CS6476 Project: Enhanced Augmented Reality

Le Hoang Van

1 Introduction

The goal of this project is to devise a pipeline that can consistently (a) detect a blank flat surface in the scene and (b) project an advertisement image onto it without the use of predefined markers. We experimented with a number of well-known methods in the literature, including Shi-Tomasi corner detector [10], SIFT [5], SURF [1], FAST [8], ORB [9], and AKAZE [3] and found that by combining Shi-Tomasi corner detector and Lucas-Kanade [6] method of estimating optic flow, we can strike a good balance of accuracy and speed. Methods such as SIFT and ORB, though more accurate at times, do not take into account the motion from one frame to the next, thus requires computation of keypoints' descriptors, which can be a bane to processing time. For example, on a *Macbook Pro 13 inch (2018)*, without writing to an output video, ORB runs at 14 FPS, AKAZE at 2 FPS, while the Shi-Tomasi and Lucas-Kanade combination (hereinafter referred to as *the algorithm* or *our algorithm*) runs at a whopping 24 FPS. In the remaining of this report, we will discuss in more details about our algorithm, how it works and what are the results of applying this algorithm to different footage.

2 Methodology

We set up the camera such that it is initially parallel to the planar surface that we are filming, meaning the advertisement image will be inserted as-is onto the first frame of each source video, since there is no projective transformation needed.

Before we can overlay the advertisement image onto the first frame, we have to determine its location and scale. Details on how to do it can be found in sub-section 2.1. Once the position and size are known, it is straightforward to find the coordinates of the four corners of the advertisement. These are required for warping the advertisement in subsequent frames, when the perspective changes.

Starting from the second frame onward, we need to compute the homography matrix between the current frame (t) and previous frame ($t-1$). To that end, we need at least four corresponding pairs of feature points from these frames; details on how to detect and track these features are in sub-sections 2.2 and 2.3, respectively. Using the homography matrix, we can compute the coordinates of the corners of the advertisement in frame t, given their coordinates in frame $t-1$. We then compute another homography matrix from the original advertisement image corners and the new coordinates, and warp the advertisement into these coordinates accordingly. We repeat the same process until we have reached the end of the video.

2.1 Find a blank patch

A blank patch is a region in the image where there are no discernible textures or visible features. Therefore, such a region would have no clearly defined corners or edges and would appear almost black in a gradient image. If we threshold the gradient image in such a way that 0 denotes an edge or corner points and 1 denotes plain, textureless background points, then after applying Euclidean distance transform on the entire image, we can blot out various features in the image.

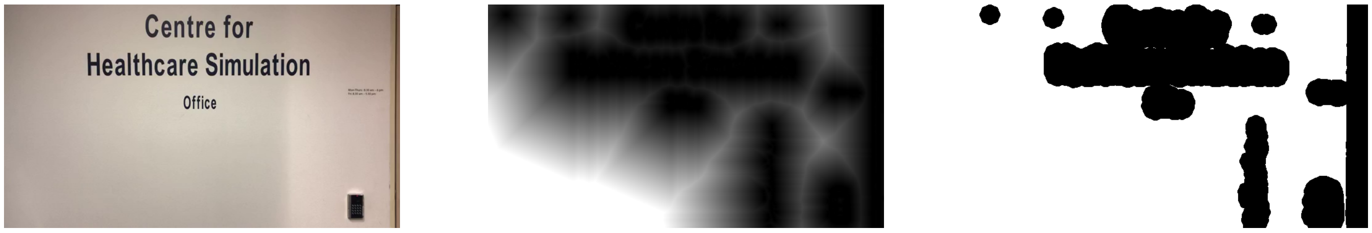


Figure 1: (Left): Original scene (Middle): After distance transformation (Right): After distance transformation and thresholding

The result (Figure 1 (Right))) will be a binary image where background points are represented with white pixels. We will be referring to this image as the *mask*. Now, we can use a sliding window with the same aspect ratio as the advertisement image but with varying sizes, to determine the location at which to insert the resized advertisement so that the image will occupy as much blank space as possible while at the same time anchored to the center of the scene without occluding existing features. For each combination of corner positions and scaling factor, we calculate a score S that comprises of two parts: $S = C - D$, where C is the count of white pixels in the *mask* image, which determines how many pixels the advertisement occupies and whether it is obscuring any existing features, and D is the sum of distance from the four corners of the advertisement to the corresponding corners of the scene, to keep the advertisement as centered as possible. The combination that yields the highest score will be selected. Figure 2 demonstrate an advertisement successfully scaled and inserted into a blank patch of a scene.

