Spatial Constraint-Based Hand Detection in First-Person View

Fiona Spencer
Computer Science
Toronto Metropolitan University
fspencer@torontomu.ca

***Abstract*—This project presents a lightweight computer vision pipeline for egocentric (point-of-view) hand-gesture detection using a cost-effective, compact camera module suitable for wearable devices. The system is designed to support real-time interaction by ensuring reliable visual input through image processing and enhancement techniques, such as blur and noise detection. Camera calibration is performed using checkerboard-based corner detection to achieve accurate geometric projection and spatial consistency. Edge detection and corner detection are implemented through methods such as Canny edge detection, Hough line transformation, and Harris corner detection. By integrating region-of-interest constraints with first-person hand detection, the proposed approach demonstrates how low-cost hardware can produce robust and practical real-time hand-gesture tracking for immersive and wearable computing applications.**

Git Repo: https://github.com/fspencer-edu/CV-Final-Project

Keywords*—*Computer vision, hand gesture recognition, camera calibration, wearable device (AR/VR), region-of-interest, monocular vision.

I. Introduction

The rapid development of augmented and virtual reality (AR/VR) headsets has expanded their use well beyond entertainment into everyday applications such as productivity, communication, and interactive systems, as well as into fields including education, medical training, manufacturing, engineering, and commercial industries [1]. AR/VR technologies are increasingly used for immersive learning environments, surgical simulation and rehabilitation, remote collaboration, and experimental visualization, and are now widely viewed as a foundational platform for future digital interactions [2]. As modern head-mounted displays such as Apple Vision Pro [2], Meta Quest, and Meta Orion AR Glasses [3] continue to evolve, intuitive and reliable interaction methods such as hand-based gestures have become a critical component for effective user experience in first-person environments.

Most commercial AR and VR headsets rely on the addition of depth-sensing hardware such as LiDAR to enable accurate three-dimensional reconstruction of the environment for precise gesture recognition and pose estimation [5]. LiDAR sensors generate dense 3D point clouds by measuring the time-of-flight of emitted laser pulses, providing depth information for scene geometry and spatial interaction. While these sensors offer high-precision depth measurements, they significantly increase system cost, power consumption, and hardware complexity, limiting accessibility and miniaturization for lightweight or experimental platforms. In contrast, this application explores a vision-only approach using a compact and low-cost ESP32 camera module (Fig. 1), demonstrating that reliable egocentric hand-gesture detection can be achieved without dedicated depth sensors.

Fig. 1. ESP Camera Model Body, FTDI, and ESP32 Camera Module

Feature tracking is a fundamental component of computer vision systems used in AR and VR, enabling the identification and temporal correspondence of visual elements across image frames. Common feature tracking approaches include point-based methods, such as corner and key point tracking using detectors like Harris [6], FAST [7], and Scale-Invariant Feature Transform (SIFT) [8], which provide robustness to scale and rotation changes; edge- and line-based methods, which exploit geometric structures such as contours and boundaries for stability in structured environments; and model- or landmark-based methods, which track predefined patterns or learned features, often used in hand, face, or object tracking.

Hand-gesture interaction has become an increasingly prominent input in wearable and spatial computing systems, offering a natural and intuitive alternative to traditional controllers, keyboards, and touch interfaces [9]. From a computer vision perspective, gesture-based interaction relies on techniques such as image preprocessing, feature extraction, hand segmentation, key point and landmark detection, and temporal tracking to robustly interpret hand motion and pose in real time. In egocentric vision systems, hand gestures are effective because they align closely with the user's field of view, allowing direct and immersive interaction [10].

The framework explored presents lightweight monocular vision pipeline for POV hand-gesture detection using a cost-efficient ESP32 camera module operating at 1080p resolution. Reliable visual input is ensured through image cleaning and enhancement techniques, including Laplacian variance–based blur detection, high-frequency residual analysis for noise detection; and preprocessing pipeline consisting of gamma correction, vignette correction, denoising, sharpening, and grayscale conversion. Camera calibration is performed using checkerboard-based corner detection to estimate intrinsic parameters (focal length and principal point), extrinsic parameters (rotation and translation vectors), and lens distortion coefficients, with reprojection error minimization used to improve geometric accuracy. Feature extraction methods such as Canny edge detection, Hough-based line detection, line filtering, and line intersection analysis are implemented to support robust

corner detection within a defined region of interest. Perspective projection and image-to-world coordinate mapping further enable spatial consistency for region-constrained hand detection. This application demonstrates that accurate and robust egocentric hand-gesture interaction can be achieved without depth sensors, providing a cost-effective and portable alternative for wearable AR/VR devices.

II. TECHNICAL

A. Image Processing and Enhancement

Image processing and enhancement are the first steps in this pipeline, divided into two stages: image cleaning and image preprocessing. The image cleaning stage performs quality assessment to remove low-quality frames. Blur is detected using Laplacian variance, which measures the amount of high-frequency edge information in an image, while noise is estimated using a high-frequency residual obtained by subtracting a Gaussian-smoothed image from the original. Frames identified as blurry or excessively noisy are excluded from further processing. The preprocessing stage enhances the remaining images to improve feature visibility and consistency. This includes gamma correction to normalize intensity response, vignette correction to fix uneven illumination, denoising to reduce sensor noise, sharpening to enhance edges, and grayscale conversion to produce a uniform representation suitable for feature extraction.

