# CS6476 FINAL PROJECT: ENHANCED AUGMENTED REALITY

**Yap Jun Hao Alson**
Spring 2019
Georgia Institute of Technology
jyap30@gatech.edu

October 25, 2019

**ABSTRACT**

This assignment requires us to find a flat surface in the video scene and project an image onto it without the use of defined markers. The image must then stay at the initial location regardless of any movement such as rotation, zooming and the image must remain at the same location when the camera's view shifts away and returns back. To find a spot for insertion of image, the Canny edge detector is employed to find a 101 by 101 area without edges that is closest to the centre. Features in the scene are detected and its corresponding descriptors computed for each frame so that a homography matrix can be derived to project an image onto each frame and its consecutive frame. Several algorithms such as ORB, BRISK and AKAZE are used that yield outputs of varying quality and speed, with BRISK being the best without compromising too much on both aspects.

## 1 Introduction

Augmented Reality (AR) is the usage of computer technology, specifically vision-based tracking, to project an object into scenes that we can see through virtual images. In one of the assignments in this course, we have already tackled on marker-based tracking and projecting an image within the boundaries of the markers. Now, we move on to the "second-half" of the vision-based tracking work as it is concluded to be either marker based or marker-less[1]. In this final project topic, we are tasked to do the same but with 2 extra challenges. Firstly, we have to do so without the usage of markers which means that we need a solution to find the most appropriate area to project the image onto. Secondly, the camera's view would be leaving the area where the image is and upon returning, it should still stay at the exact location.

I have derived an algorithmic way to detect for a flat surface to project an image onto using the Canny edge detector (explained further in Section 2.1). Next, I implemented a second algorithm which would detect keypoints, compute their descriptors, perform matching, derive the homography

matrix coupled with random sample consensus (RANSAC, as taught in lecture) and then project the image onto the scene. Here, I would be employing an accumulated homography update approach as seen in [2] which worked out well.

In addition, there will be discussions of the results about different approaches that have been tried and tested such as using different algorithms such as AKAZE and implementing non-maximal suppression technique (shown in [3]) to encourage a more evenly spatial distribution of keypoints detected. Thereafter, I touch on the pros and cons of using this approach, how does it compare to the state-of-the-art methods and future considerations for improvements.

## 2 Implementations of Algorithms

The next 2 sections would cover the solutions used to tackle this problem.

### 2.1 Finding a flat surface

A flat surface is akin to an area without edges which would mean that if the Canny Edge operator is employed on the image, then we hope to find an 'empty' patch within.

The solution starts off with taking the first frame of the video and apply a Canny edge operator on it to get a binary output with white lines indicating areas where connected edges can be detected. These edges are given a value of 1 while the rest of the image are given values of 0 which appear dark/black. A uniform box kernel (all pixels given a value of 1) of size 101x101 is created as I want the projected image to be of that size. The binary output image is then convoluted with this kernel. For each pixel, the kernel will sum up the values of the (101x101)-1 neighbouring pixels resulting in an output image that is either a value of 0 or more than 0. If the value is 0 at a pixel, that means that all the values summed up centred at that pixel is 0, which would mean that no edges are found in that area.

Since there can be multiple occurrences of such summed pixel being 0, I would want to choose the one that is closest to the middle. Thus, a simple function is created to calculate the euclidean distance between 2 pixels. All pixels that have a summed value of 0 are computed for its euclidean distance against the centre point of the image, then the pixel with the shortest euclidean distance is chosen as the area to project the image onto.

### 2.2 Projecting a motion-invariant image

Now that we have the coordinates of image's 4 corners, we have to project the image onto the first frame. This was covered back in the PS03 Assignment so a similar technique will be used here.

Next, we will grab the second frame and perform keypoints detection, descriptors computation, matching, non-maximal suppression (optional), compute homography, projection and write the frame into the video generation. After that is done, we continue the same loop of techniques till the end of the video. The following sections will cover each of this technique sequentially.

