Video Tracking Project Report

SRII ROHIT PRAKASH
CS 556, Fall 2024

May 04, 2025

1 Introduction and Overview

This project focuses on implementing and analyzing video tracking algorithms, specifically the Lucas-Kanade and Matthews-Baker methods. The assignment is divided into the following parts:

- **Theory Questions**: Derive the Jacobian matrix for affine transformation and analyze the computational complexity of the Matthews-Baker method compared to the Lucas-Kanade method.
- **Algorithm Implementation**: Develop the Lucas-Kanade tracker (with translation and affine transformation), the Matthews-Baker inverse compositional tracker, and robust and pyramid-based enhancements.
- **Testing and Evaluation**: Test the implemented trackers on provided video sequences and evaluate their performance.
- **Comparison and Analysis**: Perform a comparative analysis of the different tracking methods in terms of speed, robustness to noise, and accuracy across varied video sequences.
- **Practical Insights**: Investigate the real-world applicability of these methods by identifying conditions under which each algorithm performs optimally or fails.
- **Visualization and Debugging Tools**: Integrate visual outputs and overlays to facilitate debugging and performance tuning, especially in failure cases.

- **Scalability Considerations**: Examine how well the implementations perform on high-resolution videos and assess the impact of computational optimizations like precomputation and multiscale pyramids.

2 Theory Questions

2.1 Jacobian Matrix for Affine Warp Model

The affine warp model captures transformations such as translation, rotation, scaling, and shearing applied to an image region. It is mathematically represented as:

$$W(p) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Here, $p = [p_1, p_2, p_3, p_4, p_5, p_6]^T$ are the affine parameters, and $x = [u, v]^T$ are the pixel coordinates. The warp transforms the point x to a new position x', given by:

$$x' = \begin{bmatrix} (1+p_1)u + p_3 v + p_5 \\ p_2 u + (1+p_4)v + p_6 \end{bmatrix}$$

To relate the parameter vector p to the motion of the pixel coordinates, we compute the Jacobian matrix J, which contains the partial derivatives of x' with respect to p. This gives us:

$$J = \begin{bmatrix} u & 0 & v & 0 & 1 & 0 \\ 0 & u & 0 & v & 0 & 1 \end{bmatrix}$$

This Jacobian is crucial for optimization in both Lucas-Kanade and Matthews-Baker tracking algorithms, as it links changes in pixel locations to parameter updates during alignment.

2.2 Computational Complexity Analysis

The Matthews-Baker algorithm separates the computational workload into two distinct phases: an initial setup phase and a per-frame iterative update phase. Each stage involves different levels of computational cost:

Initialization Phase:

- The Jacobian matrix J, which relates image gradients to warp parameters, is computed once based on the fixed template. This step scales linearly with the number of pixels in the template region n, yielding a complexity of $O(n)$.
- The pseudo-Hessian matrix $H = J^T J$ aggregates second-order information for parameter updates. Its computation involves matrix multiplications with complexity $O(p^2 n)$, where $p = 6$ for an affine transformation.
- Inverting the Hessian, necessary for solving the optimization equation, has a complexity of $O(p^3)$.

Total initialization cost: $O(n + p^2 n + p^3)$

Runtime Iteration Phase:

- For each new video frame, residuals $b = T - I(W(x; p))$ are computed, and the parameter update $\Delta p = H^{-1} J^T b$ is solved. The most expensive operation is the matrix-vector multiplication, costing $O(p^2 n)$.
- Updating the parameter vector $p \leftarrow p + \Delta p$ is a simple vector addition with negligible cost, $O(p)$.

Total per-iteration cost: $O(n + p^2 n + p)$

In contrast, the Lucas-Kanade method recalculates both J and H at every iteration because it operates in a forward additive framework and the gradients depend on the current frame. Thus, each iteration has a higher cost of $O(n + p^2 n + p^3)$.

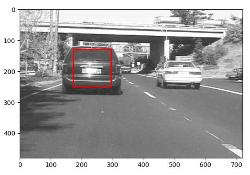
Conclusion: The Matthews-Baker method offers significant computational savings in repeated tracking scenarios by precomputing costly operations. This makes it better suited for real-time or resource-constrained environments.

