Final Project: Enhanced Augmented Reality

Matthew Molinare
mmolinare@gatech.edu
CS 6476 Computer Vision

Abstract

I present a tracking algorithm for augmented reality using feature matching on a planar structure in a scene. I perform run-time benchmarking of my video processor and discuss methods for optimization, including the animation of the target image between skipped frames.

1 Introduction

My enhanced augmented reality Python module (*ear*) utilizes feature detection, feature matching, and robust spatial transform estimation routines from the OpenCV library [5]. Once the target image has been initially mapped onto markerless, planar surface in the scene, it is tracked by updating a cumulative homography matrix that is used to project the image into the next frame of the video sequence. In order to reduce the frequency of homography estimation, a user-specified number of frames are streamed and cached after each update. At the next update, the target image is animated at each of the skipped frames by directly interpolating the components of cumulative homography matrix from the previous update.

2 Initialization

The function *get_initial_bbox* is used to determine the best initial placement of the target image on a textureless surface in the scene. It is assumed that the geometry of the scene is predominately planar and orthogonal to the camera principal axis. First, a median filter is used to perform edge-preserving smoothing on the initial video frame. Next, the edge gradient magnitude G is computed as,

$$G = |G_x| + |G_y| \tag{1}$$

The gradient components G_x and G_y are the products of the convolution with a horizontal and vertical Sobel filter, respectively. The gradient magnitude provides a measure of the strength of an edge at each pixel. A binary threshold (*edge_thresh*) is applied to G, yielding an inverted edge mask with values of 0 where there is a detected edge and values of 255 elsewhere. Next, the OpenCV function $cv2.distanceThreshold$ is used to compute pixel-wise Euclidean distances to the nearest edge element of the mask. A circle is drawn at the location and radius of maximal distance (Figure 2). The bounding box is defined by the largest rectangle, whose aspect ratio matches that of the target image, that fits inside the circle.

3 Homography Estimation

Several OpenCV routines are used to estimate the homography that best maps the target image between two video frames. Interest points are detected using the Oriented FAST and Rotated BRIEF (ORB) algorithm [7]. The oriented FAST algorithm acts as a local contrast filter, comparing the intensity of each pixel with a circular window of pixels around it; the top $num\_features$ number of features are found by applying the Harris corner metric,

$$R = \det M - k(\operatorname{trace} M)^2 \tag{2}$$

where M is the structure tensor of the video frame image and k is a constant in the range 0.04 to 0.06. The matrix M can be computed from the image gradients in (Eq. 1).

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} G_x^2 & G_x G_y \\ G_x G_y & G_y^2 \end{bmatrix} \tag{3}$$

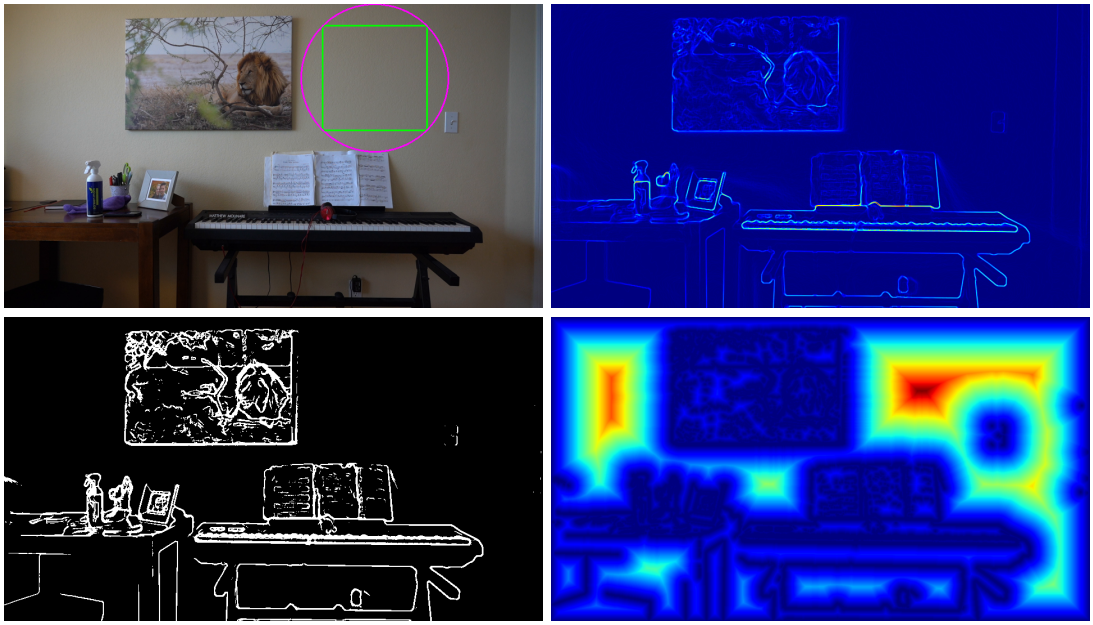

Figure 1: The initial bounding box is placed a the location of maximal distance to the nearest detected edge. The green rectangle indicates the placement of an insert with a 1:1 aspect ratio (top-left). After smoothing, the edge gradient magnitude is computed (top-right) and threshold to form the edge mask (bottom-left). The distance transform of the inverted edge mask provides information about the location and size of the bounding box over a region of low gradient energy (bottom-right).