### 2.2.1 Keypoints detection and computation of descriptors

In the lecture, we have covered the technique of SIFT[4] (Scale Invariant Feature Transform) to detect and describe local features in image. However, that was a technique introduced back in 1999 and there have been new techniques introduced since then. In this paper[5], it compares SIFT together with other new techniques introduced since then. The results of the paper have led me to try out ORB, BRISK and AKAZE to tackle this problem. SIFT and SURF are excluded because they don't seem to perform highly if computational efficiency is taken into account and partially due to them being patented, albeit the paper [5] does suggest that SIFT is the most accurate algorithm.

ORB stands for Oriented FAST and Rotated BRIEF, with its introduction and full technical details shown in this paper[6]. It leverages on the FAST[7] technique. FAST scans through all pixels, by evaluating the intensities of a circle of 16 pixels around each pixel and evaluate if it's a corner with a given threshold. It also comes with a machine learning approach and non-maximal suppression built in. However, FAST do not have an orientation component and multiscale features so ORB adapted a modified version of it. Next, it adapted a modified version of BRIEF[8] technique to compute descriptors. BRIEF converts the keypoints into a binary feature vector which is way faster than computing SIFT descriptors. Yet, it needs to be modified to be invariant to rotation so ORB uses Rotated BRIEF (rotation-aware BRIEF) without losing out on the speed aspect of BRIEF significantly.

BRISK stands for Binary Robust invariant scalable keypoints as described here [9]. It detects corners using the AGAST (Adaptive and Generic Accelerated Segment Test) algorithm described here [10] which claims to be faster than FAST, then filtering them with FAST Corner score while searching for maxima in the scale space pyramid. The eventual descriptor is also constructed as a binary string with rotation-invariant and illumination-invariant properties.

Lastly, we look at AKAZE[11] (accelerated-KAZE, where it builds on KAZE[12] which is computationally intensive). AKAZE proposed using *Fast Explicit Diffusion (FED)* embedded in a pyramidal framework to detect features in nonlinear scale space much faster and introduced *Modified-Local Difference Binary* (M-LDB) descriptors that are scale and rotation invariant.

### 2.2.2 Matching of descriptors

After we have obtained 2 sets of descriptors - one set from an image and other from its next consecutive image, we can now perform matching of descriptors and keep those that match best. This is done by using the *Brute-Force Matcher* object (**cv2.BFMatcher()**) available in the OpenCV library. It takes the descriptor of each feature in one set and match against all features in the second set using some distance calculation and then returning the closest one. Since the 3 algorithms produce binary string descriptors, then the distance calculation function used here is Hamming distance (**cv2.NORM_HAMMING**) which is simply the number of positions at which the corresponding symbols are different. Also, the parameter of *crossCheck* is used here which ensures that each feature in one set has another feature in the second set as the best match, and vice versa. This is a good alternative to the ratio test proposed in [4]. *crossCheck* was used to create all the output videos instead of Lowe's ratio test.

#### 2.2.3 Adaptive Non-Maximal Suppression (ANMS)

This method of selecting a subset of keypoint matches is described in the [3] paper. This selection algorithm will run on each of the 2 sets of keypoints found after the matching step above. Within each set of keypoints, each keypoint will be compared to all other keypoints, find the shortest distance of the keypoint that has a higher corner strength/response than itself and record the distance down. After that, all the keypoints will be sorted in descending order of the distance found for each keypoint and select the top N keypoints, where N can be tuned. In doing so, it ensures a better spread of keypoints which are determined to be a local maxima within a radius.

This step here is optional and I am using it for ORB implementation only to trim down the number of keypoint matches. The reason why this is not used for BRISK and AKAZE is because the time taken for each frame would be too long (See Table 1).