1) Cleaning

Laplacian Variance for Blur Detection

Blur detection is performed using the variance of the Laplacian, a second-order derivative operator that highlights high-frequency image content such as edges and fine details. In a sharp image, edges produce strong Laplacian responses, resulting in a high variance, whereas blurred images exhibit attenuated edge responses and lower variance [11]. Given the grayscale image I(x, y), the Laplacian is computed as $L(x, y) = \nabla^2 I(x, y)$ and the blur score is obtained as the variance of the Laplacian: $\sigma_L^2 = \text{var}(L)$.

High-Frequency Residual for Noise Detection

Noise detection is based on estimating the high-frequency residual of an image, which isolates random intensity fluctuations introduced by sensor noise [12]. A Gaussian-smoothed version of the image is first computed to remove structural content, and the residual is obtained by subtracting this smoothed image from the original:

$$R(x, y) = I(x, y) - G_\sigma(x, y)$$

2) Pre-Processing

Image preprocessing is applied to normalize visual input and improve feature reliability.

Gamma correction compensates for the non-linear camera images and varying lighting conditions in first-person views by applying a power-law transformation,

$$I_{gamma} = 255\left(\frac{\mathrm{I}}{255}\right)^{1/\gamma}$$

enhancing mid-tone contrast and improving hand–background separation (Fig. 2b).

Vignette correction mitigates radial illumination falloff caused by compact optics using a distance-based gain function,

$$I_{vignette} = I(x, y)(1 + k \cdot d(x + y)^2)$$

ensuring consistent brightness across the field of view (Fig. 2c).

Denoising suppresses sensor and compression noise while preserving edges, which is critical for stable feature extraction in real-time processing,

$$I_{denoise} = I + \epsilon$$

where ϵ represents noise reduced through edge-preserving filtering (Fig. 2d).

Sharpening, implemented via unsharp masking enhances edge contrast and improves the localization of hand contours and geometric features (Fig. 2e).

$$I_{sharp} = \alpha - (\alpha - 1)G_\sigma(I)$$

Finally, **grayscale conversion** reduces computational complexity by transforming the image into a single intensity channel, and is used for the subsequent edge detection, line extraction, and region-constrained hand detection (Fig. 2f).

$$I_{gray} = 0.299R + 0.587G + 0.114B$$

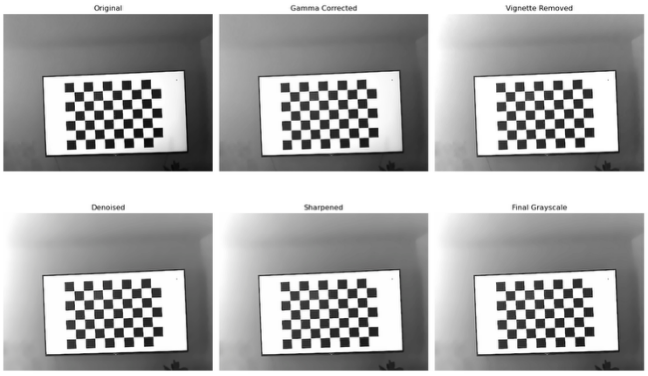

Fig. 2. Preprocessing Images: Original (a), gamma corrected (b), vignetted removed (c), denoised (d), sharpened (e), and grayscale (f)

B. Camera Calibration

Camera calibration is performed using a checkerboard-based approach to estimate both intrinsic and extrinsic camera parameters. A planar checkerboard pattern with known geometry of 9x6 squares is used as a calibration target, and internal corner points are detected and refined to sub-pixel accuracy across multiple views [13]. The corresponding two-dimensional image points and three-dimensional world coordinates are then used to compute the camera's intrinsic parameters, represented by the intrinsic camera matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The intrinsic (K) and extrinsic (R and t) parameters define the 3D world coordinates, X_w, where the 2D image point is,

$$x = K[R|t]X_w$$

Lens distortion coefficients are optimized with the intrinsic and extrinsic parameters through non-linear reprojection error minimization using 50 calibration images captured during setup when the checkerboard is detected [14], resulting in an accurate camera model for precise geometric projection and spatial consistency.

C. Feature Extraaction

Canny Edge Detection

Edge detection is performed using a gradient-based Canny edge detection framework to extract strong structural boundaries from the input image. The image is first converted to grayscale and smoothed using a Gaussian blur to suppress noise and reduce spurious gradients (Fig. 3).

The Canny operator then computes image gradients using first order derivatives,

$$G_x = \frac{\partial I_s}{\partial x}, \; G_y = \frac{\partial I_s}{\partial y}$$

where gradient magnitude and orientation are calculated,

$$|\nabla I| = \sqrt{G_x^2 + G_y^2}, \theta = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Then non-maximum suppression, and dual threshold is used to produce a binary edge map containing prominent edges [15].

