# Video Processing and Analysis - Final Project

## Stabilization

In this section, we based our approach on calculating the two-dimensional affine transformation between each frame in the video and the first frame. The transformation matrix was computed by finding interest points using the Harris Corner Detector in the first frame and matching them in subsequent frames. Our assumption was that the video is not too long, and the scene does not change significantly, so interest points from the first frame remain relevant throughout the entire video.

### Finding interest points in the first frame only:

- Convert the color frame using the **cv2.cvtColor** function to grayscale.
- Create a 2D normalized kernel using **cv2.filter2D** to filter the grayscale frame.
- Apply the Harris Corner Detector algorithm to find interest points in the filtered frame using **cv2.cornerHarris**.
- Use Non-Maximal Suppression to filter the interest points.
- Select the top 100 interest points based on their values and create a list of these points for the entire video.

### Affine transformation for each frame:

- Convert frames from RGB to grayscale.
- For each interest point in the first frame, create a 45x45 Bounding Box, and around the corresponding point in the current frame, create a Bounding Box six times larger.
- Use cv2.matchTemplate to find the correlation between the two templates and estimate the affine transformation using cv2.estimateAffine2D with RANSAC for outlier rejection.
- Apply the obtained transformation to the current frame using cv2.warpAffine and add it to the stabilized video.

### Output Analysis:

The stabilized video shows significant improvement compared to the input video. Black borders appear at the edges of the stabilized video corresponding to the locations of interest points and the affine transformation matrix applied. During video stabilization, the algorithm adjusts the position and orientation of each frame to reduce fluctuations and shakes. This adjustment may cause shifts or rotations of frames, leading to visible edges in the stabilized frames. To address this, black pixels are added to fill and "pad" areas that exceed the original frame boundaries. These black areas serve as a visual cue for the implemented stability and help the stabilized video maintain the original aspect ratio as much as possible.

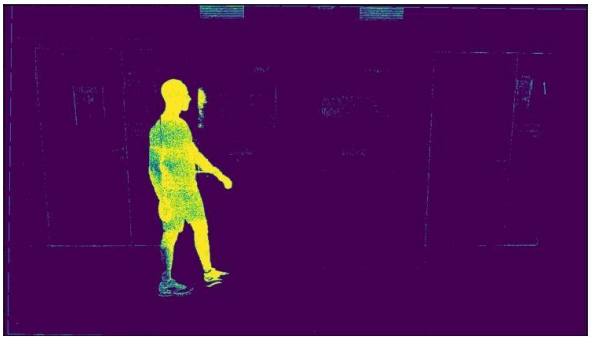
It is essential to note that the implemented solution is not perfect—small shakes are well stabilized, but its performance decreases in stabilizing strong vibrations. We hypothesize that stability issues may arise from an insufficient frame per second (fps) rate, potentially causing distortions in rapid and strong movements between consecutive frames. These distortions visible in the video may affect the alignment of interest points and the creation of a reliable affine matrix in those frames.

## Background Subtraction

In this section, we chose to use the built-in BackgroundSubtractorKNN class from cv2. This class implements background subtraction using the K-Nearest-Neighbor method. The algorithm builds statistical models of the background and foreground using the Gaussian Mixture Model (GMM) method (similar to the method taught in the lecture). Each new sample (per pixel) contributes to updating the models (adapting to changes in the scene) and is classified as either background or object based on the KNN classification using the Euclidean distance measure from the K nearest samples.

To obtain reasonable results for the initial frames as well, we ran the algorithm on all frames of the stabilized video five times consecutively. In the fifth pass, we saved the output of the algorithm (the object mask for each frame). Since stabilization is not perfect, certain pixels of the background were classified as objects in some frames. Therefore, we needed to filter them out. Additionally, we observed that the algorithm struggled in classifying pixels in the lower region of the object (presumably due to rapid pixel changes in the foot area caused by walking, compared to the slower changes in the upper part of the body).