#### 2.2.4 Computing homography matrix and projection of image

With the 2 sets of features and descriptors obtained, the homography matrix is computed using **cv2.findHomography** with the usage of RANSAC. With this matrix, it is used with the function **cv2.warpPerspective** to project the image onto the source image frame before writing into the video.

## 3 Experiments and Findings

In this section, we compare the speed and quality of using different algorithms for computing keypoints and descriptors. Do note that these results are produced on an early 2015 MacBook Pro with 2.7 Ghz dual-core Intel Core i5 and 8 GB RAM. Results may differ on different machines.

### 3.1 Comparison of algorithms analysis

Algorithms will now be compared on a single video and its performance judged by the speed and quality of video produced.

| Algorithm | No. of Keypoints | Part Avg FPS | Quality |
|---|---|---|---|
| ORB | 500 | 23.5 | Very bad |
| ORB | 1500 | 11.8 | Bad |
| ORB | 2500 | 5.0 | Bad (better than ORB 1500) |
| BRISK | - | 5.3 | Perfect |
| AKAZE | - | 4.8 | Perfect |

Table 1: Avg FPS for each algorithm used for the *room* video

The average FPS is computed by running 3 times of each algorithm on the *room* video. Quality is determined by the amount of flickering or motion seen by the projected image. As evidently seen, the ORB algorithm perform worse than the BRISK and AKAZE algorithms. However, the ORB algorithm does perform better with more features to be detected and its speed generally faster

than the other 2 algorithms. Yet, when 2500 keypoints is used, it reaches a FPS that is close to BRISK's but the video produced is still not as good. BRISK and AKAZE are very similar for quality produced but BRISK is slightly faster.

In other videos such as the *wall* video, ORB 2500 can perform almost as well as BRISK and AKAZE (please find in the output video folder).

### 3.2 Breakdown analysis

Next, we will investigate how much time is taken for each of the component in each frame and explore how it differs across the algorithm used.

| | Component's absolute time and proportion of total time per frame | | | | | |
|---|---|---|---|---|---|---|
| Metric | ORB 500(s) | Orb 500(%) | BRISK(s) | BRISK(%) | AKAZE(s) | AKAZE(%) |
| Total TPF | 0.043 | 100% | 0.19 | 100% | 0.21 | 100% |
| DAC TPF | 0.021 | 49.7% | 0.061 | 32.3% | 0.17 | 78.7% |
| Match TPF | 0.0052 | 12.2% | 0.11 | 56.2% | 0.026 | 12.3% |
| Homo TPF | 0.0013 | 3.1% | 0.005 | 2.4% | 0.003 | 1.3% |
| Proj TPF | 0.0044 | 10.4% | 0.006 | 3.15% | 0.0056 | 2.6% |
| Write TPF | 0.010 | 24.5% | 0.012 | 6.0% | 0.011 | 5.0% |

Table 2: Avg time per frame for each component in absolute seconds and proportion of total time per frame based on the *room* video. TPF: Time per frame. DAC: Detect and Compute keypoints. Match: Matching time. Homo: Homography computation. Proj: Projection. Write: Write into the video.

It appears that there are varying results across the 3 algorithms used. Unsurprisingly, the detect and computation component took a large component, in fact the highest except in BRISK. In BRISK, the matching component took a way longer time as compared to the other 2. Computing homography and projection of image are consistently low across all 3 except in ORB which is an anomaly. A similar pattern is also seen for the writing image into the video generator.

### 3.3 Other variations analysis

We will explore 2 other variation that could be used.

First is the usage of Lowe's ratio test instead of using *crossCheck*. These 2 variations are tested on one video and they are found to have very similar speed and quality performance.

Next, using the ANMS as an additional step after matching doesn't seem to yield better results. This was tested when ORB algorithm is used to further improve its performance since BRISK and AKAZE are doing well and have a high runtime already. Yet, using ANMS with a selection of 300 features on the *room* video resulted in the time taken to be tenfold and not of any higher quality. This was then omitted from the output videos produced.