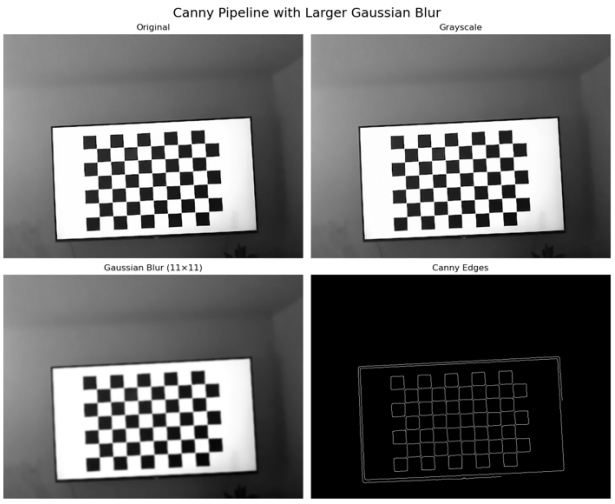

Fig. 3. Canny Edge Detection Pipeline

Hough Transformation

Detected edges are subsequently processed using the probabilistic Hough Line Transform to identify straight line segments within the scene (Fig. 4). The Hough transform represents each edge point (x, y) in a parametric line space,

$$\rho = xcos\theta + ysin\theta$$

and detects line candidates based on accumulator voting, with constraints on minimum line length and allowable gaps to filter out short or fragmented detections [16]. By extracting dominant horizontal and vertical line segments, the system robustly captures the geometric layout of the checkerboard and surrounding planar structures, supporting reliable line filtering, intersection computation, and corner extraction in later stages of the pipeline.

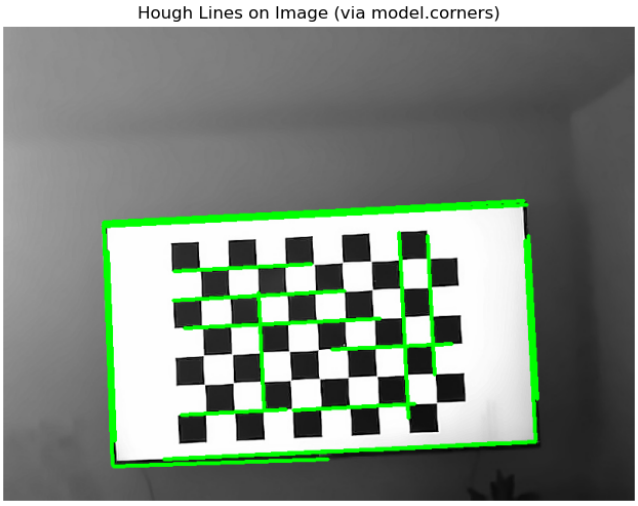

Fig. 4. Probabilistic Hough Transformation

Line Fitting and Extension

Following line detection, extracted line segments are filtered based on their orientation and length to identify dominant horizontal and vertical structures within the image. For each detected line segment defined by endpoints (x_1, y_1) and (x_2, y_2), where the orientation angle is found by,

$$\theta = tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x}\right)$$

and the Euclidean distance is,

$$L = \sqrt{(x_2 - x)^2 + (y_2 - y_1)^2}$$

Short line segments are removed using a minimum length threshold, and lines whose orientations are not close to 0°, 90°, or 180° are discarded. The remaining horizontal and vertical lines are extended to the image boundaries using a linear model, enabling reliable line intersection and corner extraction (Fig. 5).

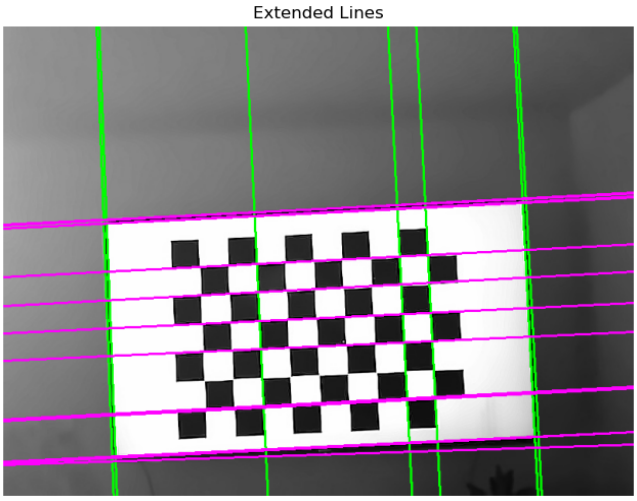

Fig. 5. Line Orientation and Extension

Line Intersection

Line intersections are computed between filtered horizontal and vertical line segments to obtain candidate corner points (Fig. 6). The resulting intersection points provide geometrically consistent corner candidates that support robust corner extraction and region-of-interest–based feature localization in later stages of the pipeline. Intersections are computed by solving,

$$\begin{bmatrix} x_2 - x_1 & x_3 - x_4 \\ y_2 - y_1 & y_3 - y_4 \end{bmatrix} [t \quad u]^T = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \end{bmatrix}$$

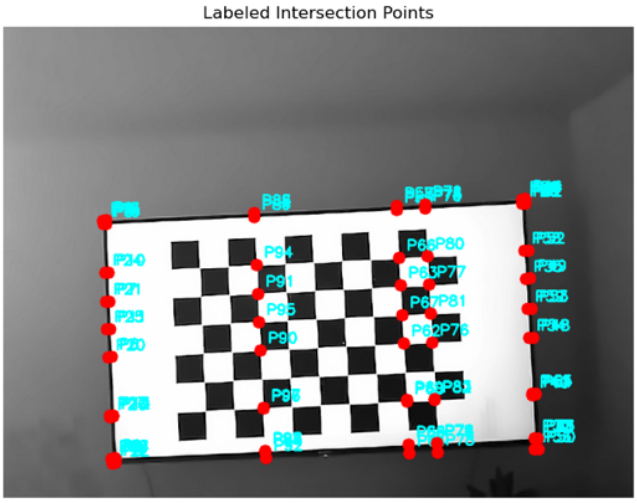

Fig. 6. Intersected Line Segments

D. Corner Detection

Corner Extraction

The four-corner detection stage combines the previous steps and geometric reasoning to localize stable corner points. From these candidate points, the final four corners are selected by identifying the spatial extrema corresponding to the top-left, top-right, bottom-left, and bottom-right points of the planar region, providing a stable and consistent definition of the region of interest (Fig. 7).

