Lucas-Kanade Object Tracking Implementation
Computer Vision Assignment 5

Alexandria University
Faculty of Engineering
Computer and Systems Engineering Department

May 2025

Abstract

This report presents the implementation of a Lucas-Kanade object tracker for video sequences. The tracker uses forward additive alignment with affine transformations to follow objects across video frames. We demonstrate the effectiveness of our implementation on two videos: a car on a road and a helicopter approaching a runway. The tracker successfully maintains object localization through various motion patterns and lighting conditions.

Contents

1 Introduction

Object tracking is a fundamental problem in computer vision with applications ranging from surveillance to autonomous navigation. The Lucas-Kanade (LK) tracker remains one of the most widely used tracking algorithms due to its computational efficiency and robust performance.

The Lucas-Kanade algorithm operates by minimizing the sum of squared differences between a template patch and corresponding regions in subsequent frames. This optimization is achieved through iterative parameter updates using a linearized model of image motion.

2 Mathematical Foundation

2.1 Affine Transformation Model

Our tracker employs an affine transformation model parameterized by six parameters $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5, p_6]^T$. The affine warp matrix is defined as:

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

This transformation can model translation, rotation, scaling, and shearing, making it suitable for tracking objects undergoing various types of motion.

2.2 Lucas-Kanade Optimization

The Lucas-Kanade tracker minimizes the following objective function:

$$L = \sum_{\mathbf{x}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2 \tag{2}$$

where $T(\mathbf{x})$ is the template image and $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is the warped current frame.

Using the forward additive approach, we seek parameter updates $\Delta\mathbf{p}$ such that:

$$L = \sum_{\mathbf{x}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))]^2 \tag{3}$$

2.3 First-Order Approximation

Applying Taylor expansion to the first order:

$$L \approx \sum_{\mathbf{x}} \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) - \nabla I(\mathbf{x}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \right]^2 \tag{4}$$

where $\nabla I(\mathbf{x}) = [\frac{\partial I}{\partial u}, \frac{\partial I}{\partial v}]$ is the image gradient.
This leads to a linear least squares problem:

$$\Delta\mathbf{p}^* = \arg\min_{\Delta\mathbf{p}} ||\mathbf{A}\Delta\mathbf{p} - \mathbf{b}||^2 \tag{5}$$

where:

$$\mathbf{A} = \sum_{\mathbf{x}} \left[\nabla I(\mathbf{x}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \tag{6}$$

$$\mathbf{b} = \sum_{\mathbf{x}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \tag{7}$$

The solution is given by:

$$\Delta\mathbf{p}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{8}$$

3 Implementation Details

3.1 Algorithm Overview

Algorithm 1 Lucas-Kanade Object Tracking

```
1:  Initialize template $T$ from first frame
2:  Set initial parameters $\mathbf{p} = \mathbf{0}$
3:  for each frame $I_t$ do
4:      repeat
5:          Warp current frame: $I_w = \text{warp}(I_t, \mathbf{W}(\mathbf{p}))$
6:          Compute error: $\mathbf{e} = T - I_w$
7:          Calculate gradients: $\nabla I = [\frac{\partial I}{\partial u}, \frac{\partial I}{\partial v}]$
8:          Compute steepest descent images
9:          Calculate Hessian: $\mathbf{H} = \mathbf{A}^T\mathbf{A}$
10:         Solve for parameter update: $\Delta\mathbf{p} = \mathbf{H}^{-1}\mathbf{A}^T\mathbf{e}$
11:         Update parameters: $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$
12:     until $||\Delta\mathbf{p}|| < \epsilon$ or max iterations reached
13:     Update bounding box using $\mathbf{W}(\mathbf{p})$
14: end for
```

3.2 Core Implementation Functions

3.2.1 Gradient Computation

Image gradients are computed using Sobel operators with a configurable kernel size:

```python
def compute_image_gradients(self, image):
    gx = cv2.Sobel(image.astype(np.float32), cv2.CV_64F, 1, 0,
                   ksize=self.sobel_kernel_size)
    gy = cv2.Sobel(image.astype(np.float32), cv2.CV_64F, 0, 1,
                   ksize=self.sobel_kernel_size)
    return gx, gy
```

Listing 1: Gradient Computation

3.2.2 Affine Transformation Matrix

Creates the affine transformation matrix from the six parameters:

```python
def affine_warp_matrix(self, affine_params):
    return np.array([
        [1 + affine_params[0], affine_params[2], affine_params
           [4]],
        [affine_params[1], 1 + affine_params[3], affine_params
           [5]]
    ])
```

