CSL7360: Computer Vision
Assignment 3

Atharva Date
Roll Number: B22AI045

October 4, 2025

Question 1: Lucas-Kanade Optical Flow

Implementation

I implemented Lucas-Kanade optical flow from scratch with the following components:

Harris Corner Detector

The Harris corner detector was implemented using the following steps:

1. Compute image gradients I_x and I_y using Sobel operators
2. Compute products of derivatives: $I_{xx} = I_x^2$, $I_{yy} = I_y^2$, $I_{xy} = I_x I_y$
3. Apply Gaussian window to compute structure tensor elements S_{xx}, S_{yy}, S_{xy}
4. Calculate corner response: $R = \det(M) - k \cdot \text{trace}(M)^2$ where $M = \begin{bmatrix} S_{xx} & S_{xy} \\ S_{xy} & S_{yy} \end{bmatrix}$
5. Apply non-maximum suppression and threshold to extract corners

Lucas-Kanade Optical Flow

The Lucas-Kanade method computes optical flow using a patch-based least squares approach:

1. For each corner point, extract a window of size $w \times w$
2. Compute spatial gradients I_x, I_y and temporal gradient $I_t = I_2 - I_1$
3. Solve the least squares system: $A^T A\mathbf{v} = -A^T b$ where $A = [I_x, I_y]$ and $b = I_t$
4. The solution $\mathbf{v} = [u, v]^T$ gives the optical flow vector

Results

(a) Frame 1 (frame10.png)

(b) Frame 2 (frame11.png)

Figure 1: Input frames for optical flow computation

Figure 2: Harris corners detected in the first frame

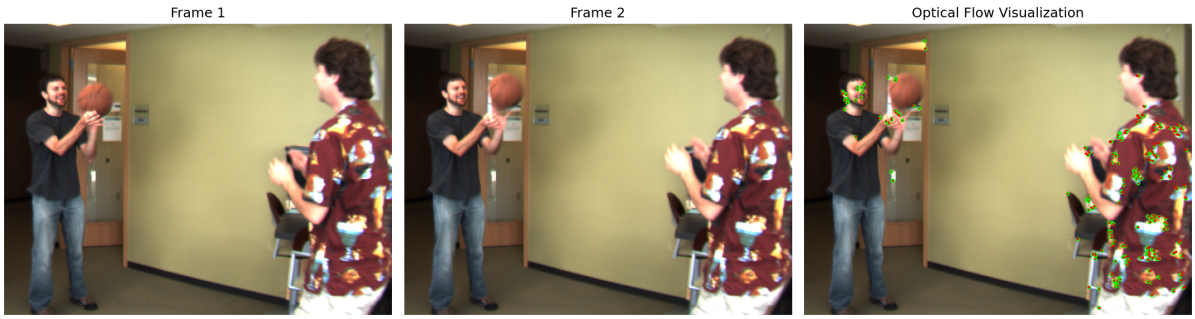

Figure 3: Optical flow visualization: (Left) Frame 1, (Middle) Frame 2, (Right) Tracked motion with displacement arrows

Figure 4: Detailed view of optical flow vectors overlaid on Frame 2

Analysis and Limitations

Limitations of Lucas-Kanade Optical Flow:

1. Aperture Problem: The Lucas-Kanade method assumes constant motion within a local window. When tracking edges or lines, motion perpendicular to the edge cannot be determined uniquely, as only the normal component of flow is observable. This leads to ambiguous flow estimates.

2. Brightness Constancy Assumption: The method assumes pixel intensities remain constant between frames ($I(x, y, t) = I(x+u, y+v, t+1)$). This fails under illumination changes, shadows, or specular reflections.

3. Small Motion Assumption: Lucas-Kanade uses first-order Taylor approximation, valid only for small displacements (typically <5 pixels). Large motions violate the linear assumption and cause tracking failure.

4. Texture Dependency: The structure tensor M must be invertible, requiring sufficient texture. Regions with uniform intensity (e.g., blank walls) have singular M, making flow computation ill-posed.

Failure Case: Consider a fast-moving object with displacement >15 pixels between frames. The linear approximation $I_t \approx I_x u + I_y v$ breaks down, resulting in erroneous flow vectors pointing in wrong directions or with incorrect magnitudes. The tracked features appear to "jump" rather than smoothly transition.