2.2 Detect features

We use Shi-Tomasi algorithm to detect corners to be used as keypoints for our tracking, which is based on Harris corner detector [4], but with a slightly modified scoring criteria, making it much more accurate and robust than the original. This algorithm is implemented in `OpenCV` as `cv2.goodFeaturesToTrack` method. We then refine the corner location by `cv2.cornerSubPix` to achieve sub-pixel accuracy, which will help with our tracking.

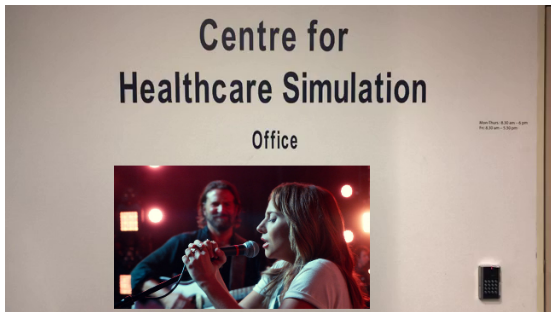


Figure 2: Original scene with advertisement

2.3 Track the features

Given the keypoints, we use Lucas-Kanade algorithm over an image pyramid of two levels to track their motion between consecutive frames. This algorithm is implemented in `OpenCV` as `calcOpticalFlowPyrLK` method. Figure 3 shows the tracking in action (the traces will be removed in the final output.)

However, we don't detect keypoints for every new frame, else the tracking might be unstable, since the features detected in frame $t-1$ may not be detected in frame t. It is not a good idea to detect features only on the first frame, either, because features may go out of view and come into view again. Instead, we want to detect new features only when the number of features being tracked drops below a certain threshold, or when the tracked features become too cluttered. Having

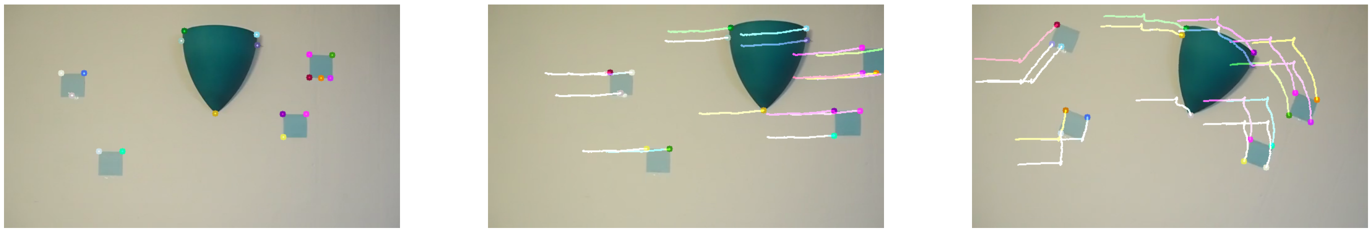


Figure 3: (Left): Keypoints in the first frame (Middle): The same set of keypoints, with traces (Right): New keypoints, with traces

evenly spread out keypoints will allow us to compute more accurate homography matrix later on and prevent the advertisement from being skewed toward a particular clutter of points.

2.4 Insert the advertisement

Using the tracked keypoints, we can compute the homography M using RANSAC method between corresponding features in consecutive frames. `OpenCV` provides an implementation of this via the `cv2.findHomography` method. Given M and the set of corner coordinates in frame $t-1$, we can compute the corresponding corner coordinates in frame t, using the `cv2.perspectiveTransform` method. After that, we need to compute another homography matrix M', which will do the warping from the original corner coordinates to the corner coordinates in frame t using the `cv2.remap` method. Figure 4 shows the inserted advertisement under different camera orientations.

