Luisa Chiu

11/29/2023

Homework 4: Augmented Reality

Qualitative Evaluation Questions:

When does it produce accurate results and when does it falter?

The tracker seems to produce accurate results majority of the time, mainly when the image is being looked at straight on without anything obscuring the stones image. It falters slightly when majority of the reference photo is covered, for example when a hand was covering more than half of the image, or when the camera points at the picture at a harsh angle, making it difficult to find matching features.

Solution Summary:

For this homework, I created a few functions for two classes: RANSAC and Tracker. In the tracker class, I completed the compute_homography() function by first grayscaling the query frame. Next, I used the cv2 SIFT object to detect and compute matching features between the query frame from the input video and the reference image using the sift.detectAndCompute() and cv2.BFMatcher() functions. Using the ratio test, I filtered these matches and applied RANSAC to determine inliers and compute the homography. For augment_frame, I processed each video frame and used the input homography to put the overlay on top of the reference image. I used cv2.warpPerspective() to apply the homography, create a mask, and used alpha blend to achieve my expected output. In the RANSAC class, I completed three functions: compute_num_iter(), compute_inlier_mask(), and find_homography(). First, I used the given equation and the numpy library to calculate the number of iterations based off of the probability of success, outlier ratio, and sample size. Since we are focused on finding the number of iterations for a homography, I chose a sample size of 4. For the second function, compute_inlier_mask(), I reshaped my reference and query points and used cv2.perspectiveTransform to apply the homography on the reference points. Then, I found the error between the query points and reference points (after applying homography) using np.linalg.norm() to find the euclidean distance between the two. Next, I created my inlier mask for when the self.inlier_threshold exceeded the error. Lastly, in find_homography() for the RANSAC class, I followed the class slides and given code structure to creat the while loop that I iterate through to find the best homography and best mask. This was done iterating through four random query points and reference points, using cv2.findHomography() to compute the homography with method equal to 0, and creating the number of inliers for that homography. The variables were all updated if the number of inliers were greater than the maximum number of inliers, implying that the new/current homography was better than the previously calculated one.