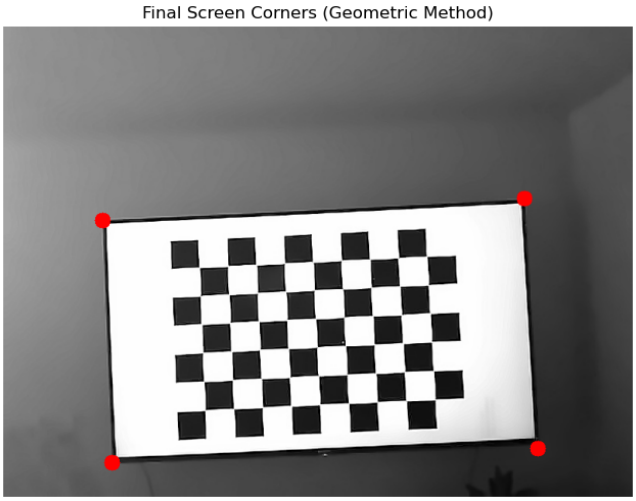

Fig. 7. Four Corner Intersection

Harris-Based Corner Refinement

To improve corner localization accuracy, an additional Harris corner refinement step is applied within a small region of interest (ROI) around each pre-estimated corner (Fig. 8). This localized refinement prevents corner drift into irrelevant regions such as screen interiors or user interface elements. For each candidate corner, a limited search window is extracted and the Harris corner response is computed to identify the strongest corner within that neighborhood [17]. The Harris response is defined as

$$R = det(M) - k[trace(M)]^2$$

where M is the second-moment matrix. This refinement creates a more accurate and stable corner estimates for subsequent processing.

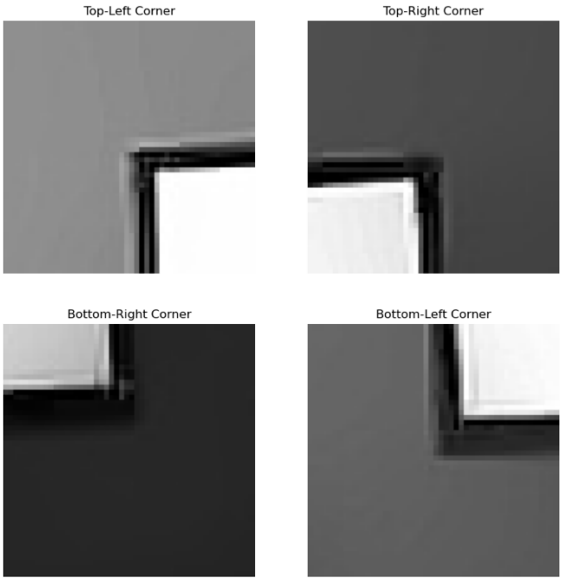

Fig. 8. ROI Harris Corner Detection

E. *Geometric Processing*

2D Image to 3D World Coordinate Mapping

For each calibration image, the camera pose is defined by a rotation matrix **R** and translation vector **t**, which relate 3D world coordinates to the camera coordinate system by $X_c = RX + t$ [14]. These parameters are loaded directly from the saved NPZ calibration data generated during the checkerboard calibration stage and are used to reconstruct the camera's position and orientation in the world reference frame for visualization.

The camera centre, C = (X, Y, Z), is shown in red and is calculated from the transformation $C = -R^T t$ in Fig. 9, where the blue lines represent the camera movement.

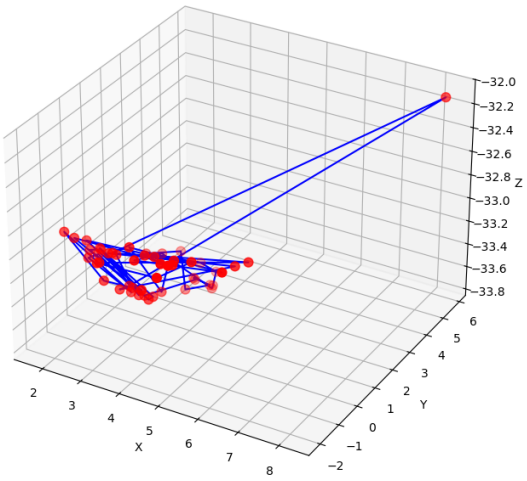

Fig. 9. Calibrated Camera Extrinsics

The checkerboard is reconstructed on the XY-plane at Z=0 in the 3D space with the respective camera point positions (Fig. 10). Camera also shows viewing frustums derived from the estimated rotation matrices and camera centers illustrate camera orientations and viewpoints. These elements provide a clear geometric interpretation of the extrinsic calibration in 3D world space from 2D image.

