Video Processing - Final Project

Project Structure

The project's code is found the following python files:

- main.py
- common_utils.py
- stabilization_utils.py
- bg_sub_utils.py
- matting_utils.py
- tracking_utils.py

Video stabilization

The algorithm we've designed stabilizes a video using the middle frame as an anchor point to which all the other frames are warped. The usage of the middle frame expresses an implicit assumption that this frame represents some average state of the overall movements throughout the video. Using this assumption helps minimizing the computational cost of later border corrections, caused by the frames' warping and required for better background subtraction.

The stabilization is done by finding distinct features in each frame and matching them to distinct features in the anchor frame, using the SIFT algorithm. Subsequently, we use these features to estimate homographic transformation between the given frame and the anchor frame. In order to minimize the probability of pairing two different features, we use only the features the have high enough similarity to each other by this algorithm:

- Describe each distinct feature (spatial point in the frame) uniquely as 128-d vector ("descriptor"; standard representation used in SIFT algorithm).
- Find the 2 nearest neighbors of each feature in terms of L2 distance in descriptors' space.
- Consider match as "good" only if the nearest neighbor is found to be much closer to the feature than the second one. Mathematically speaking, we consider a match m_1 to be good if $d(m_1) < 0.5 * d(m_2)$ where:
    - d is the similarity measure between the feature in the anchor image and the examined feature in the given frame.
    - m_1, m_2 are the two matches with the best similarity to the feature in the anchor frame.

In practice, we perform 2 stabilization iterations. The first iteration roughly stabilizes by skipping frames and estimating the warp matrix of skipped frames by a weighted average of the last and next matrices that were calculated. This step is required for estimating still background for later border effects fix. The second iteration stabilizes the video again, this time frame-by-frame, while using e the estimated background image to fill the black margins (which will come in handy later in the background subtraction).

Thus, the stabilization algorithm can be described as follows:

1. **Extracting frames:** The function retrieves all frames from the input video, using the utils.get_frames function.
1. **Anchor frame selection:** The middle frame of the video is chosen as the anchor frame.
2. **Feature detection and description calculation in current frame:** The anchor frame is analyzed to identify good features using the Harris corners algorithm and non-maximum suppression. SIFT descriptors are calculated for the detected features in the anchor frame.

3. **Feature description:** For each frame in the input video -
    1. **Feature detection and description calculation in current frame:** The current frame is processed to detect good features using the same method as the anchor frame. SIFT descriptors are computed for the detected features in the current frame.
    2. **Feature matching:** The SIFT descriptors of the anchor and current frames are matched to find corresponding feature points.
    3. **Filter matched features:** The matched feature points are filtered based on a distance ratio test to retain only the most reliable matches.
    4. **Homography estimation**: a homographic transformation is estimated between the anchor frame and the current frame using the matched feature points.
        1. in case of frame skip, the skipped frames given with the result of a weighted average between the previous homography and the current one:

$$\hat{M}_j = (1 - w_j) * M_i + w_j * M_{i+s}$$

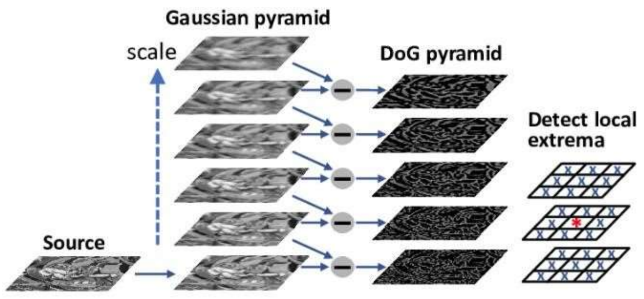
$$w_j = \frac{|j - (i - s)|}{s}$$

Where s is the frames' skip (we used $s = 3$), i (and $i + s$) is the index of the previous (and next) computed homography transform, and $j \in (i, i + s)$ is the index of the frame within the skipped range.

    5. **Mask channel addition**: A constant channel is added to the current frame to create a mask for the edges. This will be helpful for filling the black margins and bringing a nicer result.
    6. **Warping**: The current frame is stabilized by warping it according to the computed homography.
    7. **Fill Black Margins**: the stabilized frame and mask are sent to **fill_black_margins**, which fills the black margins of the warped image with meaningful content by taking the background estimation values. Together with cropping 2.5% of the margins, this helps to maintain the visual consistency and coherence of the stabilized image.
    8. **Writing the stabilized frame**: The stabilized frame is written to the output video.
4. The process continues until all frames have been processed.