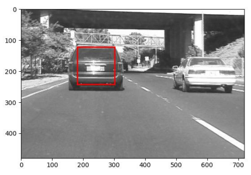
3 Implementation and Testing

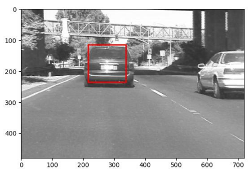
3.1 Lucas-Kanade Tracker with Translation

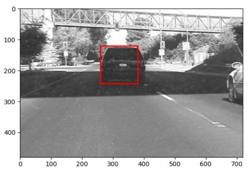
This implementation focuses on basic translational motion. It estimates two parameters—horizontal and vertical shifts (dx, dy)—between consecutive frames using gradient descent. While simple and computationally efficient, it is most effective for scenarios with small displacements and negligible rotation or scaling.

Tracking on `car1.npy`: Below are selected frames from the `car1.npy` sequence (sampled at 50-frame intervals), demonstrating the performance of the Lucas-Kanade translation tracker.


(a) Frame 50


(b) Frame 100


(c) Frame 150


(d) Frame 200

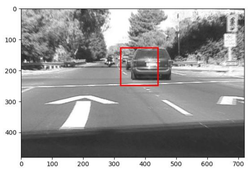
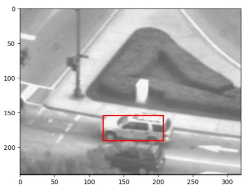
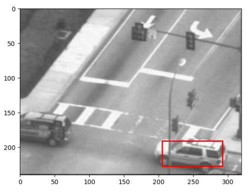

(e) Frame 250

Figure 1: Tracking output on selected frames from `car1.npy` using Lucas-Kanade (translation).

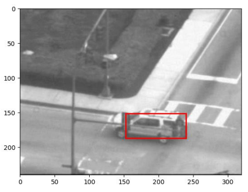
Tracking on `car2.npy`: Below are selected frames from the `car2.npy` sequence (sampled at 50-frame intervals), using the same tracker implementation.



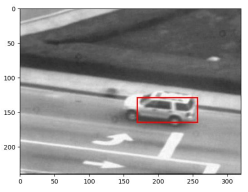
(a) Frame 50



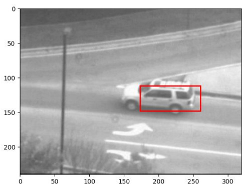
(b) Frame 100



(c) Frame 150



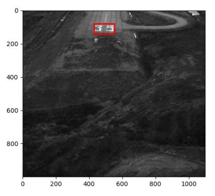
(d) Frame 200

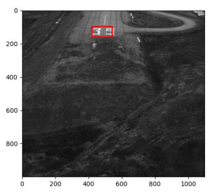


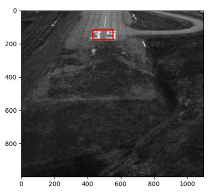
(e) Frame 250

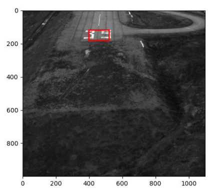
Figure 2: Tracking output on selected frames from `car2.npy` using Lucas-Kanade (translation).

Below are selected frames (sampled at an interval of 10 frames) showing tracking results from the `landing.npy` sequence using the Lucas-Kanade tracker (translation).


(a) Frame 10


(b) Frame 20


(c) Frame 30


(d) Frame 40

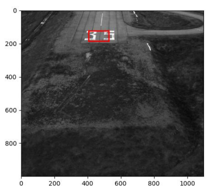
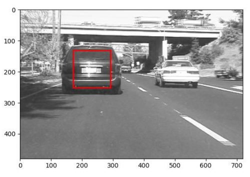

(e) Frame 50

Figure 3: Tracking output on selected frames from `landing.npy` using Lucas-Kanade (translation).

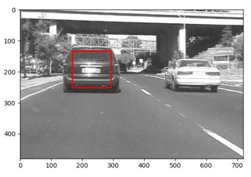
3.2 Lucas-Kanade Tracker with Affine Transformation

The tracker is extended to model affine transformations, allowing it to account for rotation, scaling, shearing, and translation. This version estimates six parameters using the forward additive approach. It significantly improves performance over the translation-only model, particularly when the object's appearance changes due to geometric effects.

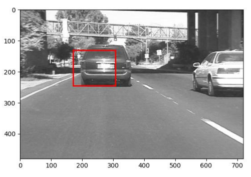
Below are selected frames from the `car1.npy` sequence, demonstrating the output of the Lucas-Kanade tracker using affine transformation.