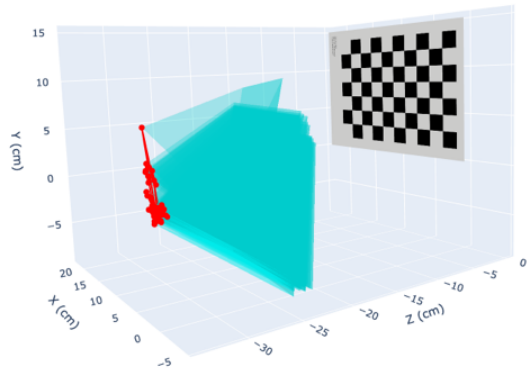

Fig. 10. Calibrated Camera Extrinsics with Frustrum in World Space

F. *Index Finger Tracking*

Hand tracking is performed using a real-time, landmark-based approach on frames streamed from the ESP32 camera. MediaPipe API employs a two-stage pipeline consisting of a palm detector followed by a hand landmark regression model that estimates 21 hand key points [18]. Detection is constrained to a predefined region of interest to reduce false positives and limit processing to the valid interaction area. The index fingertip is tracked by extracting its landmark and projecting its normalized coordinates into image space, with temporal smoothing ensuring stable, low latency tracking for first-person interaction (Fig. 11).

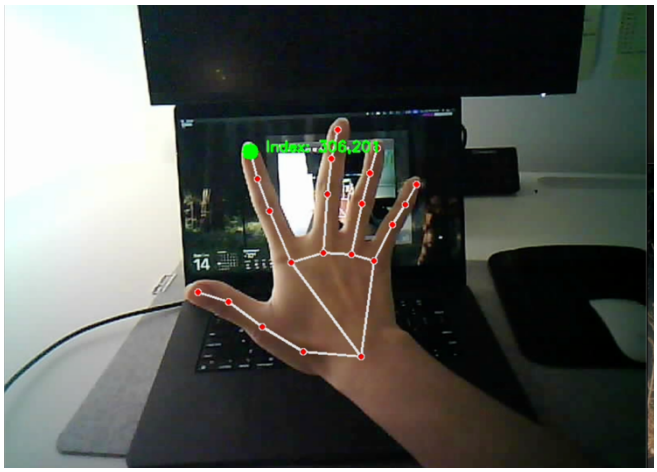

Fig. 11. MediaPipe Hand Tracking

III. EXPERIMENTS

A. *Hardware*

System evaluation was performed using an ESP32 camera module configured to stream 1080p video over a local network. The device was programmed with Arduino-based firmware and interfaced through an FTDI module to deploy an onboard HTTP server for continuous video transmission. The live stream is ingested directly by the Python-based vision pipeline, which executes real-time image processing, camera calibration, and hand-gesture detection. This configuration enables end-to-end assessment of system performance under realistic streaming, latency, and computational constraints.

B. *Software*

The software implementation is developed in Python and uses established computer vision and numerical libraries. OpenCV is used for core image processing and geometric analysis tasks, including grayscale conversion (cv2.cvtColor), Gaussian smoothing (cv2.GaussianBlur), Canny edge detection (cv2.Canny), probabilistic Hough line detection (cv2.HoughLinesP), Harris corner detection (cv2.cornerHarris), and checkerboard-based camera calibration (cv2.findChessboardCorners, cv2.cornerSubPix, cv2.calibrateCamera). MediaPipe Hands is employed for real-time hand detection and tracking, using a two-stage pipeline consisting of palm detection followed by hand

landmark regression to estimate 21 hand keypoints, with index finger landmarks used for gesture interaction. Numerical computations and coordinate transformations are handled using NumPy, while supporting libraries enable visualization, data management, and real-time video stream integration. This modular Python-based implementation enables efficient real-time processing on resource-constrained hardware.

C. Application

The application operates through a structured two-stage pipeline comprising an initialization phase and a real-time interaction phase. During the setup stage, the system captures a sequence of 50 calibration images whenever a checkerboard pattern is detected in the camera stream. These images undergo automated data cleaning to remove frames affected by excessive blur or noise, followed by image preprocessing steps including gamma correction, vignette correction, denoising, sharpening, and grayscale conversion to enhance feature visibility. Using the refined dataset, checkerboard corner correspondences are extracted and used to estimate the camera's intrinsic parameters, including focal length, principal point, and lens distortion coefficients, as well as extrinsic parameters represented by rotation and translation vectors for each view. This calibration establishes a consistent geometric mapping between image space and the physical scene.

D. Results

After the camera has been calibrated, the runtime application loads the computed camera parameters and performs real-time corner detection to define a stable region of interest within the image (Fig. 13). Hand tracking is then executed within this bounded region using a landmark-based detection approach, with the index finger serving as the primary interaction point. Temporal analysis of index finger motion enables the classification of directional gestures, such as up, down, left, and right, constrained to the calibrated corner regions. This integration of geometric calibration, region-based feature detection, and real-time gesture tracking enables robust egocentric interaction while maintaining low computational overhead suitable for resource-constrained hardware platforms.

Hand-gesture interaction is enabled by tracking the user's index finger within the calibrated corner-defined boundary. A leftward swipe gesture, detected as a continuous horizontal displacement of the index fingertip across successive frames, is interpreted as a left tab command to change the displayed screen (Fig. 12).