where $w(x, y)$ is a window function. Once the features have been detected, the Brute-Force Matcher (BFM) is used to match features between two video frames [6]. For every feature in the first frame, BFM returns the feature in the second frame whose distance across ORB feature vectors is minimal. All matches are then sorted by this distance. The OpenCV function $cv2.findHomography$ is used to robustly estimate the homography that best aligns the top $num\_matches$ number of matches. The RANSAC method of this routine detects outliers by fitting the homography model to many random sample subsets of the input data; it selects the set of model parameters that aligns the greatest number of features with their respective matches within a user-specified $ransac\_thresh$ distance threshold.

An optional Adaptive Non-Maximal Suppression (ANMS) routine is used to get features that are evenly distributed throughout the image. Based on the method proposed by Brown et al. [1], features are sorted based the minimum distance to any other feature with a greater Harris corner response (Eq. 2). In theory, ANMS can reduce the computational cost of matching by restricting the number of features extracted from each video frame. However, it was found to have a longer run time than BFM with negligible performance benefits in terms of the amount of observed drift. The ANMS routine was implemented using masked arrays in NumPy; this is syntactically simple, yet computational inefficient approach. One way significant speedups

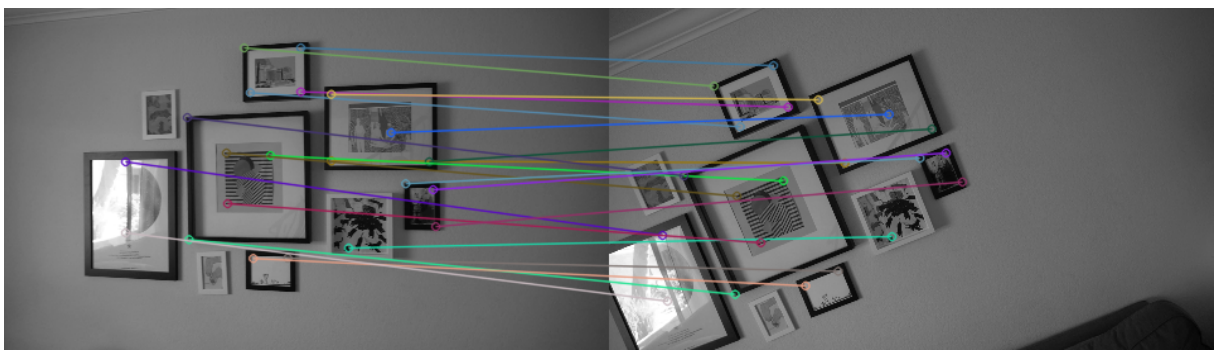

Figure 2: ORB features are correctly matched between two video frames exhibiting a significant shift in perspective using a combination of BFM and RANSAC.

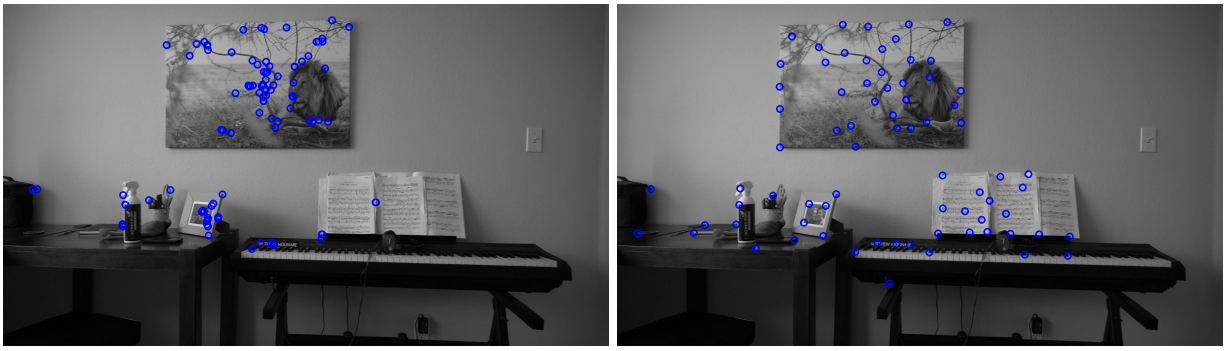

Figure 3: ANMS results in a more uniform distribution of keypoint features. The left image shows the top 100 ORB features from a single video frame. The right image shows the first 100 out of the top 5,000 ORB features sorted using ANMS.

to ANMS could be achieved is by creating a Cython extension to the Python source code that eliminates redundant computations of the pairwise distances between features. In practice, the use of ANMS was not necessary to generate satisfactory output products.