Question 2: Graph Cut Segmentation

Implementation

I implemented binary image segmentation using graph cuts with the following energy formulation:

Energy Function

The segmentation minimizes the energy:

$$E(L) = \sum_{p \in P} D_p(L_p) + \lambda \sum_{(p,q) \in \mathcal{N}} V_{pq}(L_p, L_q)$$

where:

- **Data Term** $D_p(L_p)$: Likelihood of pixel p belonging to foreground/background based on Gaussian color models. For each seed set (foreground/background), I estimate mean μ and covariance Σ, then compute:

$$D_p(\text{fg}) = -\log \mathcal{N}(I_p | \mu_{\text{fg}}, \Sigma_{\text{fg}})$$

$$D_p(\text{bg}) = -\log \mathcal{N}(I_p | \mu_{\text{bg}}, \Sigma_{\text{bg}})$$

- **Smoothness Term** V_{pq}: Potts model penalizing label discontinuity:

$$V_{pq}(L_p, L_q) = \lambda \exp\left(-\frac{\|I_p - I_q\|}{10}\right) \cdot [L_p \neq L_q]$$

Graph Construction

1. Create nodes for each pixel
2. Add terminal edges (source/sink) with weights from data term
3. Add neighbor edges (4-connectivity) with weights from smoothness term
4. Solve min-cut/max-flow using PyMaxflow library

Seed Specification

Seeds are specified programmatically in the code:

- **Foreground seeds:** Green circles placed on object regions
- **Background seeds:** Red circles placed on background regions