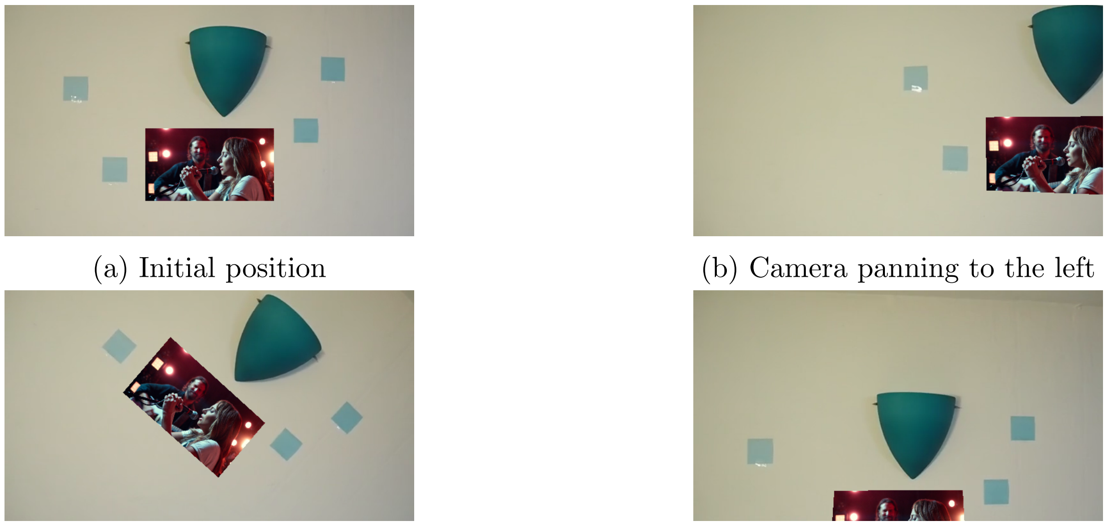

(a) Initial position
(b) Camera panning to the left
(c) Camera rotating to the left
(d) Camera moving upward and zooms in

Figure 4: Markerless AR in action!

3 Results

Although an algorithm for automatically determining the scale and location of the advertisement image in the scene is described in sub-section 2.1, we did not use it to produce the final output. Instead, we will provided the height or width of the advertisement and let the algorithm calculate only the location, to ensure that the inserted advertisement would not sit too close to any of the edges or features. This is purely for aesthetic purposes. We ran the script on three videos with varied lighting conditions and number of features. All videos were shot with an iPhone 8 Plus camera under default video mode without a tripod.

- Video 1: Living room wall with a lamp and a few stick-it notes. This is an example of **bad** lighting and **few** features.
- Video 2: Classroom wall with text and a lock device. This is an example of **adequate** lighting and **relatively many** features.
- Video 3: Library hall wall with framed articles. This is an example of **excellent** lighting and **lots of** features.

Some of the better output frames for Video 1 are shown in Figure 4 but overall the quality is not as good as good as the output videos for Video 2 and Video 3. Figure 5 demonstrates a few not-so-good frames with visible distortions. Generally, the more objects we have in the background, the more corners or features there are and the more information we have to compute a more accurate homography. Lighting condition is one of the most important factors as better lighting conditions make the features pop out against the background and the corners more defined, resulting in higher quality keypoints. Another critical factor is stabilization; having a good video capturing system setup would eliminate motion blur, which would destroy all the features and something we should always try to avoid. The type of transition (panning/rotating/zooming), however, has almost no effect on the result image. As long as we can make sure there are plenty of good features to track even when we are moving the camera, then there should be no loss of quality whatsoever.

Figure 6 demonstrates some output frames from Video 2. There is some flickering going on in the output video when we zoomed in on the wall, which is only apparent when viewing the video. Since a large number of features start to disappear as we zoom in the advertisement, it struggles to keep its shape. The result is a slightly skewed image.

Figure 7 shows several output frames from running our algorithm on Video 3. Thanks to good lighting condition and abundant objects in the background, the warped advertisement doesn't move or change its shape or display any kinds of distortion as the camera moves around.