4 Tracking

The target image is mapped into video frame i using the cumulative homography matrix H_i. The initial homography H_0 is computed directly using the method of least squares on the four point correspondences given by the corners of the target image and the corners of the initial bounding box. A user-specified number of frames N (or *frame_step* in the configuration YAML file) is streamed from the video sequence and cached. At each frame $i = jN$ (where $j = 1, 2, \dots$), the homography H_{j-1}^{j} that maps the insert from frame $j-1$ to j is robustly estimated using the techniques described in the previous section. In the function *update_homography*, the cumulative homography is computed as,

$$H_j = H_{j-1}^{j} H_{j-1} \tag{4}$$

Direct matrix interpolation is then used to approximate the partial homographies corresponding to each of the skipped frames in the cache. That is, using H_{j-1} and H_j, the function *interpolate_homographies* is used to linearly interpolate each component of the matrix separately,

$$H_i = (1-t)H_{j-1} + tH_j \tag{5}$$

where,

$$t = i \,(\mathrm{mod}\, N) \text{ for } i = jN, jN+1, \dots, (j+1)N \tag{6}$$

Although this method is significantly faster than estimating a homography using extracted keypoint features, it is not without drawbacks. Shoemake and Duff propose that a point transformed by a weighted sum of matrices equals the weighted sum of the transformed points [3]. This behavior results in the distortion of the target image under rotation. The authors propose a method to decompose a homogeneous matrix into its primitive matrices for translation, rotation, scale, shear, and perspective. These components can be interpolated separately to achieve a rigid shape animation. It is assumed, however, that for sufficiently high frame rates, small N, and smooth camera movements, the angle of rotation between updates is small enough such that the distortion resulting from direct homography interpolation is imperceptible. If these conditions are not satisfied, the animation will not be able to capture irregular or high-frequency motion.

5 Results

The *ear* video processor was applied to three interior scenes. See the References section at the bottom of this document for a link to a video presentation in which the user interface to the code is demonstrated and both positive and negative results are shown [11]. Additionally, there are links to each of the output videos in their entirely [12]. Each of these scenes contained a blank section of wall upon which the target image was projected. One of the reason for success was that the feature-dense regions of the scene were

Routine	Time (s)	FPS	% Total Time
Adaptive Non-Maximal Suppression	22.2	58	31.6
Video I/O	22.0	59	31.4
ORB Feature Detection	10.0	129	14.3
Image Warping	6.6	196	9.4
BFM Feature Matching	2.4	538	3.4

Table 1: Run-time benchmarking on a Dell XPS 13 9370 equipped with a 1.8 GHz Intel Core i7 processor and 8GB of RAM. A single thread was used to process a video consisting of 1,291 frames at 1080p resolution with a cache size of $N = 8$. Routines related to ANMS and file I/O have have the greatest impact on run-time.

either located on the same wall or were nearly co-planar; this allowed the homography to accurately relate the same planar surface in space between two frames.

Using a large number of detected ORB features in lieu of applying ANMS on a smaller number of features resulted in the most favorable balance between run-time and visual quality. Table 6 lists the CPU computing time of each major routine in a single-threaded framework for a high resolution input video. Excluding file I/0, ORB and ANMS are the two most computationally expensive processes. The decision to eliminate the use of ANMS in the default processing mode was supported by the assumption that the planar surface has a lot of texture, allowing it to be sufficiently sampled by detected feature points across its entire observable extent.

The parameter with perhaps the greatest impact on performance was the number of frames interpolates (N). Figure 5 shows the effect of N on drift. Stochastic noise inherent to feature detection results in the accumulation of errors from the least-squares estimation of the alignment between matches. Therefore, the fewer updates that occur, the less drift is observed. Additionally, the computational cost of linear interpolation is nominal compared the full feature-based method. This assumes, of course, that there are no hardware constraints to memory caching. Perceptually, however, increasing N may result in unnatural (non-rigid) or asynchronous motion. If the camera movement is to rapidly accelerate, the target image projection may appear to lag behind the scene. This is observed beginning when the time between updates approaches roughly a quarter-second. For a video recorded at 60 frames per second, this corresponds to $N = 15$.