Figure 5: Original bird image for segmentation

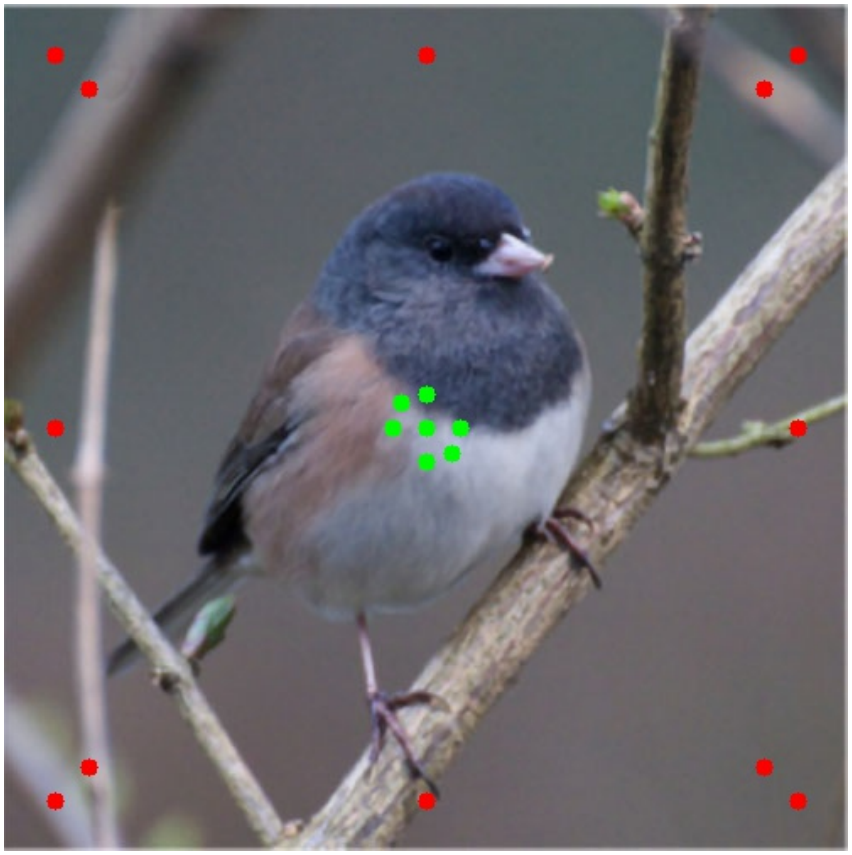

Figure 6: Bird image with foreground (green) and background (red) seed placement

Results with Varying Smoothness Weight

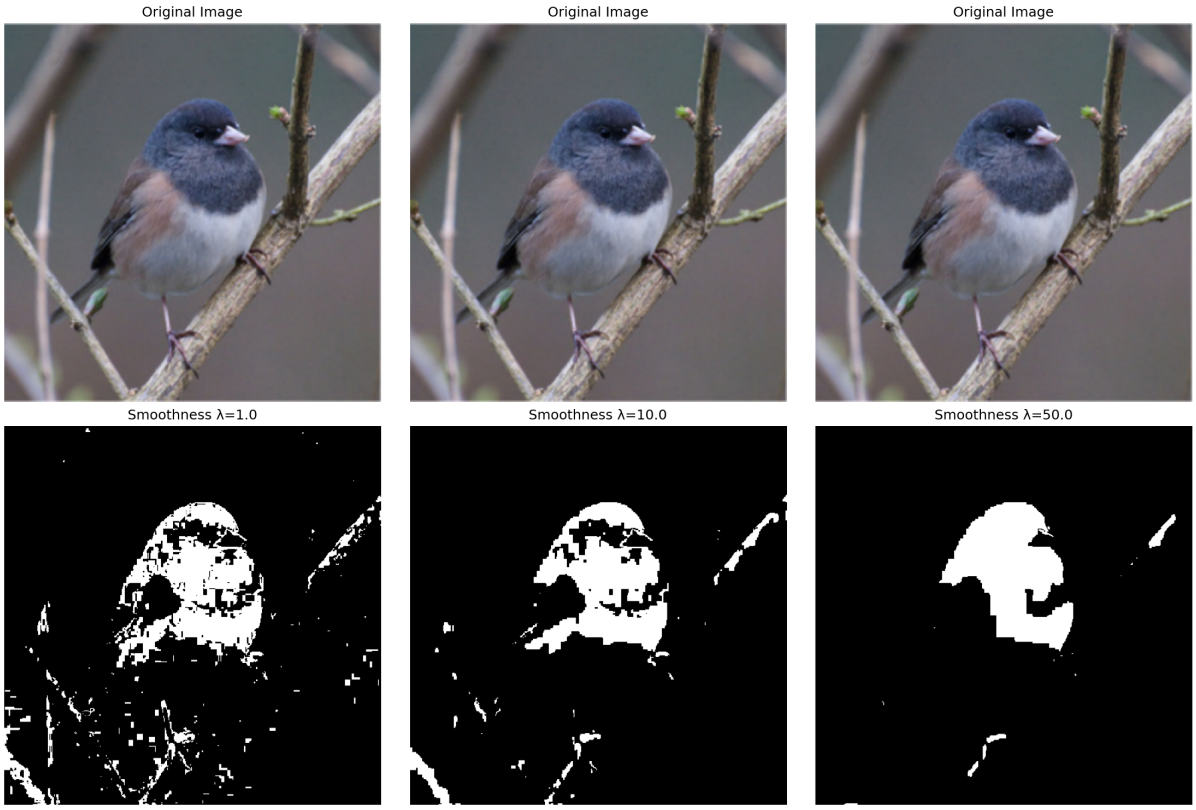

Figure 7: Bird image: Effect of smoothness weight λ on segmentation. Top row shows original image, bottom row shows segmentation results for $\lambda = 1.0$, $\lambda = 10.0$, and $\lambda = 50.0$.

(a) $\lambda = 1.0$ (noisy) (b) $\lambda = 10.0$ (balanced) (c) $\lambda = 50.0$ (over-smoothed)

Figure 8: Individual segmentation results with different smoothness weights

Figure 9: Final segmentation result with $\lambda = 10$ overlaid on original image

Analysis and Limitations

Effect of Smoothness Weight λ:

Low λ (e.g., 1.0): Segmentation becomes noisy and sensitive to color variations. Small regions are frequently mislabeled, resulting in fragmented boundaries. The data term dominates, causing over-segmentation based solely on color similarity.

Medium λ (e.g., 10.0): Provides balanced segmentation with smooth boundaries. The smoothness term regularizes the solution while respecting color differences, yielding coherent object regions.

High λ (e.g., 50.0): Over-smooths the segmentation, potentially merging distinct objects. Boundaries become overly regularized, ignoring fine details and color transitions. The smoothness term dominates, leading to under-segmentation.

Limitations:

1. Simple Color Model: Single Gaussian models fail to capture multimodal color distributions. Objects with varying textures or colors (e.g., patterned clothing) cannot be represented accurately, causing partial segmentation failures.

2. Texture Ignorance: The method uses only color information, ignoring texture patterns. Textured regions with similar average colors to the background get misclassified, as spatial frequency information is not captured.

3. Seed Dependency: Segmentation quality heavily depends on seed placement and quantity. Insufficient or poorly placed seeds lead to inaccurate Gaussian models, propagating errors throughout the segmentation. Interactive refinement is often necessary.

Code Repository

All implementation code is available as Python scripts:

- `assignment.py`

Google colab link:
[https://colab.research.google.com/drive/1D8nsR9yxVe5CVXeNUqSXsABeGxR3PszW?usp=sharing]