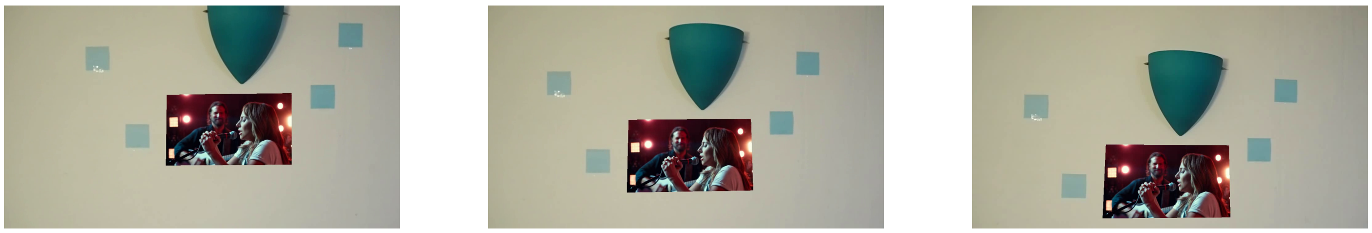
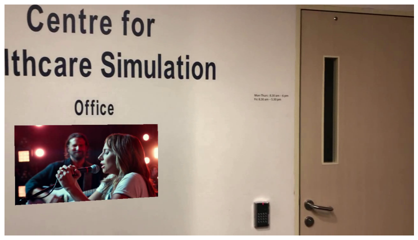


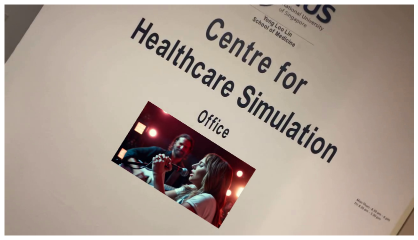
Figure 5: Examples of defective warping from output of Video 1

4 Performance statistic analysis

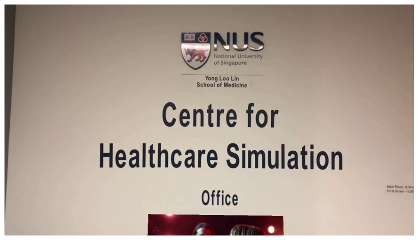
We ran `cProfile` on all three videos and the result is summarized in Table 1. Our algorithm achieved on average 20 - 22 FPS. Warping image was the most costly process, taking up more than 50% of the total runtime. The reason is that there are a lot of matrix multiplications and array casting involved in this step, both are expensive operations whose cost is directly proportional to the resolution of the source image.



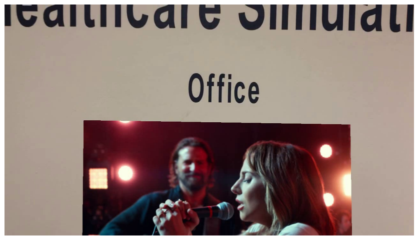
(a) Camera panning to the left



(b) Camera rotating to the right

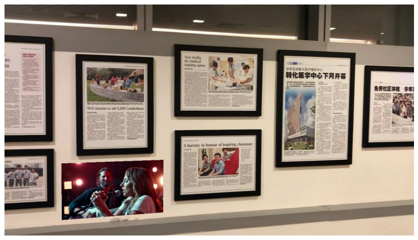


(c) Camera moving upward

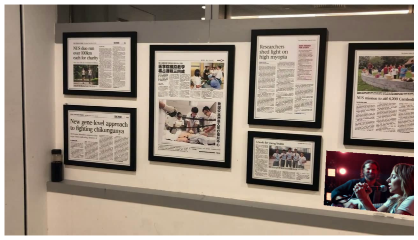


(d) Camera zooming in

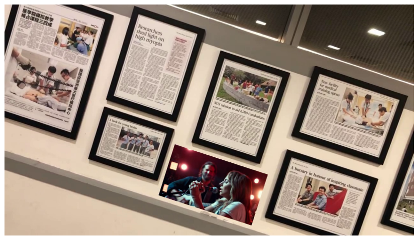
Figure 6: Example output frames from running the algorithm on Video 2



