HW3 Motion Estimation & Compensation

109550119 邵筱庭

1. 載入圖片為 uint8 並初始化數值

```python
# Load images
one_gray = cv2.imread('one_gray.png', cv2.IMREAD_GRAYSCALE).astype(np.uint8)
two_gray = cv2.imread('two_gray.png', cv2.IMREAD_GRAYSCALE).astype(np.uint8)

# Specify block size and search ranges
block_size = 8
search_ranges = [8, 16, 32]
search_methods = ['FS', 'TSS']
```

2. 主架構
   - 使用 motionEstimation()使用選定的搜索方法和搜索範圍計算 motion vectors
   - 利用 motion vectors 和 motionCompensation()進行運動補償，產生出 compensated_frame
   - 計算 residual
   - 使用 cv2.PSNR()計算

```python
# Perform motion estimation and compensation for each search range and method
for search_method in search_methods:
    for search_range in search_ranges:
        # Start the timer
        start_time = time.time()

        # Perform motion estimation using the chosen search method
        motion_vectors = motionEstimation(one_gray, two_gray, block_size, search_range, search_method)

        # Stop the timer and calculate the runtime
        end_time = time.time()
        runtime = end_time - start_time

        # Perform motion compensation
        compensated_frame = motionCompensation(two_gray, motion_vectors, block_size)

        # Calculate the residual
        residual = two_gray.astype(int) - compensated_frame.astype(int)
        residual = np.clip(residual, 0, 255).astype(np.uint8) # Avoid overflow

        # Save the reconstructed frame and the residual
        cv2.imwrite(f'reconstructed_{search_method}_{search_range}.png', compensated_frame)
        cv2.imwrite(f'residual_{search_method}_{search_range}.png', residual)

        # Calculate and print the PSNR
        psnr = cv2.PSNR(two_gray.astype(np.float32), compensated_frame.astype(np.float32))

        print(f'Search method: {search_method}-{search_range}, PSNR: {psnr}, Runtime: {runtime} seconds')
```

3. motionEstimation()
   - 首先取得錨定幀的高度和寬度
   - 創建空的 motion vectors
   - 根據 block size 遍歷影像的每一個 block，此外在邊界需要調整區塊大小
   - 根據選定的搜索方式獲得最佳匹配的座標，並存入 motion vectors

```python
def motionEstimation(anchor_frame, target_frame, block_size, search_range, search_method):
    """
    Motion Estimation using either Three-Step Search or Full Search with a single search range
    """
    h, w = anchor_frame.shape
    motion_vectors = np.zeros((h // block_size, w // block_size, 2), dtype=int)

    for y in range(0, h, block_size):
        for x in range(0, w, block_size):
            # Adjust the block size for boundary blocks
            actual_block_size = min(block_size, h - y, w - x)

            # Extract the anchor block
            anchor_block = anchor_frame[y:y + actual_block_size, x:x + actual_block_size]

            # Perform the chosen search method
            if search_method == 'TSS':
                best_match_coords = threeStepSearch(anchor_block, target_frame, search_range, (x, y))
            elif search_method == 'FS':
                best_match_coords = fullSearch(anchor_block, target_frame, search_range, (x, y))

            # Set motion vector for the block
            motion_vectors[y // block_size, x // block_size, :] = np.array(best_match_coords) - np.array([x, y])

    return motion_vectors
```

4. motionCompensation()
   - 首先取得錨定幀的高度和寬度
   - 創建一個與錨定幀相同大小的全零影像，用於存儲補償後的影像
   - 設定 padding，對錨定幀進行填充以處理靠近邊界的運動補償
   - 根據 block size 遍歷影像的每一個 block
   - 獲取 block 內部的 motion vector，並計算出運動補償後的 block 位置
   - 最後將這些 blocks 組合成補償後的影像

```python
def motionCompensation(anchor_frame, motion_vectors, block_size):
    h, w = anchor_frame.shape
    compensated_frame = np.zeros_like(anchor_frame)

    # Pad the anchor frame to handle motion compensation near the borders
    pad_size = block_size // 2  # Adjust as needed
    anchor_frame_padded = np.pad(anchor_frame, pad_size, mode='constant', constant_values=0)

    for y in range(0, h, block_size):
        for x in range(0, w, block_size):
            actual_block_size = min(block_size, h - y, w - x)

            dy, dx = motion_vectors[y // block_size, x // block_size]

            # Calculate the actual position of the block after motion compensation
            compensated_y, compensated_x = y + dy, x + dx

            # Ensure compensated_y and compensated_x are within valid range
            compensated_y = max(0, min(compensated_y, h - actual_block_size))
            compensated_x = max(0, min(compensated_x, w - actual_block_size))

            # Extract the compensated block from the padded anchor frame
            compensated_block = anchor_frame_padded[compensated_y:compensated_y + actual_block_size + 2 * pad_size, compensated_x:compensated_x + actual_block_size + 2 * pad_size]

            # Adjust the slicing to match the dimensions of the block being assigned
            compensated_frame[y:y + actual_block_size, x:x + actual_block_size] = compensated_block[pad_size:pad_size + actual_block_size, pad_size:pad_size + actual_block_size]

    return compensated_frame
```