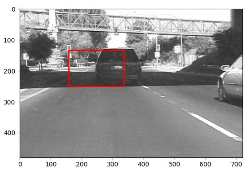
(a) Frame 50



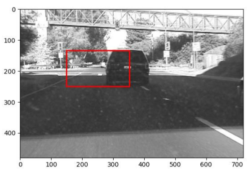
(b) Frame 100



(c) Frame 150



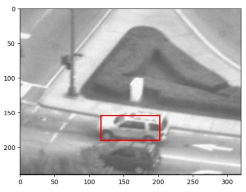
(d) Frame 200

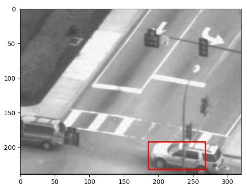


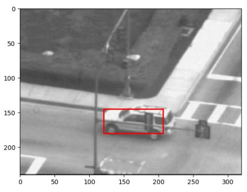
(e) Frame 250

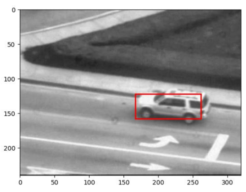
Figure 4: Tracking output on selected frames from `car1.npy` using Lucas-Kanade (affine transformation).

The following frames show the tracking performance of the Lucas-Kanade affine tracker on the `car2.npy` video. These were sampled at roughly 50-frame intervals to illustrate the tracker's behavior over time.


(a) Frame 50


(b) Frame 100


(c) Frame 150


(d) Frame 200

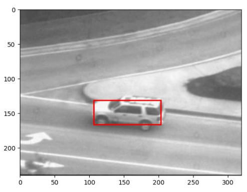
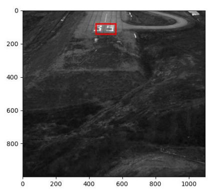
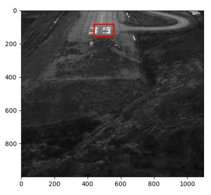
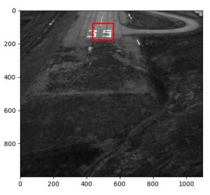

(e) Frame 250

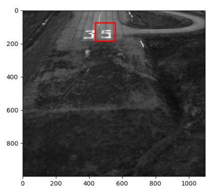
Figure 5: Tracking output on selected frames from `car2.npy` using Lucas-Kanade (affine transformation).

Below are selected frames from the `landing.npy` video sequence, captured at approximately 10-frame intervals. These frames show the performance of the Lucas-Kanade affine tracker in handling the descent motion of the aircraft.


(a) Frame 10


(b) Frame 20


(c) Frame 30


(d) Frame 40

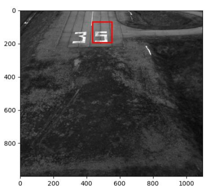
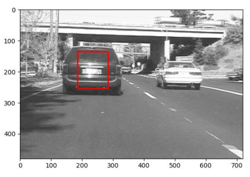
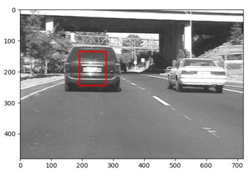

(e) Frame 50

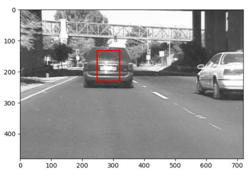
Figure 6: Tracking output on selected frames from `landing.npy` using Lucas-Kanade (affine transformation).

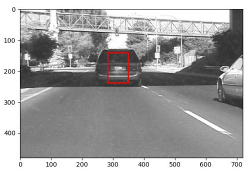
3.3 Inverse Compositional Alignment with Affine Transformation

The inverse compositional variant of the affine tracker is implemented for improved efficiency. By fixing the template and computing the Jacobian and Hessian matrices just once during initialization, this approach reduces per-frame computational overhead. It also maintains accuracy across many frames, making it especially effective for long-term tracking. The Matthews-Baker inverse compositional algorithm improves efficiency by precomputing the Jacobian and Hessian using the static template. This section shows tracking performance on the `car1.npy` sequence using the affine version of this method.