.



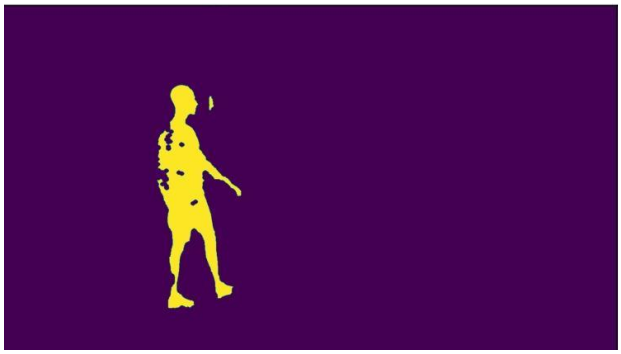
To improve the results, we applied a series of morphological operations, all implemented in the **cv2** library:

- **Closing Operation:**
    - Used a relatively small 3x3 kernel to fill small holes in the mask across all regions of the object.
    - Applied an elongated elliptical kernel, larger only for the bottom 40% of the frames, aiming to thicken the lower part of the object in the mask.
    - The purpose was to address any small gaps in the mask and enhance the representation of the legs in the object mask.

These morphological operations help refine the object mask and address specific challenges, such as small holes and variations in the lower part of the object. The assumption is that the same object is present in both videos, and this step is necessary to account for walking steps

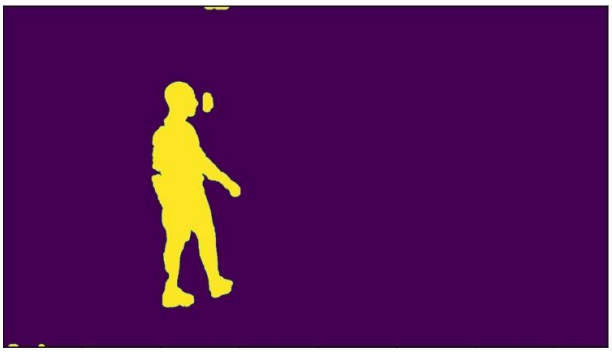
- **Erosion Operation:**
    - Applied an elliptical kernel of size 7x7 to filter out parts of the background that were misclassified as the object.

The erosion operation helps eliminate small regions in the background that might have been wrongly classified as part of the object. This step contributes to refining the accuracy of the object mask by removing false positives.



- **Dilation Operation:**
    - Used a slightly larger elliptical kernel of size 11x11 to "fill in" additional holes in the mask and thicken the object after the erosion operation.

  Dilation is applied to counteract the effects of erosion, ensuring that any small gaps created by the erosion operation are filled, and the object in the mask is thickened. This step helps maintain the overall shape and size of the object in the final mask.



At this stage, we leveraged the knowledge that the object is singular, allowing us to filter out any blobs in the mask that do not belong to the object, essentially excluding the largest blob. To achieve this, we utilized the **cv2.connectedComponentsWithStats** function from the **cv2** library. It takes a binary mask as input and assigns each pixel a specific label by defining 8-connectivity (a parameter we chose). According to this definition, a pixel is assigned a specific label only if one of its 8 neighbors (including diagonals) belongs to the same label.

The function returns a list of labels with associated statistics (such as label size, centroid coordinates, and bounding box, among others). Using this information, we removed all pixels not belonging to the largest label (excluding the background), assuming it represents the object.

Finally, we obtained a mask containing only the object in a well-defined manner, but it still included a "thread" of background pixels adjacent to the object. Therefore, we applied morphological operations again, specifically using erosion and closing, to refine the final output.