Fig. 12. Hand Gesture Results from Swiping Left

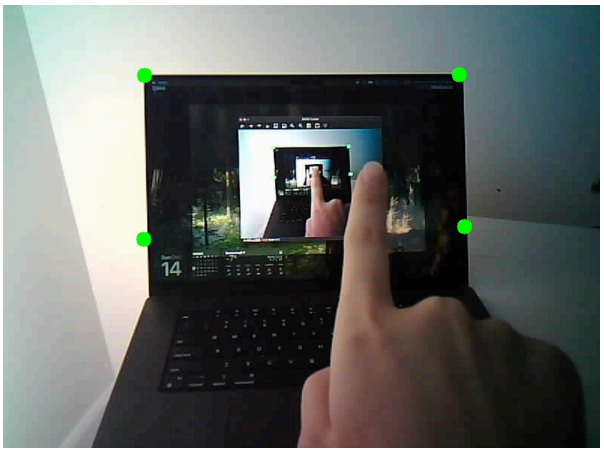

Fig. 13. Screen Capture of Hand Gesture Interaction with Calibrated Corner-Defined Boundary

The complete vision pipeline was evaluated across multiple display mediums, including a laptop screen (Fig. 14), desktop monitor (Fig. 15), and television (Fig. 16), to determine robustness under varying screen sizes, viewing angles and distances. In all tested scenarios, the system successfully detected screen boundaries, established stable regions of interest, and enabled reliable hand-gesture interaction. These results demonstrate that the proposed pipeline generalizes well across different display environments and real-world scenarios.

Pipeline Steps:

1. **Canny Edge Detection:** Extracts strong intensity gradients to identify prominent structural edges within the image.
2. **Hough Line Transformation:** Detects dominant straight-line segments from the edge map using a probabilistic Hough transform.
3. **Line Orientation Filtering:** Classifies detected line segments as horizontal or vertical based on orientation thresholds and removes spurious detections.
4. **Line Extension:** Extends filtered line segments to the image boundaries to form continuous geometric constraints.
5. **Line Intersection Computation:** Computes intersections between horizontal and vertical lines to generate candidate corner points.
6. **Corner Detection and Selection:** Refines and selects stable corner points to define a consistent region of interest.
7. **Extrinsic Estimation and 3D Mapping:** Uses the detected corner geometry to estimate camera extrinsic parameters and map the scene into a three-dimensional coordinate frame.