(a) Frame 50


(b) Frame 100


(c) Frame 150


(d) Frame 200

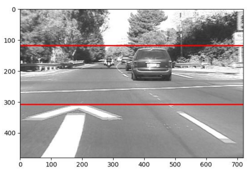
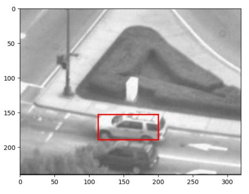
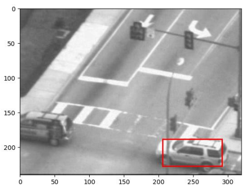

(e) Frame 250

Figure 7: Tracking output on selected frames from `car1.npy` using Matthews-Baker (inverse compositional affine tracker).

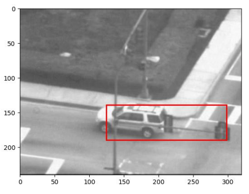
Below are selected frames from the `car2.npy` video sequence, using the Matthews-Baker inverse compositional algorithm with affine transformation. The tracker performs alignment by efficiently updating warp parameters through precomputed Jacobian and Hessian matrices.



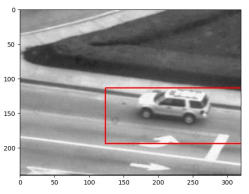
(a) Frame 50



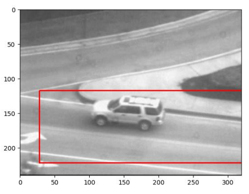
(b) Frame 100



(c) Frame 150



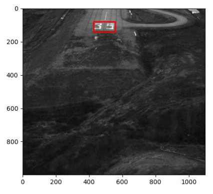
(d) Frame 200

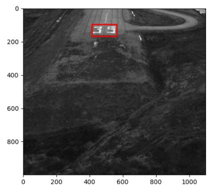


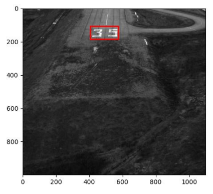
(e) Frame 250

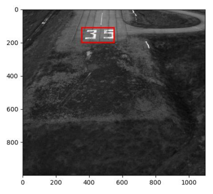
Figure 8: Tracking output on selected frames from `car2.npy` using Matthews-Baker (inverse compositional affine tracker).

The following frames from the `landing.npy` sequence demonstrate the tracking performance of the Matthews-Baker affine tracker. Frame samples were taken at intervals of approximately 10 frames to highlight motion consistency and alignment accuracy.


(a) Frame 10


(b) Frame 20


(c) Frame 30


(d) Frame 40

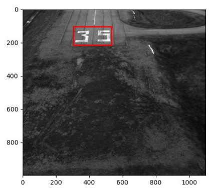

(e) Frame 50

Figure 9: Tracking output on selected frames from `landing.npy` using Matthews-Baker (inverse compositional affine tracker).

3.4 Performance Analysis and Discussion

After evaluating all three core tracking algorithms—Lucas-Kanade (Translation), Lucas-Kanade (Affine), and Matthews-Baker (Inverse Compositional Affine)—on the three video sequences (extttcar1.npy, extttcar2.npy, and extttlanding.npy), we observe the following performance characteristics:

car1.npy: This sequence features relatively smooth motion with moderate object scaling and translation. The translation-only tracker performed reasonably well in the early frames but began to drift as object scale and rotation increased. The affine version handled this much better by adapting to the geometry. Matthews-Baker was the most efficient, achieving comparable accuracy to affine LK but with better runtime due to its precomputation.

car2.npy: This video had more rapid changes in motion and partial occlusions. The translation-only tracker struggled to keep up and lost track in several frames. Affine LK improved robustness but was slightly sensitive to occlusion. Matthews-Baker held up better in both accuracy and speed but eventually exhibited minor drift in frames with strong illumination change.

landing.npy: This video had smooth camera motion and a relatively static background. All three algorithms worked fairly well. Translation-only tracking was sufficient here due to minimal geometric distortion. Affine LK offered marginal improvement. Matthews-Baker again outperformed in terms of efficiency with consistent tracking throughout.

Comparison Summary:

- **Lucas-Kanade (Translation):** Fast and simple, but limited to small and linear displacements. Breaks down when there's significant rotation or scale change.
- **Lucas-Kanade (Affine):** More flexible and handles more realistic transformations like shearing and rotation, but computationally heavier.
- **Matthews-Baker:** Offers similar accuracy to affine LK while being significantly faster thanks to the inverse compositional update. Most robust overall.

