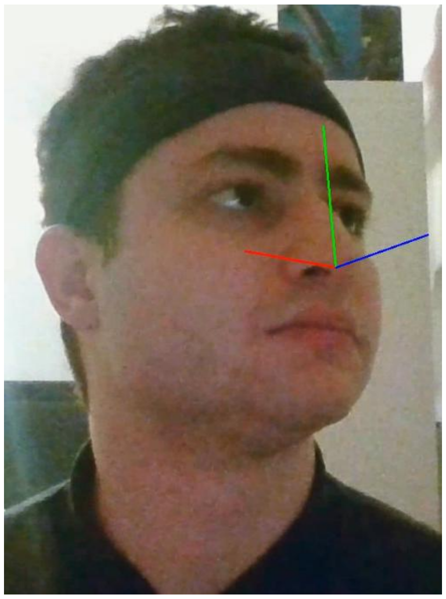
Global gaze estimation result

Landmarks and head pose

Dense face landmarks from MediaPipe Face Mesh enable accurate geometric processing. Landmarks such as the outer eye corners, mouth corners and nose tip are used with `solvePnP` to estimate the rotation and translation of the head. The visualisation shows landmarks overlaid on the face and axes representing the head pose.



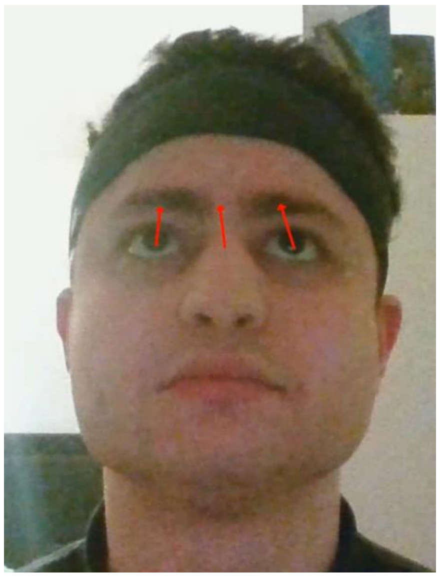
Face landmarks overlay



Head pose axes visualization

Iris gaze and vergence

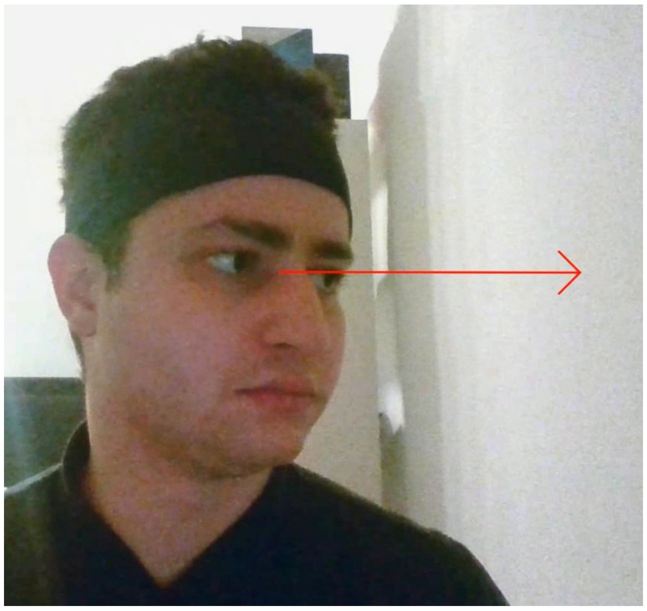
After de-rotating the eyes, thresholding and morphological operations isolate the irises. Vectors drawn from the centre of each eye to the iris centre indicate gaze direction and vergence.



Iris gaze estimation result

Fusion and smoothing

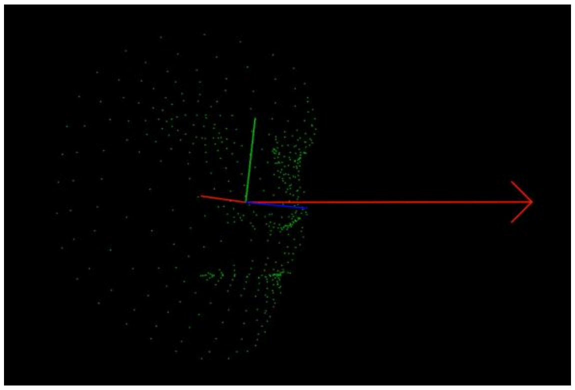
The Kalman filter fuses the gaze estimates from L2CS-Net and the iris module and smooths them temporally. The final gaze vector is stable and accurate even when individual estimates fluctuate.



Final fused gaze vector

3-D scene rendering

The Scene3D module reconstructs a 3-D scene by transforming and projecting facial landmarks back into image space. Axes indicate head orientation and a coloured arrow visualises the gaze vector intersecting the scene.



3-D scene representation with gaze and face landmarks

Conclusion

The EyeTracking project demonstrates a complete gaze-tracking pipeline built upon deep learning, geometric computer vision and probabilistic filtering. By calibrating the camera, detecting faces, estimating gaze angles with L2CS-Net, finding facial landmarks, solving for head pose using PnP, computing iris-based gaze and fusing them with a Kalman filter, the system achieves real-time 3-D gaze estimation. The 3-D scene reconstruction and depth smoothing further enhance usability.

From a methodological perspective, each step in the pipeline was chosen for a reason: YOLOv8 balances speed and accuracy for face detection; L2CS-Net's classification–regression framework yields precise angle estimates; MediaPipe Face Mesh provides dense landmarks for robust head pose; `solvePnP` offers reliable extrinsic estimation; simple thresholding suffices for iris segmentation; and the Kalman filter delivers smooth fusion. The alternatives, such as two-stage detectors, purely regression-based gaze networks or particle filters, each have merits but also trade-offs in complexity and latency.

Future improvements could include using a stereo camera for depth estimation, adopting transformer-based gaze networks, or integrating eye–environment context to infer fixation points. Nevertheless, the current implementation sets a strong baseline for real-time eye tracking using only a single camera and open-source tools.