**CHAPTER 4**

**SIMULATION AND IMPLEMENTATION OF VIDEO STABILIZATION ON MATLAB, PYTHON AND FPGA**

**4.1. Simulation Structure in MATLAB**

This simulation shows how to stabilize a video that was captured from a jittery platform. One way to stabilize a video is to track a salient feature in the image and use this as an anchor point to cancel out all perturbations relative to it. This procedure, however, must be bootstrapped with knowledge of where such a salient feature lies in the first video frame. In this simulation, a method of video stabilization was explored that works without any such a priori knowledge. It instead automatically searches for the "background plane" in a video sequence, and uses its observed distortion to correct for camera motion.

This stabilization algorithm involves two steps. First, the affine image transformations is determined between all neighboring frames of a video sequence using the estimateGeometricTransform function applied to point correspondences between two images. Second, the video frames are wrapped to achieve a stabilized video.

**4.2. Read Frames from a Movie File**

The first two frames of a video sequence are read as intensity images since color is not necessary for the stabilization algorithm, and because using grayscale images improves speed. Both frames are being shown below side by side, and a red-cyan color composite is produced to illustrate the pixel-wise difference between them. There is obviously a large vertical and horizontal offset between the two frames.

****

Figure 4.1. The First Two Frames of a Video Sequence



Figure 4.2. Red-cyan Color Composite Between Two Frames

(Frame A = red, Frame B = cyan)

**4.3.** **Collect Salient Points from Each Frame**

The goal of this step is to determine a transformation that will correct for the distortion between the two frames. The estimateGeometricTransform function can be used for this, which will return an affine transform. This function must be provided as input with a set of point correspondences between the two frames. To generate these correspondences, first, points of interest are collected from both frames, then select likely correspondences between them.

In this step MATLAB produce these candidate points for each frame. To have the best chance that these points will have corresponding points in the other frame, points around salient image features are needed such as corners. For that, simulation uses the detectFASTFeatures function, which implements one of the fastest corner detection algorithms.

The detected points from both frames are shown in the figure below. Observe how many of them cover the same image features, such as points along the tree line, the corners of the large road sign, and the corners of the cars.

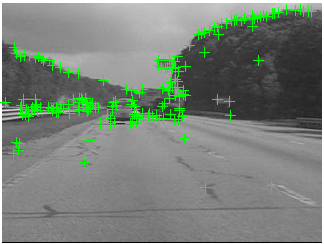


Figure 4.3. Corners in Frame A

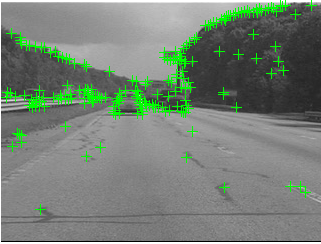


Figure 4.4. Corners in Frame B

**4.4. Select Correspondences Between Points**

This step pick correspondences between the points derived above. For each point, a Fast Retina Keypoint (FREAK) descriptor is extracted that is centered around it. The matching cost between points is the Hamming distance since FREAK descriptors are binary. Points in frame A and frame B are matched putatively. Note that there is no uniqueness constraint, so points from frame B can correspond to multiple points in frame A.

Match features which were found in the current and the previous frames. Since the FREAK descriptors are binary, the matchFeatures function uses the Hamming distance to find the corresponding points.

The image below shows the same color composite given above, but added are the points from frame A in red, and the points from frame B in green. Yellow lines are drawn between points to show the correspondences selected by the above procedure. Many of these correspondences are correct, but there is also a significant number of outliers.

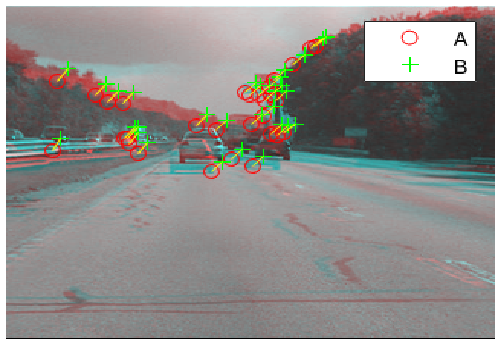


Figure 4.5. Correspondences Between Frame A and B

**4.5. Estimating Transform from Noisy Correspondences**

Many of the point correspondences obtained in the previous step are incorrect. But this simulaiton can still derive a robust estimate of the geometric transform between the two images using the M-estimator SAmple Consensus (MSAC) algorithm, which is a variant of the RANSAC algorithm. The MSAC algorithm is implemented in the estimateGeometricTransform function. This function, when given a set of point correspondences, will search for the valid inlier correspondences. From these it will then derive the affine transform that makes the inliers from the first set of points match most closely with the inliers from the second set. This affine transform will be a 3-by-3 matrix of the form:

(4.1)

The parameters *a* define scale, rotation, and shearing effects of the transform, while the parameters are translation parameters. This transform can be used to warp the images such that their corresponding features will be moved to the same image location.