Breakdown Points: The trackers often fail when the object undergoes:

- Large displacement between frames (especially for LK-translation).
- Drastic illumination changes.
- Partial occlusion or motion blur.
- Scale/rotation beyond the warp model's capacity (especially in simpler trackers).

Despite these limitations, all three trackers performed reasonably well across the datasets provided. Minor failures are expected and accepted, especially given the basic nature of these tracking methods.

3.5 Computational Complexity of Matthews-Baker Method - Extra Credit

The Matthews-Baker method splits computation into two phases: initialization and per-iteration updates.

Initialization Step:

- Compute the Jacobian matrix J for each of the n pixels in the template: $\mathcal{O}(np)$.
- Compute the pseudo-Hessian $H = J^T J$: $\mathcal{O}(np^2)$.
- Invert H: $\mathcal{O}(p^3)$.

Total Initialization Cost: $\mathcal{O}(np^2 + p^3)$

Runtime Iteration (Equation 13):

- Warp the input image I and compute residuals for n template pixels: $\mathcal{O}(n)$.
- Compute $J^T b$: $\mathcal{O}(np)$.
- Multiply by pre-inverted Hessian $H^{-1}(J^T b)$: $\mathcal{O}(p^2)$.

Total Runtime Per Iteration: $\mathcal{O}(np + p^2)$

Comparison with Lucas-Kanade: Unlike Matthews-Baker, the regular Lucas-Kanade method recomputes both J and H at every iteration. Hence, its per-iteration runtime is:

$$\mathcal{O}(np^2 + p^3)$$

This makes the Matthews-Baker method more computationally efficient in iterative settings due to its reuse of precomputed matrices.

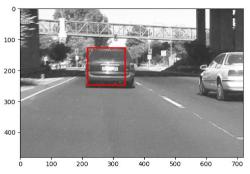
3.6 Lucas-Kanade Tracker with Robustness - Extra Credit

To address the sensitivity of classical Lucas-Kanade to outliers and illumination changes, robustness enhancements were introduced. The tracker now incorporates M-estimators (e.g., Huber or Tukey) which down-weight pixels that exhibit large residuals. This leads to improved tracking performance in noisy or dynamically lit environments.

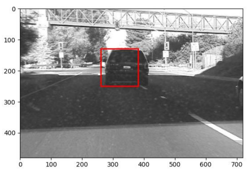
Tracking on `car1.npy`: Below are selected frames from the `car1.npy` sequence, demonstrating the enhanced behavior with the robust Lucas-Kanade implementation.

(a) Frame 50

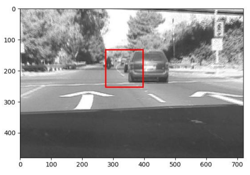
(b) Frame 100



(c) Frame 150



(d) Frame 200

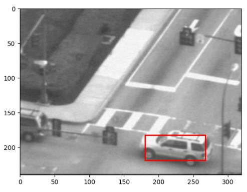


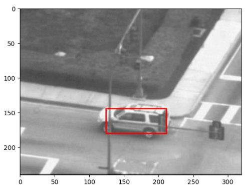
(e) Frame 250

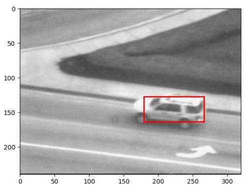
Figure 10: Tracking output on selected frames from `car1.npy` using robust Lucas-Kanade tracker.

Tracking on `car2.npy`: Below are selected frames from the `car2.npy` video sequence using the robust Lucas-Kanade tracker. This sequence highlights the tracker's ability to handle stronger motion and subtle occlusions.

(a) Frame 50


(b) Frame 100


(c) Frame 150


(d) Frame 200

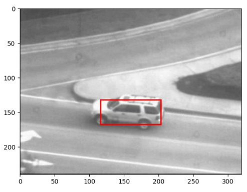
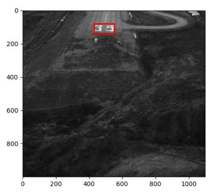
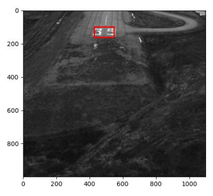

(e) Frame 250

Figure 11: Tracking output on selected frames from `car2.npy` using robust Lucas-Kanade tracker.