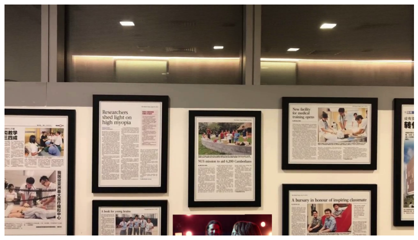
(a) Camera panning to the left



(b) Camera panning to the right



(c) Camera rotating to the right



(d) Camera moving upward

Figure 7: Example output frames from running the algorithm on Video 3

There is quite some variations in the time taken for feature detection between Videos 1 & 3 and Video 2. This is mostly due to the difference in shooting distance and how the transitions were created and also the nature of objects in the background, which resulted in new features being generated and tracked more often in Video 2 than the others. Having identified the parts that were more time consuming, we now have a clearer idea of what can be done to get closer to real time processing (24 - 30 FPS). If we can shoot the video in perfect lighting i.e. with sunlight as the light source then that would remove the need to de-noise the image, saving us 3 - 3.5% in run time. If we can film the scene in such a way that not a lot of features are disappearing from one frame to the next and that the features are evenly spread out, then we will not have to call the feature detector so many times to re-generate features. This would be another 2 to 2.5% saving in time. Without writing the video out, we can also save up to 10 - 14% of time. In addition, we can get some speed-up if we switch to a compiled language such as `C++`, though not by a significant margin as although we are using `Python`, all the heavy-lifting matrix operation is being done in `C++` anyway.

	Video 1	Video 2	Video 3
Pre-process (grayscale, smooth)	3.00	3.12	3.65
Track features	3.89	5.27	3.69
Detect features	6.74	0.14	7.08
Calculate homography	0.32	0.61	0.39
Warp image	55.24	55.00	44.67
I/O	12.53	15.05	10.56
Total time	84.64	80.92	72.27
Total number of frames	1784	1767	1440

Table 1: Runtime of algorithm on different videos

5 State-of-the-art and future work

Modern AR platforms such as ARCore and ARkit can consistently achieve real-time processing running on mobile devices across a wider range of camera inputs, thanks to an algorithm called SLAM [2].

Figure 8 describes the standard architecture of SLAM, which comprises of four main modules. On mobiles, **sensor data** is primarily camera, augmented by accelerometer, gyroscope and depending on the device light sensor. **Front end** is where feature extraction and keypoint identification happen. SLAM also has a **back end** system that does error correction to compensate for the drift and also takes care of localizing pose model and overall geometric reconstruction. The result containing the tracked features and locations are called a **SLAM estimate**. Since the back end component is most relevant to our algorithm, we can get a handle of how it can be further improved by understanding how SLAM works.

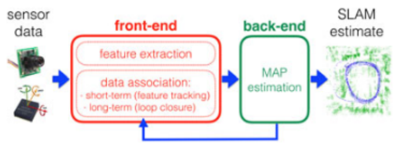


Fig. 2. Front end and back end in a typical SLAM system. The back end can provide feedback to the front end for loop closure detection and verification.

Figure 8: Components of a SLAM system

To this end, we will look at a recent implementation of open source SLAM called ORB-SLAM [7]. The algorithm, as a first step, uses ORB to find keypoint and generate binary descriptors – which can be likened to us detecting features with Shi-Tomasi. Then it looks for new frames, performs keypoint detection and tries to match these keypoints with those from the previous frame to get a spatial distance – we did something similar with our Lucas-Kanade algorithm. In the next step, it refines the camera pose by projecting the estimated initial camera pose into next camera frame to search for more keypoint which corresponds to the one it already knows – we did not take this into account in our current pipeline, but this is something that would be done in future work with an error correction algorithm such as Kalman filter.

In addtion, one of the goals of AR is when you walk back to your starting point it should understand you have returned. The inherent inefficiency and the induced error make it hard to accurately predict this. This is called loop closing for SLAM. ORB-SLAM handles it by defining a threshold. It tries to match keypoints in a frame with next frames and if the previously detected frames matching percentage exceeds a threshold then it knows you have returned.

References

[1] H Bay, A Ess, T Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. *Computer Vision and Image Understanding*, 110:346–359, 01 2008.

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, Dec 2016.

[3] Pablo Fernández Alcantarilla. Fast explicit diffusion for accelerated features in nonlinear scale spaces. 09 2013.

[4] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.

[5] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[6] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[7] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015.

[8] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. volume 3951, 07 2006.

[9] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, Nov 2011.

[10] Jianbo Shi and Carlo Tomasi. Good Features to Track. pages 593–600, 1994.