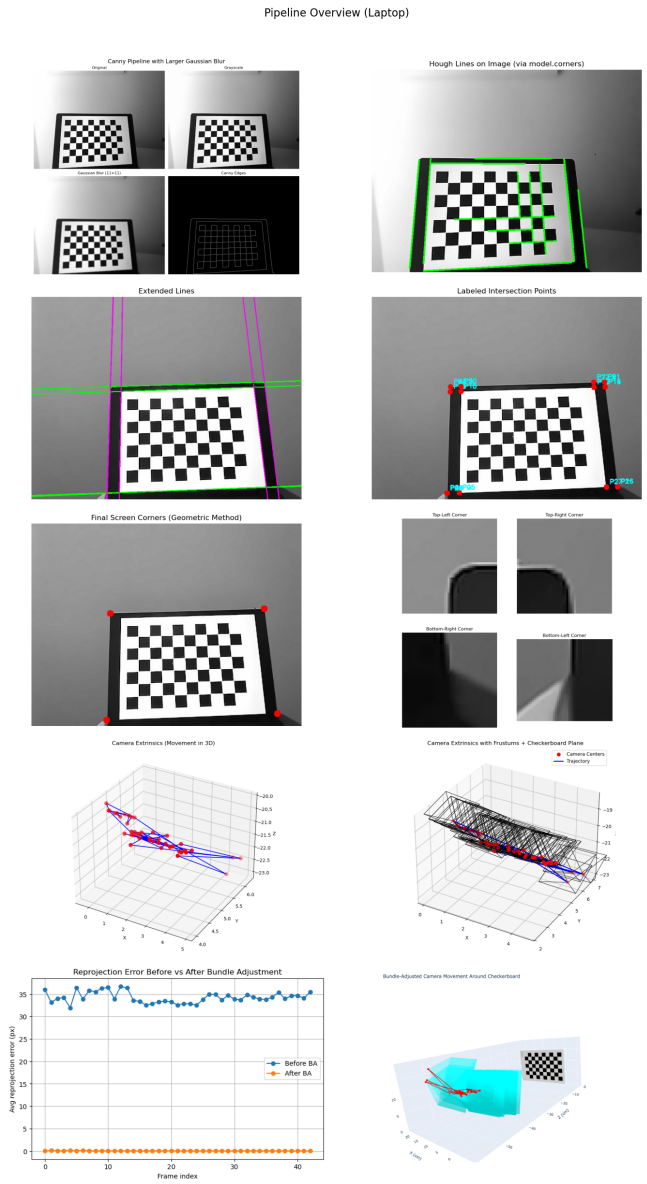

Fig. 14. Calibration Pipeline on Laptop

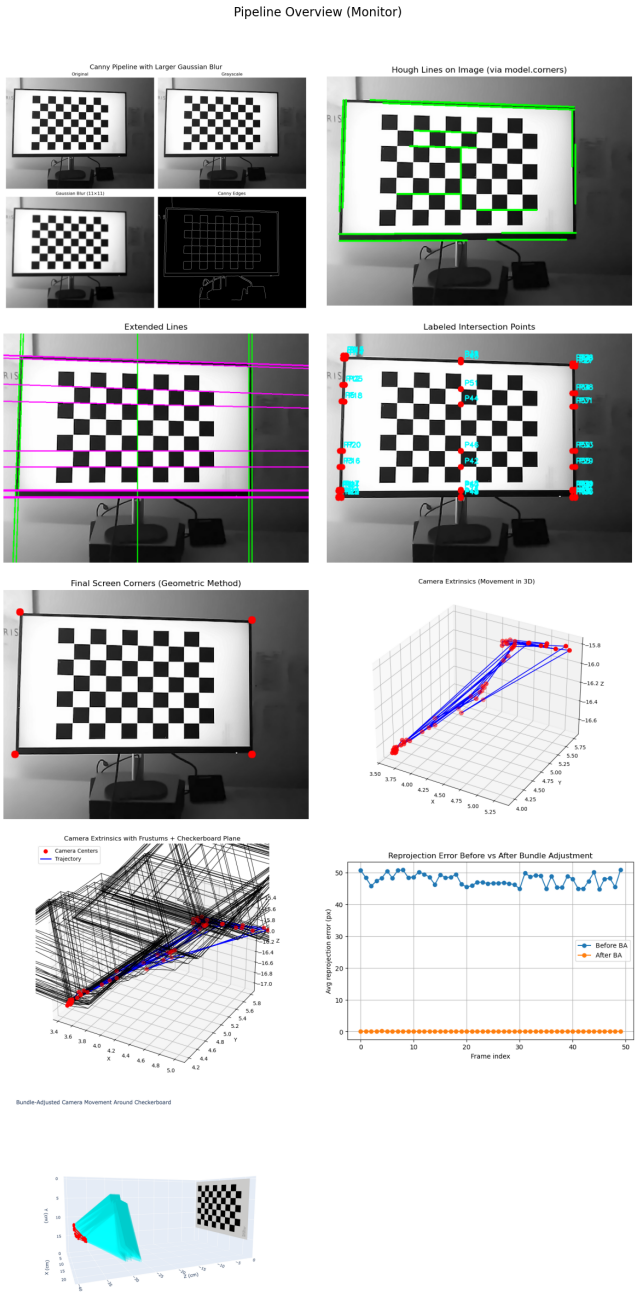

Fig. 15. Calibration Pipeline on Monitor

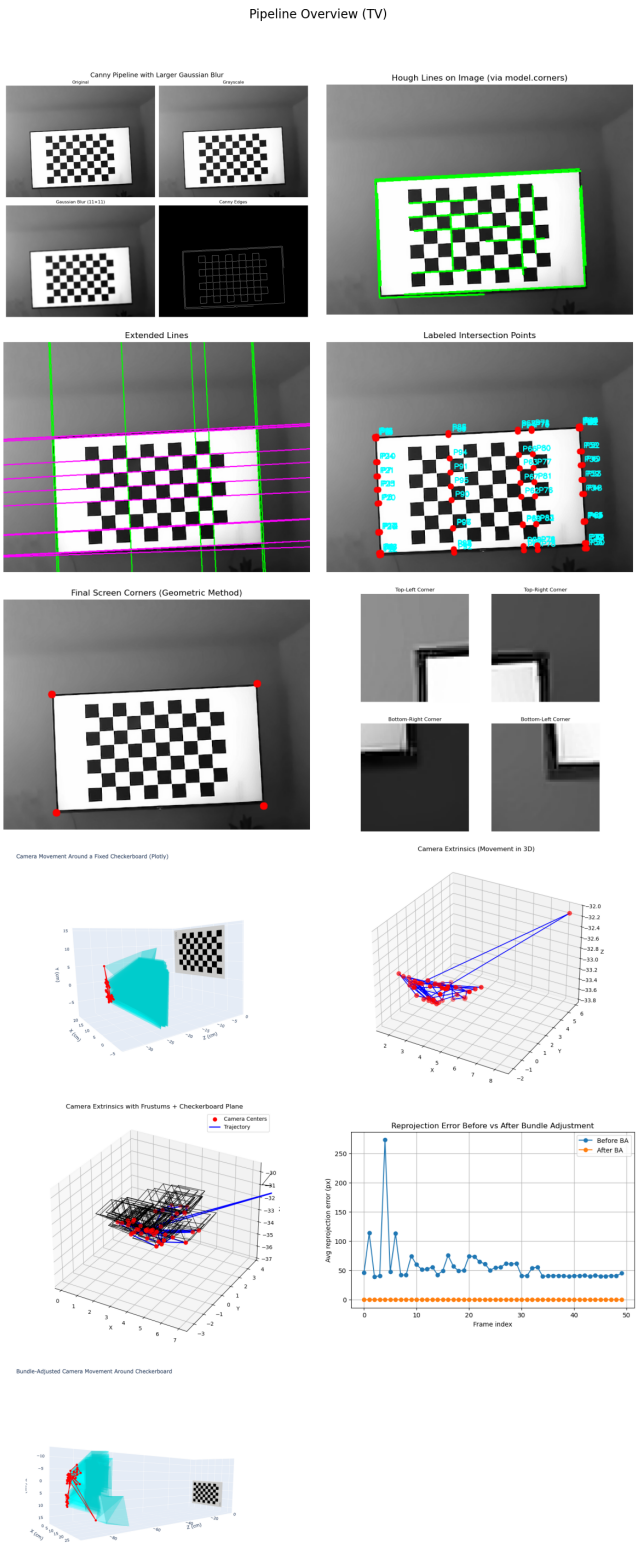

Fig. 16. Calibration Pipeline on TV

IV. CONCLUSION

This application presents a lightweight, vision-only framework for POV (point-of-view) hand-gesture interaction using a low-cost ESP32 camera module. By combining classical computer vision techniques with structured geometric reasoning, the system achieves reliable screen boundary detection, region-of-interest stabilization, and real-time index finger gesture recognition without the use of depth sensors. The pipeline integrates edge detection, line extraction, corner localization, camera calibration, and extrinsic parameter estimation to establish a consistent geometric reference frame for interaction.

