CS 5330: Pattern Recognition and Computer Vision

Northeastern University

Lab 10: Camera

* *Contributed by Fall 2024 TAs: Byunghyun Ko, Yihan Wang and Taiwei Cui*

Contours

1. Setting up Camera in OpenCV
2. Introduction to Optical Flow
3. Motion Detection using Optical Flow
4. Feature Tracking using Optical Flow
5. Code Example

Setting up Camera in OpenCV

- Cv2.VideoCapture(0)
  - 0 is used for default camera
  - 1 is used for external camera, if available
- Reading and Displaying Frames
  - After accessing the camera, we can read frames in a loop and display them
  - Common functions that are utilized:
    - cap.read() -> caputres each frame the camera
    - cv2.imshow("Camera Feed", frame) -> displays the current frame
    - Note: att the end, you need to **release** the camera using cap.release()

Introduction to Optical Flow

- Optical Flow: tracks the apparent motion of pixels between consecutive frames.
  - Detects changes over time, making it useful for tracking objects or identifying movements of an object
- In this lab, we will learn how we can implement optical flow for:
  - Motion detection
  - Feature tracking

https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html

Motion Detection using Optical Flow

- There are two methods when using optical flow to detect motions
  - Lucas-Kanade Optical Flow
    - Commonly used method for sparse optical flow
    - Tracks specified feature points across frames
    - Strengths: computationally efficient, accurate for tracking feature points, less sensitive to small motions, scalable
    - Weaknesses: not suitable for large motions, limited to specific points
  - Dense Optical Flow
    - Calculates optical flow for all points in the frame, which can be computationally intensive
    - Strengths: detailed motion information, good for large motions, useful for scene-wide analysis
    - Weaknesses: computationally intensive, slower processing, sensitive to noise

Motion Detection using Optical Flow

(a) Sparse Optical Flow – Lukas Kanade

(b) Dense Optical Flow - Gunnar Farneback

https://www.researchgate.net/figure/Sparse-Optical-Flow-vs-Dense-Optical-Flow-a-Sparse-Optical-Flow-Lukas-Kanade-b_fig2_361803148

Motion Detection using Optical Flow

- Lucas-Kanade Optical Flow (Sparse)
  - cv2.calcOpticalFlowPyrLK(prev_frame, next_frame, prev_points, None, **params)
    - **prev_frame**: The first frame (grayscale) in which the initial points are located. It serves as the starting point for tracking.
    - **next_frame**: The subsequent frame (also in grayscale) where the algorithm will search for the new position of the points.
    - **prev_points**: The points in prev_frame that you want to track in next_frame. Typically detected using cv2.goodFeaturesToTrack.
    - **None**: Placeholder for output array, which we set to None.
    - **params**: A dictionary containing parameters for the Lucas-Kanade method.

Motion Detection using Optical Flow

- Dense Optical Flow
  - cv2.calcOpticalFlowFarneback(prev_frame, next_frame, None, pyr_scale, levels, winsize, iterations, poly_n, poly_sigma, flags)
    - **prev_frame**: The first frame (grayscale) in which motion is calculated.
    - **next_frame**: The subsequent frame (also grayscale) where the optical flow is calculated.
    - **None**: Placeholder for the output array, which we set to None in typical usage.
    - **pyr_scale**: Parameter specifying the image scale between pyramid levels (0 to 1). For example, 0.5 scales the image to half its size at each level. A smaller scale increases accuracy but requires more computation.
    - **levels**: Number of pyramid levels used in the calculation. More levels improve the algorithm's ability to track large motions but increase computation.
    - **winsize**: Size of the window used for averaging flow. A larger window smooths the flow but may lose finer details.
    - **iterations**: Number of iterations the algorithm does at each pyramid level to refine the flow.
    - **poly_n**: Size of the pixel neighborhood used to find polynomial expansion in each pixel. Larger values are more robust to noise but require more computation. Common values are 5 or 7.
    - **poly_sigma**: Standard deviation of the Gaussian used for neighborhood weighting. A larger value results in smoother flow but reduces sensitivity to smaller motion.
    - **flags**: Operation flags for modifying the algorithm behavior.