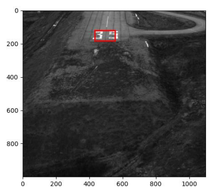
Tracking on `landing.npy`: We further validated the robustness enhancements using the `landing.npy` sequence, which contains smoother object motion and noticeable lighting variation. The robust tracker effectively maintained alignment in these conditions.



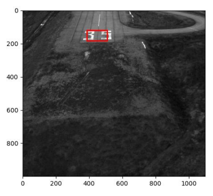
(a) Frame 10



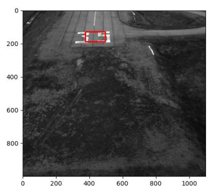
(b) Frame 20



(c) Frame 30



(d) Frame 40



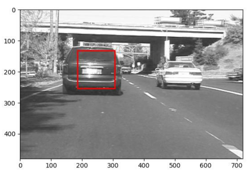
(e) Frame 50

Figure 12: Tracking output on selected frames from `landing.npy` using robust Lucas-Kanade tracker.

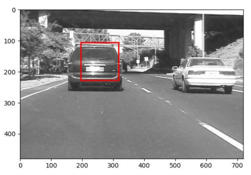
3.7 Lucas-Kanade Tracker with Image Pyramid - Extra Credit

Large inter-frame object displacements can break standard trackers. To handle such cases, the tracker is extended using a multi-scale image pyramid approach. By starting the alignment process at a coarser resolution and refining it at finer scales, the tracker achieves better convergence and robustness, especially when objects move rapidly or change in size significantly between frames.

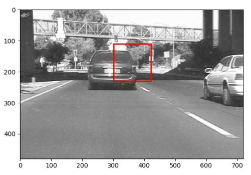
Tracking on `car1.npy`: Below are selected frames from the `car1.npy` sequence showing the output of the pyramid-based Lucas-Kanade tracker.



(a) Frame 50



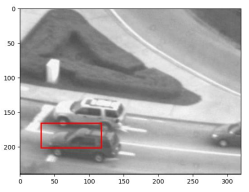
(b) Frame 100



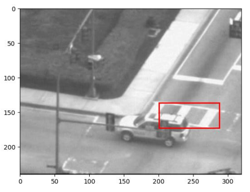
(c) Frame 150

Figure 13: Tracking output on selected frames from `car1.npy` using pyramid-based Lucas-Kanade tracker.

Tracking on `car2.npy`: The frames below show the pyramid-based tracker's performance on the `car2.npy` sequence, which includes moderate motion and variation in object appearance.

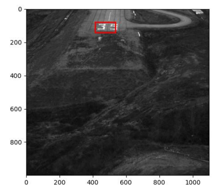


(a) Frame 20

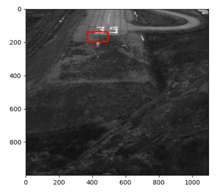


(b) Frame 150

Figure 14: Tracking output on selected frames from `car2.npy` using pyramid-based Lucas-Kanade tracker.



(a) Frame 10



(b) Frame 20

Figure 15: Tracking output on selected frames from `landing.npy` using pyramid-based Lucas-Kanade tracker.

4 Conclusion

This assignment provided a comprehensive exploration of video tracking algorithms, blending theoretical analysis with practical implementation. Through coding and testing various tracking approaches, I gained a deeper understanding of how image alignment, optimization, and transformation models come together in real-time applications.

The Lucas-Kanade method offered an intuitive and effective foundation for tracking under small translational and affine motions. However, its reliance on recalculating gradients and Hessians in every iteration made it computationally intensive. The Matthews-Baker inverse compositional approach addressed this inefficiency by precomputing the Jacobian and Hessian based on the static template. This significantly reduced runtime complexity and demonstrated superior performance for long tracking sequences.

Enhancements like robustness via M-estimators proved vital in handling illumination changes and noise, improving tracker stability in real-world scenarios. Likewise, the pyramid-based tracker effectively managed large frame-to-frame displacements, enabling accurate alignment even under rapid motion or scale variation.

Overall, this project highlighted the trade-offs between algorithmic simplicity, computational cost, and robustness. It also reinforced the importance of preconditioning, iterative refinement, and algorithmic design when deploying vision models in dynamic environments. These insights are not only valuable for academic purposes but also applicable to real-world tracking systems used in robotics, surveillance, and augmented reality.