To further refine the 3D calibration results, bundle adjustment was applied to optimize the camera's intrinsic and extrinsic parameters across all calibration images. By minimizing the overall reprojection error, this optimization improved geometric consistency and reduced residual alignment errors, resulting in more accurate camera pose estimation and more stable corner localization during interaction [19].

While the current system focuses on directional index finger gestures for basic interaction, the framework can be extended to support a richer set of hand gestures to increase usability and interaction expressiveness. Using additional spatial and temporal relationships between hand landmarks, gestures such as pinch, grab, and rotate could be detected to enable intuitive manipulation of virtual objects, including selection, translation, rotation, and scaling within three-dimensional space [20] (Fig. 17). Ideally, this interaction framework would be integrated into a heads-up display (HUD), where the POV camera continuously detects hand motion within the user's field of view while simultaneously presenting visual feedback and overlaid detections directly onto the display. This approach would allow real-time visualization of detected gestures, interaction boundaries, and virtual content, supporting immersive and useful interaction in AR/VR and wearable technology environments.

Fig. 17. DesignAR Immersive 3D Modeling Application

R EFERENCES

[1] "The rise of AR & VR Technologies Across Key Industries," *Unity*, https://unity.com/resources/rise-of-ar-vr-across-industries (accessed Dec. 14, 2025).

[2] T. A. Syed et al., "In-Depth Review of Augmented Reality: Tracking Technologies, Development Tools, AR Displays, Collaborative AR, and Security Concerns," *Sensors*, vol. 23, no. 1, p. 146, Dec. 2022, doi: 10.3390/s23010146.

[3] Apple Inc., "Introducing Apple Vision Pro: Apple's first spatial computer," *Apple Newsroom*, Jun. 2023. [Online]. Available: https://www.apple.com/ca/newsroom/2023/06/introducing-apple-vision-pro/

[4] Meta Platforms, Inc., "Orion AI glasses: The future of AR glasses technology," *Meta*, 2024. [Online]. Available: https://www.meta.com/emerging-tech/orion/

[5] W. B. Garry, T. J. Ames, M. A. Brandt, S. Slocum, T. G. Grubb, and J. L. Heldmann, "Virtual analog environments: Exploring LiDAR data in virtual reality," *in Proc. 49th Lunar and Planetary Science Conf. (LPSC),* The Woodlands, TX, USA, 2018, paper no. 2424. [Online]. Available: https://www.hou.usra.edu/meetings/lpsc2018/pdf/2424.pdf

[6] J. Sánchez, N. Monzón, and A. Salgado, "An Analysis and Implementation of the Harris Corner Detector," *Image Processing On Line,* vol. 8, pp. 305–328, Oct. 2018, doi: 10.5201/ipol.2018.229.

[7] C. Lu, X. Qi, K. Ding, and B. Yu, "An Improved FAST Algorithm Based on Image Edges for Complex Environment," *Sensors*, vol. 22, no. 19, p. 7127, Sept. 2022, doi: 10.3390/s22197127.

[8] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[9] A. Hameed, S. Möller, and A. Perkis, "How good are virtual hands? Influences of input modality on motor tasks in virtual reality," *Journal of Environmental Psychology*, vol. 92, p. 102137, Dec. 2023, doi: 10.1016/j.jenvp.2023.102137.

[10] W. Torres et al., "A Framework for Real-Time Gestural Recognition and Augmented Reality for Industrial Applications," *Sensors*, vol. 24, no. 8, p. 2407, Apr. 2024, doi: 10.3390/s24082407.

[11] A. Kheradmand and P. Milanfar, "Motion deblurring with graph Laplacian regularization," in *Proc*. SPIE, vol. 9020, 2014. [Online]. Available: https://users.soe.ucsc.edu/~milanfar/publications/conf/Deblur_Graph_SPIE.pdf

[12] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning Low-Level Vision," *International Journal of Computer Vision*, vol. 40, no. 1, pp. 25–47, Oct. 2000, doi: 10.1023/A:1026501619075.

[13] B. Chen, C. Xiong, and Q. Zhang, "CCDN: Checkerboard corner detection network for robust camera calibration," *arXiv preprint*, arXiv:2302.05097, 2023. [Online]. Available: https://arxiv.org/pdf/2302.05097

[14] OpenCV, "Camera Calibration," *OpenCV Documentation*, 2024. [Online]. Available: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

[15] OpenCV, "Canny edge detection," *OpenCV Documentation*, 2024. [Online]. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

[16] OpenCV, "Hough line transform," *OpenCV Documentation*, 2023. [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

[17] OpenCV, "Harris corner detection," *OpenCV Documentation*, 2024. [Online]. Available: https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html

[18] Google, "MediaPipe Hands," *MediaPipe Documentation*, 2024. [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

[19] N. Sameer, N. Snavely, S. M. Seitz, and R. Szeliski, "Bundle adjustment in the large," *Univ. of Washington, Seattle*, WA, USA, Tech. Rep., 2010. [Online]. Available: https://grail.cs.washington.edu/projects/bal/bal.pdf

[20] P. Reipschläger and R. Dachselt, "DesignAR: Immersive 3D-Modelling Combining Augmented Reality with Interactive Displays," in *Proceedings of the 2019 ACM International Conference on Interactive Surfaces and Spaces,* Daejeon Republic of Korea: ACM, Nov. 2019, pp. 29–41. doi: 10.1145/3343055.3359718.