Feature Tracking using Optical Flow

- **Shi-Tomasi Corner Detector:** can help us identify corners and features in the frame that are stable and easy to track
  - cv2.goodFeaturesToTrack(gray_frame, maxCorners, qualityLevel, minDistance)gray_frame: The grayscale frame for feature detection.
    - **maxCorners**: Maximum number of corners to return.
    - **qualityLevel**: Minimum quality of corners to consider.
    - **minDistance**: Minimum distance between detected corners.
  - We then pass the detected features to cv2.calcOpticalFlowPyrLk to track them across consecutive frames

Code Example

Code

```
# Set up camera
cap = cv2.VideoCapture(0)

# Parameters for Shi-Tomasi Corner Detection to find points for Lucas-Kanade
feature_params = dict(maxCorners=100, qualityLevel=0.3, minDistance=7, blockSize=7)

# Parameters for Lucas-Kanade Optical Flow
lk_params = dict(winSize=(15, 15), maxLevel=2, criteria=(cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Read the first frame
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)

# Detect initial points to track using Shi-Tomasi Corner Detection
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)
```

Shi-Tomasi Corner Detection:
•Detects good features to track in the first frame, which will be used as points for the **Lucas-Kanade** sparse optical flow method.

Code

```
# Create a mask for drawing Lucas-Kanade optical flow tracks
mask = np.zeros_like(old_frame)


while True:
    # Capture a new frame
    ret, frame = cap.read()
    if not ret:
        break
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # ---- Lucas-Kanade Optical Flow (Sparse) ----
    # Calculate optical flow using Lucas-Kanade for sparse feature points
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points (those successfully tracked)
    if p1 is not None:
        good_new = p1[st == 1]
        good_old = p0[st == 1]
```

Lucas-Kanade Optical Flow (Sparse):
•Calculates the optical flow for the detected points across frames using **cv2.calcOpticalFlowPyrLK**.
•Draws the motion paths for each point, with lines and circles representing the tracks on the frame.

Code

```
# Draw the tracks for Lucas-Kanade Optical Flow
for i, (new, old) in enumerate(zip(good_new, good_old)):
    a, b = new.ravel()
    c, d = old.ravel()
    mask = cv2.line(mask, (a, b), (c, d), (0, 255, 0), 2)
    frame = cv2.circle(frame, (a, b), 5, (0, 0, 255), -1)

# Overlay the Lucas-Kanade tracks on the original frame
lk_output = cv2.add(frame, mask)

# ---- Dense Optical Flow ----
# Calculate dense optical flow using Farneback method
flow = cv2.calcOpticalFlowFarneback(old_gray, frame_gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)

# Convert flow to HSV format for visualization
hsv = np.zeros_like(frame)
hsv[..., 1] = 255

# Convert flow to polar coordinates to get the magnitude and angle
mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
hsv[..., 0] = ang * 180 / np.pi / 2  # Hue represents direction
hsv[..., 2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)  # Value represents magnitude
```

Dense Optical Flow

- Calculates the dense optical flow using **cv2.calcOpticalFlowFarneback** for the entire frame, resulting in motion vectors for each pixel.
- Converts the dense flow to an HSV image for visualization:
  - **Hue** represents the flow direction.
  - **Value** represents the flow magnitude (speed of motion).
- The HSV image is then converted to BGR format for display as a color map, showing the direction and speed of motion across the whole frame.

Code

```
    # Convert HSV to BGR for visualization
    dense_output = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

    # Display the results
    cv2.imshow("Lucas-Kanade Optical Flow (Sparse)", lk_output)
    cv2.imshow("Dense Optical Flow", dense_output)

    # Update the previous frame and points for the next loop iteration
    old_gray = frame_gray.copy()
    if good_new is not None:
        p0 = good_new.reshape(-1, 1, 2)

    # Break the loop on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the camera and close all windows
cap.release()
cv2.destroyAllWindows()
```

Displaying Results:
•The output from Lucas-Kanade is displayed in one window, while the dense optical flow (color-coded map) is shown in another window.

Loop and Update:
•Updates the previous frame (old_gray) and points (p0) for the next iteration.
•The loop runs continuously until the user presses **'q'** to quit.