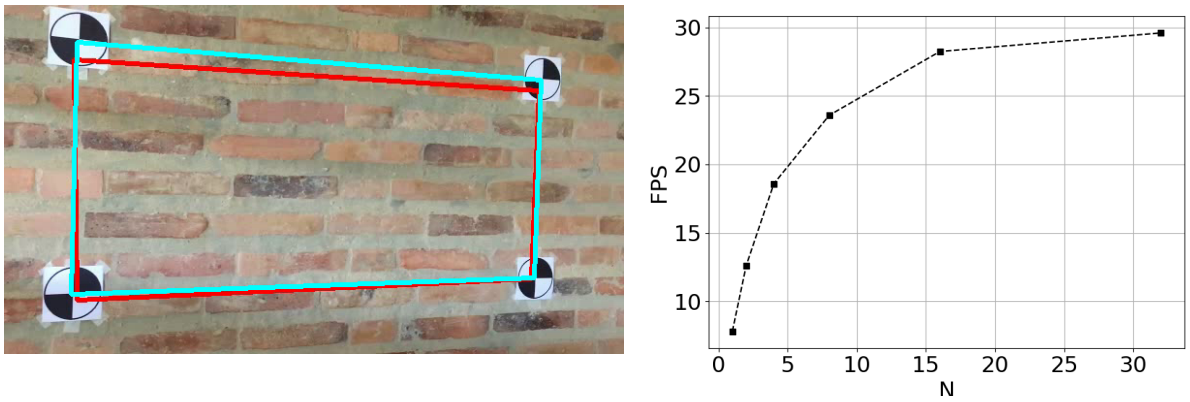

Figure 4: Left: Increasing the number of frame interpolates (N) decreases the accumulation of alignment error from each update. The effect of N on video sequence comprised of 1,146 frames is visualized here. Notice the amount of tracking drift of the red bounding box ($N = 1$) compared to the cyan bounding box ($N = 20$). The corners of the bounding boxes were initially placed on the black and white markers. Right: The video processing frame rate increases with N; significant improvements are observed until around $N = 8$.

Video	FPS @ 720p	FPS @ 1080p
living_room	55	26
kitchen	52	24
office	58	26

Table 2: Run-time comparison between input videos. Each of the videos were processed using 2,000 ORB features (with ANMS disabled) and a cache size of $N = 8$.

6 Conclusion

Integrated augmented reality systems rely on measurements from multiple sensors. For example, Apple's ARKit framework leverages the device's visual system as well as its Inertial Measurement Unit (IMU) to directly track the camera's pose. This information is combined using a Kalman Filter to get the most reliable estimate of true position [8]. Google's ARCore goes even further, integrating planar surface detection and light estimation to add realism to the interaction between a virtual object and its environment [9]. The position tracking capabilities of ARCore are also being outside the scope of augmented reality. For example, it is possible to achieve indoor real-time navigation up to an accuracy of 2cm on a mobile device using the simultaneous localization and mapping (SLAM) rountine in the ARCore SDK [10]. The methods for vision-based tracking discussed in this report are fundamental to these technologies. Even without the aid of additional sensor information, it has been shown that a markerless surface can be tracked in a 720p resolution video at nearly 60 frames per second using solely texture information in the surrounding scene.

References

[1] M. Brown, R. Szeliski, and S. Winder. *Multi-Image Matching using Multi-Scale Oriented Patches*. CVPR (1). 2005.

[2] G. Simon, A. W. Fitzgibbon, A. Zisserman. *Markerless Tracking using Planar Structures in the Scene*. Proceedings IEEE and ACM International Symposium on Augmented Reality (ISAR 2000). IEEE, 2000.

[3] K. Shoemake, T. Duff. *Matrix Animation and Polar Decomposition*. Proceedings of the Conference on Graphics Interface. 1992.

[4] Notes on Adaptive Non-Maximal Suppression
`http://cs.brown.edu/courses/cs143/2013/results/proj2/valayshah/`

[5] OpenCV (Open Source Computer Vision Library)
`https://opencv.org/`

[6] Brute-Force Matcher Documentation
`https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html`

[7] ORB (Oriented FAST and Rotated BRIEF) Documentation
`https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html`

[8] M. Miesnieks. *Why is ARKit better than the alternatives for AR?* A Medium Corporation. 2017.
`https://medium.com`

[9] ARCore Overview
`https://developers.google.com/ar/discover/`

[10] R. Mendez. *Indoor Real-Time Navigation with SLAM on Your Mobile* Arm Community.
`https://community.arm.com/`

[11] Video Presentation
`https://youtu.be/L5L6Jtl2ygY`

[12] Output Videos
`https://youtu.be/ua4H3Via6o8`
`https://youtu.be/6s6_oeDP7g8`
`https://youtu.be/mi3BBgcRhHA`