**Final Project – Video Processing**

**Stabilization:**

The objective of the stabilization module is to enhance video stability, enabling successful execution of the following operations.

Our approach for video stabilization used feature matching.  
We iterate over the video frames and stabilize them with respect to the initial frame.

The stabilization process for each frame includes the following steps:

* **Extraction of keypoints and corresponding descriptors using SIFT**SIFT, or Scale Invariant Feature Transform, is an algorithm designed to identify salient key points and their corresponding descriptors in an image.  
  It has the property of rotational invariance, which is vital for stabilization purposes.  
    
  The workflow of SIFT involves the following stages:  
  + Calculating octaves: The image is resized to multiple scales (octaves), with each octave containing a resized version of the image that is half the size of the previous octave.
  + Blurring: Within each octave, the image is convolved with a Gaussian kernel having different variances. Consequently, each octave consists of images with increasing levels of blur.
  + DoG: The DoG (Difference of Gaussians) is calculated by computing the difference between neighboring octaves.
  + Getting potential keypoints: Each pixel in the DoG pyramid is compared with its eight neighbors, along with nine pixels in the subsequent scale and nine pixels in the previous scale. If a pixel stands out as an extremum, it is considered a potential keypoint.
  + Selection of descriptive keypoints: Potential keypoints with responses below a given threshold and those representing edges are discarded, leaving only the most descriptive keypoints.
  + Assigning orientations: An orientation is assigned to each keypoint to enhance rotation invariance. The direction with the most prominent gradient is chosen as the keypoint's orientation, and subsequent calculations are performed relative to this orientation.
  + Keypoint descriptor: For each keypoint, a 16x16 pixel window is taken and divided to 16 subwindows of 4x4 pixels. Within each subwindow, an 8-bin gradient histogram is computed to represent eight directions. The keypoint orientation is subtracted from each direction, and high values are saturated to handle illumination variations. These 16 histograms are concatenated to form a single 128-length feature vector, which serves as the keypoint descriptor.
* **Feature Matching Using a Brute Force Matcher**We employ a brute force matcher to compare descriptors between two frames. This matcher compares each descriptor in one image to all descriptors in the second image using L2 distance. As a result, every keypoint gets its best match from the other image. We used cv2.BFMatcher().match() for this step, omitting the crossCheck option due to stable results and timing considerations.
* **Finding an Optimal Homography with RANSAC**We used RANSAC (Random Sample Consensus) to find an optimal homography transformation that aligns the features.  
  Homography, offering more degrees of freedom than an affine transformation, produced better results for our scenario.  
  Our purpose was to find one homography for the entire frame, so we expected numerous outliers (for instance because of the person's movement).   
  In order to do that we used RANSAC, known for its robustness against outliers.  
  In our implementation, we used cv2.findHomography(...,method=cv2.RANSAC,...) for this step.
* **Warping With The Optimal Homography**  
  Finally, we stabilize the video by warping each frame using the homography obtained between the frame and the initial frame. We used cv2.warpPerspective() for this purpose in our implementation.

The code implementing the stabilization part can be found in the file video\_stabilization.py .

**Matting and Tracking:**

This block's goal is to perform matting and tracking.  
The matting is of the image foreground with a new background image using a calculated alpha map.  
The tracking includes detection of the moving object and plotting a bounding box around it.

The block contains implementation of the following steps:

* **Alpha Map Creation:**
  + We first create a trimap by subtracting morphological dilation from morphological erosion using 5x5 kernels. We then set the resulting area with a value of 0.5. We'll use the notation for this area.
  + We calculate geodesic distance maps from the foreground and background using the trimap's 0 and 1 values as initial foreground and background locations. We get two distance maps.
  + We calculate probability maps for the pixels to be in the foreground and background in the area , using gaussian KDE on the luma part of the image. As a result we get two probability maps: for foreground and background respectively. For this part we use Scipy's gaussian\_kde.
  + We combine the distance and probability maps to obtain alpha values for using the provided formula:

Where and are the foreground and background distance maps.

Note that this section is executed on a cropped area of the image to optimize computation time.

We extracted the bounding box using the trimap for effectiveness.

* **Matting:**

We perform matting of the foreground image (human walking) with a new background using the alpha map.

To do that, we follow these steps:

* + We set low and high confidences as "confident" foreground and background by thresholding the alpha map.  
    The threshold values we used are 0.9 and 0.1.
  + We iterate over all "non confident" pixels in the alpha map and create a 7x7 area of interest around each pixel.
  + We minimize the equation:  
     For all possible combinations of and in the window.  
     represents the alpha value for the current pixel, is the foreground value in the original image within the window, and is the background value in the original image within the window.

We Obtain the required foreground value for matting by finding the combination of foreground and background pixels that, along with the given alpha, produce the desired pixel value.

* + We create the final image by adding the background for areas classified as "confident" background and the foreground for areas classified as "confident" foreground, and the values we calculated for non-confident areas.
* **Detection:**As stated before, we extract the detection area from the previously generated trimap. We convert the bounding box parameters to center format.

All relevant code for this block can be found in the matting.py file.