## Matting

In this part, we implemented the Matting algorithm as taught in the lecture. The algorithm we implemented includes several steps performed for each frame from the captured video and the binary video:

1) Preliminary Processing:

To avoid running computationally expensive functions (geodesic distance calculation, KDE editing) on RGB input, we converted the RGB frames to the HSV representation. We chose the channel that

yielded the best results after evaluating brightness (V) and saturation (S) channels, compared to the hue (H) channel. We used cv2.connectedComponentsWithStats on the binary frame to find the bounding box of the object. The bounding box was expanded to capture additional background around the object, and we cropped the processed frame accordingly. This decision was based on the observation that functions used later (KDE, geodesic distance calculation) are computationally expensive and unnecessary for the entire frame.

2) Creating Scribbles:

$$S_{FG} = Erode(Bin\_Frame), \quad S_{BG} = Erode(not(Bin\_Frame))$$

Scribbles for the object and background were defined using the binary frame. We generated a mask for FG (object) identical to the binary frame and a mask for BG (background) using the bitwise_not operation on the FG mask. Morphological erosion was applied to both masks to capture pixels with higher confidence of belonging to the object or background, excluding the region between them.

3) Probability Maps of Background and Object:

Gaussian KDE class from the SciPy. Stats package was employed to calculate probability maps of the background and object based on the cropped frame. Probability Density Functions (PDFs) were normalized using $PDF(BG) = \frac{P(BG)}{P(BG)+P(FG)}$ and PDF(FG) = $\frac{P(FG)}{P(BG)+P(FG)}$, where the argument is the gray level.[0 ,255]

4) Computation of Geodesic Distance Maps:

The GeodisTK package, specifically the geodesic2d_fast_marching function, was used to compute geodesic distance maps for each pixel to the provided scribbles. $V(F)$, a binary image, was defined as 1 for pixels where $D(FG) < D(BG)$ and 0 otherwise.

5) Creating the Narrow Band and Alpha Trimap:

The edges of the binary image $V(F)$ were found using cv2.Canny, followed by morphological dilation using cv2.dilate around the detected edges. This defined the narrow band where uncertainty exists about pixel belonging. The alpha_trimap image was created, having values of 1 in the definite object region, 0 in the definite background region, and 0.5 (arbitrary value) in the narrow band.

6) Calculation of the Opacity Map and Final Output:

The weights (W(FG), W(BG)) for alpha in the narrow band were calculated using $W_{FG} = \frac{{D_{FG}}^{-r} * P_{FG}}{{D_{FG}}^{-r} * P_{FG} + {D_{BG}}^{-r} * P_{BG}}$ and $W_{FG} = \frac{{D_{BG}}^{-r} * P_{BG}}{{D_{FG}}^{-r} * P_{FG} + {D_{BG}}^{-r} * P_{BG}}$, with r chosen as 0.9.
These weights populated alpha in the narrow band, and in other regions, alpha equaled alpha_trimap. Finally, the object was blended onto the new background using the equation:

$\text{Output} = \alpha * \text{Original\_RGB\_FRAME} + (1-\alpha) \cdot \text{NewBackground}.$

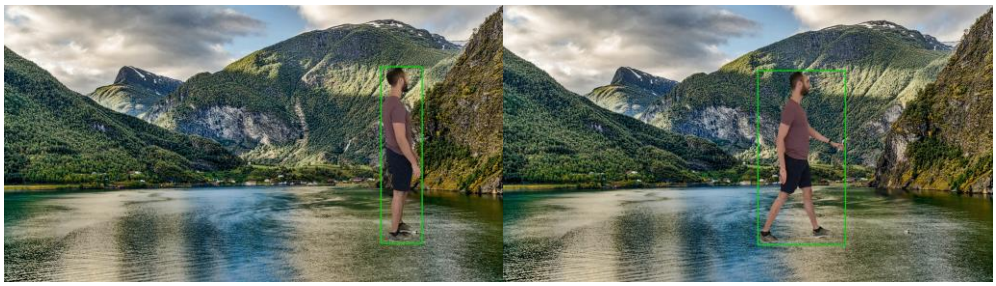
We included a check to ensure that the new background dimensions match the original frame, adjusting them using resize if necessary.