SIFT Algorithm - Explanation

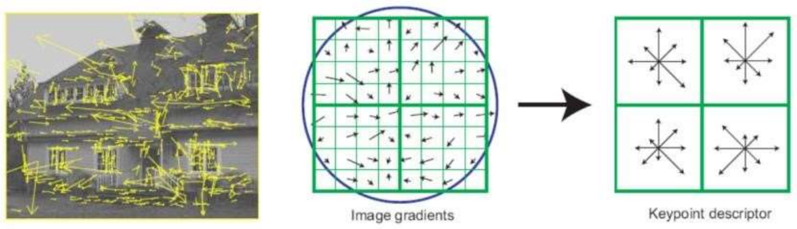
The features matching was done by using the SIFT algorithm. SIFT detects and describes distinctive local image features called keypoints, that are identified based on their scale-space extrema in the Difference of Gaussians (DoG) pyramid.



Here's a brief explanation of how it works:

1. The original image is convolved with a series of Gaussian filters at different scales to create a scale-space pyramid. Each level of the pyramid represents the image at a different scale. 2. Adjacent levels of the scale-space pyramid are subtracted to obtain the Difference of Gaussians (DoG) pyramid. This helps to enhance the regions with significant intensity changes at different scales.
3. The DoG pyramid is analyzed to identify keypoints. A keypoint is typically a local maximum or minimum in the DoG pyramid compared to its neighboring pixels in scale and space. These keypoints represent stable image features that are invariant to scale changes.
4. Once the keypoints are identified, precise locations and scales are determined by fitting a quadratic function to the nearby samples in the DoG pyramid. This localization process helps to accurately locate the keypoints within the image.

The algorithm computes and encodes a descriptor for each keypoint by considering the local image gradient orientations in the vicinity of each keypoint.



This descriptor is invariant to scale and rotation, making it effective for matching keypoints across different images. The matching process involves comparing the descriptors of keypoints in different images using distance metrics like Euclidean distance. The keypoints with the best matches are considered reliable correspondences.

Even though it's slower than LK, we've preferred SIFT over LK since it performed significantly better, and the time cost was bearable.

Elaboration of Key CV2 functions

1. SIFT
    - **cv2.SIFT_create()**:creates an instance of the SIFT feature extractor.
    - **sift.detectAndCompute()**: combines the detection of keypoints and the computation of descriptors.
    - **cv2.BFMatcher()**: initializes a brute-force matcher for feature matching.
    - **matcher.knnMatch()**:performs a k-nearest neighbors match between the SIFT descriptors of two images. We set the parameter **k** to 2 to obtain the two best matches for each descriptor.
2. Warp
    - **cv2.findHomography()**: computes the homography matrix between two sets of corresponding points. It estimates the geometric transformation between the reference image and the current image.
    - **cv2.warpPerspective()**: used to apply a perspective transformation to an image. It warps the image according to a given homography matrix, which represents the geometric transformation between two sets of corresponding points.

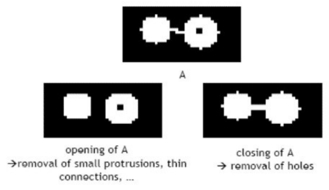
Background Subtraction

We used a K-Nearest Neighbors (KNN) background subtractor model using OpenCV implementation cv2.createBackgroundSubtractorKNN.

KNN background subtraction is a pixel-based algorithm that classifies each pixel as either foreground or background based on its similarity to the previously observed background pixels. It considers the K nearest neighbors in the feature space to determine the label of the current pixel.

The subtraction algorithm can be described as follows:

1. A K-Nearest Neighbors (KNN) background subtractor model (**cv2.createBackgroundSubtractorKNN**) is initialized.
2. The pre-learning phase begins, where the background subtractor model is applied iteratively to the video frames in mixed order to learn and adapt to the background.
    1. For enhancing the training and, the training was back-and-forth, meaning that we introduced the video to the model frame by frame from the beginning of the video to its end, and then again from the end to the beginning. Since KNN implementation for background subtraction classifies each pixel by its history, this approach improved the performance of the KNN trained model at the beginning of the video, due to the fact the its last state has strong correlation with the beginning of the video but also memory of the rest of the video.
3. The code enters a loop to process each frame in the video:
    1. The background subtraction is applied to the current frame using the background subtractor model, resulting in a foreground mask (**fg_mask**) that highlights the moving objects in the frame.
    2. Post-processing steps are performed on the foreground mask to denoise and enhance the results. This includes applying a median filter (**cv2.medianBlur**) and performing morphological operations (opening and closing) to reduce noise using an elliptical structuring element (**cv2.morphologyEx**) for noise reduction and filling holes in the foreground mask.