### 3.4 Additional findings

As this implementation fundamentally relied on features detection and descriptors computation, we would require videos that show scenes that can display plenty of such features with strong corner strengths. Without sufficient distinctive keypoints, no matter what algorithm is used, it would be hard to produce a high quality video. Videos that are taken of an empty surface without sufficient keypoints have performed badly and are not shown. Hence, the videos taken have surrounding objects so that they can produce different distinctive keypoints for detection by the algorithm. In addition, it is hard if movement of the camera is fast and/or jerky because this could result in aliasing (i.e. not enough frames are captured) which would hinder accurate computation of homography. This will be shown in the video presentation.

## 4 Future improvements

Going forward, there are several methods that can be used to improve the performance in terms of speed and/or quality further. There are certain trivial methods such as using a machine with higher processing speed, taking "better" videos such as using a tripod to keep motions more stable, rewriting the code in C++ (which tends to provide a boost up as compared to Python code) and optimizing the code as much as possible by vectorizing and reducing for loops whenever possible.

Another possible method would be to experiment with other ANMS methods - Range Tree ANMS, K-d Tree ANMS and Suppression via Square Covering, which are proposed in this paper[13]. These ANMS methods provide very evenly distributed keypoints and offer a much faster speed up compared to the Brown's original ANMS method. The original ANMS method is slow as it consists of 2 for loops through the entire set of keypoints. These methods are worth trying and if it works well with ORB (which have a much faster detect and compute time), then the overall time taken may be shorter than BRISK and AKAZE and yet still achieving a comparable quality.

Even with more improvements, it will probably still pale in comparison to the state-of-the-art (SOTA) methods out there. Highly successful SOTA methods have been used in products such as the mobile app Pokemon Go and Apple's Augmented Reality Kit (ARKit). However, it utilizes device motion information as well to create AR experiences. This should not be surprising as having more information would definitely help, because after all searching for keypoints and computing descriptors are also utilizing information at hand. [14] talks about hybrid tracking which is the utilization of a combination of imagery information and sensory motion information to create AR.

The output videos for this project may be found at:

- Room: `https://youtu.be/cRIK4aO6PDk`
- Living: `https://youtu.be/yH-kGlZrD-s`
- Wall: `https://youtu.be/lkiryMUmHpo`

## References


[1] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE, sep 2008.

[2] G. Simon, A.W. Fitzgibbon, and A. Zisserman. Markerless tracking using planar structures in the scene. 2000.

[3] M. Brown, R. Szeliski, and S. Winder. Multi-image matching using multi-scale oriented patches. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR05)*. IEEE.

[4] David G Lowe et al. Object recognition from local scale-invariant features. In *iccv*, volume 99, pages 1150–1157, 1999.

[5] Shaharyar Ahmed Khan Tareen and Zahra Saleem. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. IEEE, Mar 2018.

[6] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*. IEEE, nov 2011.

[7] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[8] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[9] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *2011 International Conference on Computer Vision*. IEEE, Nov 2011.

[10] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *European Conference on Computer Vision (ECCV'10)*, September 2010.

[11] P. F. Alcantarilla, J. Nuevo, and A. Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In *British Machine Vision Conf. (BMVC)*, 2013.

[12] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. KAZE features. In *Eur. Conf. on Computer Vision (ECCV)*, 2012.

[13] Oleksandr Bailo, Francois Rameau, Kyungdon Joo, Jinsun Park, Oleksandr Bogdan, and In So Kweon. Efficient adaptive non-maximal suppression algorithms for homogeneous spatial keypoint distribution. *Pattern Recognition Letters*, 106:53–60, apr 2018.

[14] Deepti Prit Kaur and Archana Mantri. Computer vision and sensor fusion for efficient hybrid tracking in augmented reality systems. In *2015 IEEE 3rd International Conference on MOOCs, Innovation and Technology in Education (MITE)*. IEEE, oct 2015.