## Tracking:

We implemented tracking in two possible ways:

1. **Using the Binary Mask Calculated in the Background Subtraction Stage:**
    - As explained in the tutorial, we utilized the binary mask calculated in previous stages to separate the object from the background and draw a bounding box around it.
    - When reading each frame from the matted.avi video, we also read the corresponding frame from the binary.avi video, converting it to grayscale. We applied the cv2.connectedComponentsWithStats function (as detailed in the background subtraction stage) to the binary frames to obtain the bounding boxes of all labels. Since we know that, in practice, only two labels exist (background and object), as we preserved the binary frames in the background subtraction stage. We drew the bounding box on the color frames and saved the information to a JSON file.

Please note that the detailed explanation of the bounding box drawing and saving to a JSON file was omitted for brevity. If you need more details or clarification, feel free to ask.



The advantage of this method lies in precise tracking of the accurate contours of the derived object.

2. **Particle Filter (Previously Implemented in Homework 3):**
    - We utilized the particle filter that we implemented in a previous homework, which includes prediction functions, particle sampling from the current belief, normalized histogram calculation, and computation of the Bhattacharyya distance between the current histogram and the reference histogram from the first frame.
    - We chose to use 100 particles. After realizing that tracking on the full frames takes a long runtime, we introduced a scaling parameter to perform tracking on a smaller

image. Therefore, we downscaled the image by a factor of 10 in each dimension. The obtained coordinates were then brought back to the original dimensions.

- We calculated the initial position only based on the first frame, from the binary mask.
- After several attempts, successful tracking was achieved only when making a crucial assumption about the motion model—keeping the size of the bounding box constant, meaning the object does not move closer or farther from the camera.
- We defined the state vector $[x_{pos}\ y_{pos}\ w\ h\ \frac{dx}{dt}\ \frac{dY}{dt}]$ where the Gaussian noise variances of the model are [1, 1, 0, 0, 1, 1] = $[\sigma_x\ \sigma_y\ \sigma_w\ \sigma_h\ \sigma_{\frac{dx}{dt}}\ \sigma_{\frac{dy}{dt}}]$.

**Noteworthy considerations:**

- According to our estimation, there is relatively high variability between frames of the object during walking (legs and arms moving—sometimes open and sometimes closed), creating a significant portion of the background entering and exiting the bounding box and complicating the tracking. Therefore, we chose to reduce the width of the bounding box by a factor of 3 and track a "contracted" version of the object. Of course, in the final result, we multiplied the obtained width by 3 again.

**Tracking Decision:**

In this approach, there were occasional deviations from the object's body frame. However, the particle filter demonstrated resilience and could recover, returning the bounding box to a reasonable position.

The choice between the binary method and the particle filter is determined by the Boolean variable **use_binary**, passed to the tracking function. Since the particle filter worked only when certain assumptions were met and we weren't sure if these assumptions held for the additional video, we decided to use the binary mask tracking to make the solution more robust (i.e., set **use_binary=True**).

- Note: In the JSON file, we wrote the bounding box for each frame in the following format: $[Y_{center}, X_{center}, Half_{Height}, Half_{Width}]$ (the same format as in the particle filter in homework 3).

## Summary and Notes:

- Total runtime: approximately 10 minutes (Stabilization: ~2 minutes, Background subtraction: ~2 minutes, Matting: ~6 minutes, Tracking: 5 seconds).
- Throughout most of the project, we emphasized optimizations aimed at improving runtime (especially in the matting stage), achieving good results in half of the maximum runtime.
- Identified background subtraction as the stage that most influences the robustness of the algorithm, meaning it significantly affects the quality of the final result. Therefore, we endeavored not to rely too much on specific features from our video to prevent overfitting.
- The code was executed on a machine with Ubuntu 18.04 LTS and also on a virtual machine with Ubuntu 22.04 LTS.