    3. A threshold operation (**cv2.threshold**) is applied to convert the foreground mask into a binary mask.
    4. The desired content is extracted from the original frame by performing a bitwise AND operation (**cv2.bitwise_and**) between the original frame and the binary mask.
    5. The binary mask and the extracted frame are written to the respective output videos.

The process continues until all frames in the video have been processed.

Elaboration of Key CV2 functions

- cv2.createBackgroundSubtractorKNN ○ The **subtractor.apply()** method is used to apply the background subtractor to each frame and obtain the foreground mask.

- **cv2.morphologyEx** - performs various morphological operations, such as erosion, dilation, opening, closing, etc., on a given image. These operations are commonly used for tasks like noise removal.

Matting

Creating the matted video involves the following steps:

1. **create_alpha**: This function processes the stabilized and binary video frames to create an alpha video. It reads the frames sequentially and calls **create_alpha_frame** for each frame. The resulting alpha frames are written to an output video file. For each frame:
    1. **create_alpha_frame**: It calls **extract_certainty_regions** to obtain the certainty masks and **estimate_conditional_distributions** to estimate the conditional distributions. Using the probabilities from the distributions, it creates an alpha map that assigns values of 1 to the foreground, 0 to the background, and interpolated values for the uncertainty region.
    2. **extract_certainty_regions**: takes a binary frame as input and performs morphological operations (erosion and dilation) to extract the foreground (erosion), background(dilation), and uncertainty regions (dilation-erosion).

    3. **estimate_conditional_distributions**: takes the stabilized frame and the certainty masks obtained from the previous step. It applies bitwise operations to separate the foreground, background, and uncertainty regions from the stabilized frame. It then performs morphological operations on the certainty masks to extract internal edges of the foreground and background. The function samples foreground and background pixels from the internal edges and estimates conditional distributions (using KDE) for the uncertainty region.

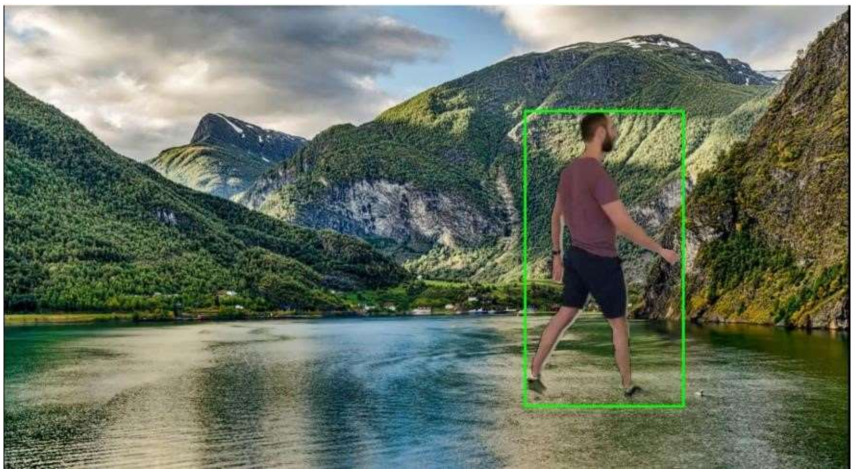
2. **create_matted**: takes the extracted video and an alpha video as inputs. It combines the alpha frames with the extracted frames and a new background image to create a matted video. The alpha frames are used to blend the foreground and background elements, producing the final matted frames. The resulting frames are written to an output video file.

Elaboration of key functions

- **scipy.stats.gaussian_kde()** - used to estimate the probability density function (PDF) of a set of observations using kernel density estimation (KDE) with a Gaussian kernel.
    - kde.evaluate(points)- evaluates the estimated PDF at specific points (the uncertainty mask in our case).

Tracking

Object tracking using the binary video. It assumes that the binary frames provided accurately represent the location of the object in the video. For each frame, it calls the **find_tracking_window** function, passing the corresponding binary frame from **binary_frames**. This function finds the contours in the image using **cv2.findContours**, and selects the largest contour as the object, this will be our person in very high probability. It then calculates the bounding rectangle of the contour. Then it extracts the coordinates and dimensions of the bounding rectangle (x, y, w, h). finally, it draws the bounding box on the matted frame using **cv2.rectangle**, specifying the starting and ending coordinates of the rectangle, the color (0, 255, 0) for green, and a line thickness of 2. This visualizes the tracking bounding box on the frame.



Elaboration of key CV2 functions

- **cv2.findContours ()** - used to detect contours in binary images. Contours are simply the boundaries of objects or shapes within an image.