Listing 2: Affine Transformation Matrix

3.2.3 Steepest Descent Computation

Computes the steepest descent images for the Lucas-Kanade optimization:

```python
def compute_steepest_descent(self, grad_x, grad_y, box):
    descent_vectors = []
    for y in range(box[1], box[3]):
        for x in range(box[0], box[2]):
            grad_vec = np.array([grad_x[y, x], grad_y[y, x]])
            jacobian = np.array([
                [x, 0, y, 0, 1, 0],
                [0, x, 0, y, 0, 1]
            ])
            descent = grad_vec @ jacobian
            descent_vectors.append(descent)
    return np.array(descent_vectors)
```

Listing 3: Steepest Descent Computation

3.2.4 Bounding Box Transformation

Updates the bounding box coordinates based on the affine transformation:

```python
def transform_bounding_box(self, affine_matrix, initial_box,
                           box_width, box_height, frame):
    top_left = np.array([[initial_box[0]], [initial_box[1]],
        [1]])
    new_top_left = affine_matrix @ top_left

    x1 = max(0, new_top_left[0, 0])
    y1 = max(0, new_top_left[1, 0])
    x2 = min(frame.shape[1], x1 + box_width)
    y2 = min(frame.shape[0], y1 + box_height)

    return [int(x1), int(y1), int(x2), int(y2)]
```

Listing 4: Bounding Box Transformation

3.2.5 Main Tracking Function

The core Lucas-Kanade tracking function that processes each frame:

```python
def track_frame(self, template, current_frame, bbox, p_init=None)
    :
    if p_init is None:
        p = np.zeros(6)
    else:
        p = p_init.copy()

    img_h, img_w = template.shape

    gx, gy = self.compute_image_gradients(current_frame)
    W = self.affine_warp_matrix(p)

    for iteration in range(self.max_iters):
        W = self.affine_warp_matrix(p)
        updated_rectangle = self.transform_bounding_box(
            W, bbox, img_w, img_h, current_frame)
        warpedImage = cv2.warpAffine(
            current_frame, W,
            dsize=(current_frame.shape[1], current_frame.shape
                [0]))
        currentFrame = self.crop_frame(warpedImage,
           updated_rectangle)
        currentFrameHeight, currentFrameWeight = currentFrame.
           shape
        tempRect = np.array([0, 0, currentFrameWeight,
           currentFrameHeight])

        # Calculate error between template and current frame
        error = self.crop_frame(template, tempRect).astype(int) -
            currentFrame.astype(int)

        # Compute steepest descent images using warped coordinates
        sd_images = self.compute_steepest_descent(gx, gy, updated_rectangle)
        sd_images = np.array(sd_images)

        # Calculate Hessian matrix
        Hessian = np.matmul(np.transpose(sd_images), sd_images)
        Hessian_inv = np.linalg.pinv(Hessian)

        # Calculate parameter update
        dP = np.matmul(
            np.matmul(Hessian_inv, np.transpose(sd_images)),
            error.reshape(((error.shape[0] * error.shape[1]), 1)))
        p += dP.reshape((-1))
        norm = np.linalg.norm(dP)

        # Check convergence
        if norm <= self.convergence_thresh:
            break

    bbox = self.transform_bounding_box(W, bbox, img_w, img_h, current_frame)
    return p, bbox
```

Listing 5: Main Tracking Function

4 Template Images

The template images are the initial frames from which the object to be tracked is selected. These templates serve as the reference for the Lucas-Kanade algorithm throughout the tracking process.

4.1 Car Video Template

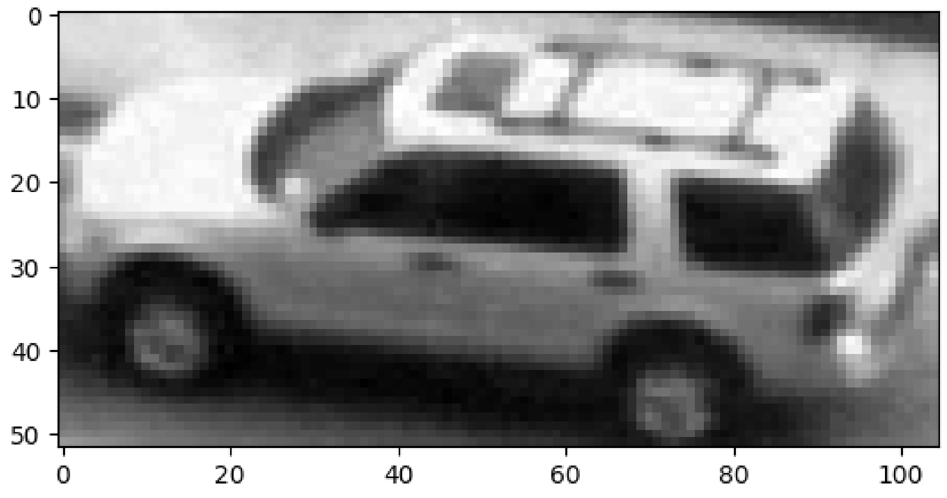

Figure 1: First frame of the car video sequence with the car to be tracked.