5. fullSearch
   - 首先取得目標幀的高度和寬度並獲取錨定區塊的尺寸
   - 使用錨定位置初始化最佳匹配坐標
   - **遍歷整個搜索範圍中的搜索點**
   - 計算錨定區塊和搜索區塊之間的 MSE，若 MSE 較小澤更新最佳匹配坐標
   - 最後回傳最佳的匹配座標

```python
# Implementation of Full Search for Motion Estimation
def fullSearch(anchor_block, target_frame, search_range, anchor_position):
    h, w = target_frame.shape
    anchor_block_size = anchor_block.shape[0]

    # Initialize best match coordinates with the anchor position
    best_match_coords = anchor_position
    min_mse = float('inf')

    # Iterate through search points in the entire search range
    for search_y in range(max(0, anchor_position[1] - search_range), min(h - anchor_block_size, anchor_position[1] + search_range)):
        for search_x in range(max(0, anchor_position[0] - search_range), min(w - anchor_block_size, anchor_position[0] + search_range)):
            # Extract the search block from the target frame
            search_block = target_frame[search_y:search_y + anchor_block_size, search_x:search_x + anchor_block_size]

            # Calculate the Mean Squared Error (MSE) between anchor block and search block
            mse = np.sum((anchor_block - search_block) ** 2) / anchor_block.size

            # Update best match coordinates if MSE is smaller
            if mse < min_mse:
                min_mse = mse
                best_match_coords = (search_x, search_y)

    return best_match_coords
```

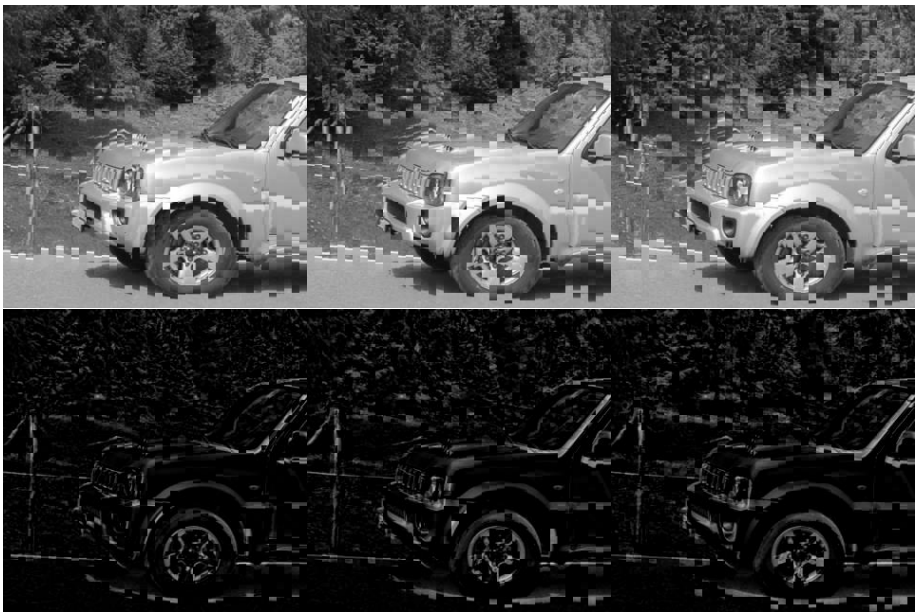
6. threeStepSearch()
   - 首先取得目標幀的高度和寬度並獲取錨定區塊的尺寸
   - 使用錨定位置初始化最佳匹配坐標
   - 定義 search steps (每次的 search 砍半)
   - **三個不同的搜索步驟下進行搜索**
   - **在最佳匹配坐標周圍的搜索範圍內遍歷搜索點**
   - 計算錨定區塊和搜索區塊之間的 MSE，若 MSE 較小澤更新最佳匹配坐標
   - 最後回傳最佳的匹配座標

```python
# Implementation of Three-Step Search for Motion Estimation
def threeStepSearch(anchor_block, target_frame, search_range, anchor_position):
    h, w = target_frame.shape
    anchor_block_size = anchor_block.shape[0]

    # Initialize best match coordinates with the anchor position
    best_match_coords = anchor_position

    # Define the search steps
    search_steps = [search_range // 2, search_range // 4 , search_range // 8]

    # Three steps
    for step in search_steps:
        min_mse = float('inf')
        x, y = best_match_coords

        # Iterate through search points in the search range around the best match coordinates
        for dx in range(max(0, x - step), min(w - anchor_block_size, x + step) + 1):
            for dy in range(max(0, y - step), min(h - anchor_block_size, y + step) + 1):
                search_x, search_y = dx, dy

                # Check if the search block is inside the target frame
                if search_x >= 0 and search_y >= 0 and search_x + anchor_block_size <= w and search_y + anchor_block_size <= h:

                    # Extract the search block from the target frame
                    search_block = target_frame[search_y:search_y + anchor_block_size, search_x:search_x + anchor_block_size]

                    # Calculate the Mean Squared Error (MSE) between anchor block and search block
                    mse = np.sum((anchor_block - search_block) ** 2) / anchor_block.size

                    # Update best match coordinates if MSE is smaller
                    if mse < min_mse:
                        min_mse = mse
                        best_match_coords = (search_x, search_y)

    return best_match_coords
```

7. 最終結果

- Full Search 之結果 (8/16/32)

- Three Step Search 之結果 (8/16/32)



- PSNR 與 Runtime 比較

```
Search method: FS-8, PSNR: 16.02268143250487, Runtime: 2.947319984436035 seconds
Search method: FS-16, PSNR: 15.017588826773816, Runtime: 12.334311246871948 seconds
Search method: FS-32, PSNR: 14.20706542554861, Runtime: 63.95784854888916 seconds
Search method: TSS-8, PSNR: 17.734678628281515, Runtime: 1.786557912826538 seconds
Search method: TSS-16, PSNR: 15.802040759710112, Runtime: 6.600359678268433 seconds
Search method: TSS-32, PSNR: 14.93184401533355, Runtime: 54.35276651382446 seconds
```