A limitation of the affine transform is that it can only alter the imaging plane. Thus it is ill-suited to finding the general distortion between two frames taken of a 3-D scene, such as with this video taken from a moving car. But it does work under certain conditions.

Below is a color composite showing frame A overlaid with the reprojected frame B, along with the reprojected point correspondences. The results are excellent, with the inlier correspondences nearly exactly coincident. The cores of the images are both well aligned, such that the red-cyan color composite becomes almost purely black-and-white in that region.

Note how the inlier correspondences are all in the background of the image, not in the foreground, which itself is not aligned. This is because the background features are distant enough that they behave as if they were on an infinitely distant plane. Thus, even though the affine transform is limited to altering only the imaging plane, here that is sufficient to align the background planes of both images. Furthermore, if the background plane has not moved or changed significantly between frames, then this transform is actually capturing the camera motion. Therefore correcting for this will stabilize the video. This condition will hold as long as the motion of the camera between frames is small enough, or, conversely, if the video frame rate is high enough.

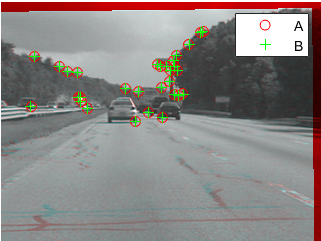


Figure 4.6. Color Composite Showing frame A overlaid with the reprojected frame B

**4.6. Transform Approximation and Smoothing**

This step could use all the six parameters of the affine transform above, but, for numerical simplicity and stability, some of the motion models are choosn to re-fit the matrix as a simpler scale-rotation-translation transform. This has only four free parameters compared to the full affine transform's six: one scale factor, one angle, and two translations. This new transform matrix is of the form:

(4.2)

To show that the error of converting the transform is minimal, frame B is projected with both transforms and show the two images below as a red-cyan color composite. As the image appears black and white, obviously the pixel-wise difference between the different projections is negligible.



Figure 4.7. Color Composite of Affine and s-R-t Transform Outputs

**4.7. Run on the Full Video**

This step is applying the above steps to smooth a video sequence. For readability, the above procedure of estimating the transform between two images has been placed in the MATLAB® function cvexEstStabilizationTform. The function cvexTformToSRT also converts a general affine transform into a scale-rotation-translation transform.

At each step the transform *H* between the present frames are calculated . The frames are fitted as an s-R-t transform, *HsRt*. Then these frames are combined with the cumulative transform, *H*cumulative, which describes all camera motion since the first frame. The last two frames of the smoothed video are shown in a Video Player as a red-cyan composite.

With this code, you can also take out the early exit condition to make the loop process the entire video.

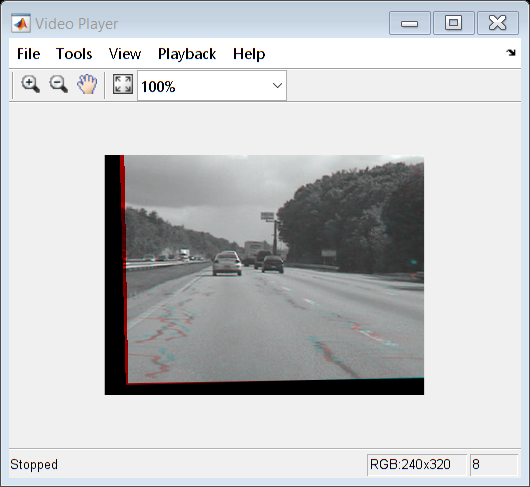


Figure 4.8. Stabilized Video in Video Player

During computation, we computed the mean of the raw video frames and of the corrected frames. These mean values are shown side-by-side below. The left image shows the mean of the raw input frames, proving that there was a great deal of distortion in the original video. The mean of the corrected frames on the right, however, shows the image core with almost no distortion. While foreground details have been blurred (as a necessary result of the car's forward motion), this shows the efficacy of the stabilization algorithm.



Figure 4.9. Raw Input and Corrected Sequence