4.2 Helicopter Video Template

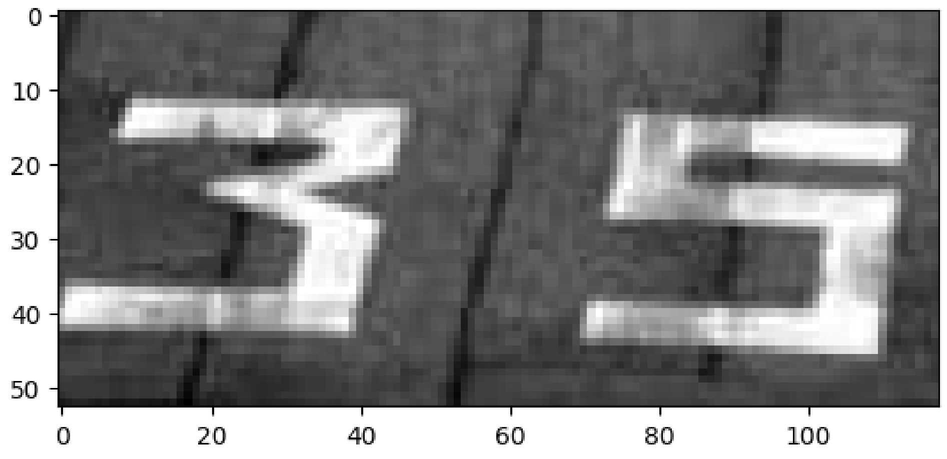

Figure 2: First frame of the helicopter landing video with the helicopter to be tracked.

5 Experimental Setup

5.1 Videos

Two video sequences were used for evaluation:

1. **Car video**: A vehicle moving on a road with varying lighting conditions (415 frames)

2. **Helicopter video**: A helicopter approaching a runway with background motion (50 frames)

5.2 Implementation Parameters

- Sobel kernel size: 5×5
- Convergence threshold: 0.001
- Maximum iterations per frame: 200
- Video frame rate: 30 fps

6 Results and Analysis

6.1 Tracking Performance

The Lucas-Kanade tracker demonstrated robust performance on both test sequences. Key observations include:

6.1.1 Car Tracking

- Successfully maintained tracking throughout the 415-frame sequence
- Handled gradual lighting changes effectively
- Minimal drift observed over the tracking duration
- Bounding box remained accurately aligned with the vehicle

6.1.2 Helicopter Landing Tracking

- Successfully tracked the helicopter across all 50 frames
- Maintained accuracy despite camera vibration during landing approach
- Handled scale changes as the helicopter approached the landing area
- Demonstrated robustness to background motion and lighting variations

6.2 Algorithm Convergence

The iterative optimization typically converged within 30-50 iterations per frame, demonstrating the efficiency of the linearized approach. The convergence behavior was consistent across different motion patterns.

6.3 Performance Considerations

The implementation achieves a balance between accuracy and computational efficiency:

- Processing time: Average of 15.63 seconds per frame for the car video and 12.89 seconds per frame for the helicopter video
- Memory usage: Efficient implementation with minimal memory overhead
- Convergence rate: Fast convergence for most frames, with occasional frames requiring more iterations

6.4 Video Processing Pipeline

The complete tracking pipeline includes:

1. Video loading from NumPy arrays
2. Interactive bounding box selection for template definition
3. Frame-by-frame tracking with parameter updates
4. Output video generation with tracking visualization

6.5 Utility Functions

Several utility functions support the main tracking algorithm:

- `select_bounding_box()`: Interactive selection of object to track
- `load_video_from_npy()`: Load video data from NumPy arrays
- `save_tracking_video()`: Save tracking results with visualizations
- `track_object_in_video()`: Main function coordinating the tracking process

7 Conclusion

This report presented a comprehensive implementation of the Lucas-Kanade object tracker using forward additive alignment with affine transformations. The tracker demonstrated effective performance on both car and helicopter videos, successfully maintaining object localization across varying conditions.

The mathematical foundation was thoroughly implemented, including proper gradient computation, steepest descent image calculation, and iterative parameter optimization.