Computer Vision Based Augmented Reality Effect

F. Xia[1], Y. Cui[1]

***Abstract*— In this work, we use a computer vision based pose estimation algorithm for real-time tracking of a textured flat surface. The algorithm is applied with a 3D model loader to produce an augmented reality effect. The system is implemented with OpenCV and OpenGL using Python for functionality demo. A Java based Android implementation for pose estimation is also conducted for feasibility analysis.**

I. INTRODUCTION

The goal of augmented reality (AR) is to insert virtual information in the real world providing the end-user with additional knowledge about the scene. The added information, such as virtual objects, must be precisely aligned with the real world.

Conventional augmented reality applications utilize sensor fusion of on-board IMU and camera input for overlay of artificial objects onto the footage taken by a hand held device such as a cell phone or tablet. AprilTags are often used for determination of surface pose with regard to the camera, as shown in Fig. 1. However, since AprilTags need to be attached to the tracked object, sometimes it can be inconvenient.

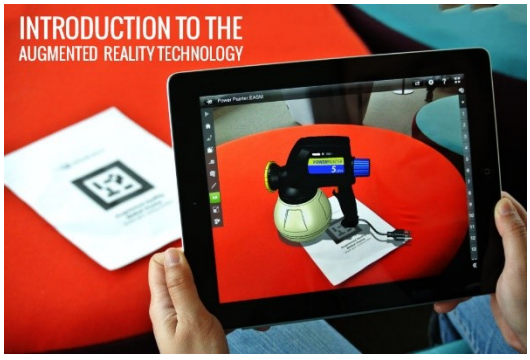

Fig. 1. Overlay of a virtual object on an AprilTag using a tablet [1].

In this project, the focus is to utilize computer vision techniques for implementation of a particular augmented reality case. The virtual object (a 3D Pikachu) will be overlaid on an ordinary object that has a flat textured surface without using AprilTags.

II. SYSTEM OVERVIEW

The overall goal of the project is to track a 2D textured surface and find its pose in terms of translation and rotation. By applying proper transformation to the 3D model in the world frame of the rendered scene, the virtual object is projected onto the tracked object. Thus, an augmented reality effect can be realized.

The system diagram of the computer vision based augmented reality system implemented in this project is shown in Fig. 2. This diagram illustrates a single cycle of the program that is running for each input frame. Before the process begins, we take an image of the textured 2D surface to be tracked and use it as the "target image input" in the diagram. Together with the video frame input, we start the process for pose estimation using the OpenCV library. The input video frame is also utilized as the background texture. The Pikachu 3D model is rendered and projected on the final scene for visualization using OpenGL library. In the following sections, we give a detailed illustration about each component in this process.

III. COMPUTER VISION FUNCTIONS

The first step in computer vision pose estimation section is to use the Oriented FAST and Rotated BRIEF (ORB) detector to extract features that are good for tracking and comparison later in the two images. The second step is to use the Fast Approximate Nearest Neighbor Search Library (FLANN) to conduct feature point matching between the two sets of feature descriptors. The third step is to use the RANdom SAmple Consensus (RANSAC) based algorithm for homography matrix estimation. The fourth step is to apply corresponding matching points with homography matrix and utilize the SolvePNP function in OpenCV to obtain the translation and rotation vector estimation. Finally, a median filter is applied to the estimated rotation and translation to reduce the jiggling of the overlaid 3D model as the vectors are directly used by the OpenGL functions for rendering.

A. Feature Detection

Feature detection is a process for identifying key positions such as sharp corners where rich texture information can be extracted. There are various methods for feature detection that can be useful for extracting the feature points from our input frames. Algorithms such as Scale-Invariant Feature Transform (SIFT) and Speeded Up Robust Features (SURF) have good performance but can be computationally expensive. In our case, we chose to use the Oriented FAST and Rotated BRIEF (ORB) detector for its comparative performance with fast detection speed, which is important for implementation on a system with limited resource for real time applications. In addition, the ORB detector is open source while the SIFT and SURF are both pattern protected.

[1] F. Xia and Y. Cui is with the Mechatronics Research Laboratory at Massachusetts Institute of Technology, 02139, Cambridge, MA, USA. For questions, please email: xiafz@mit.edu

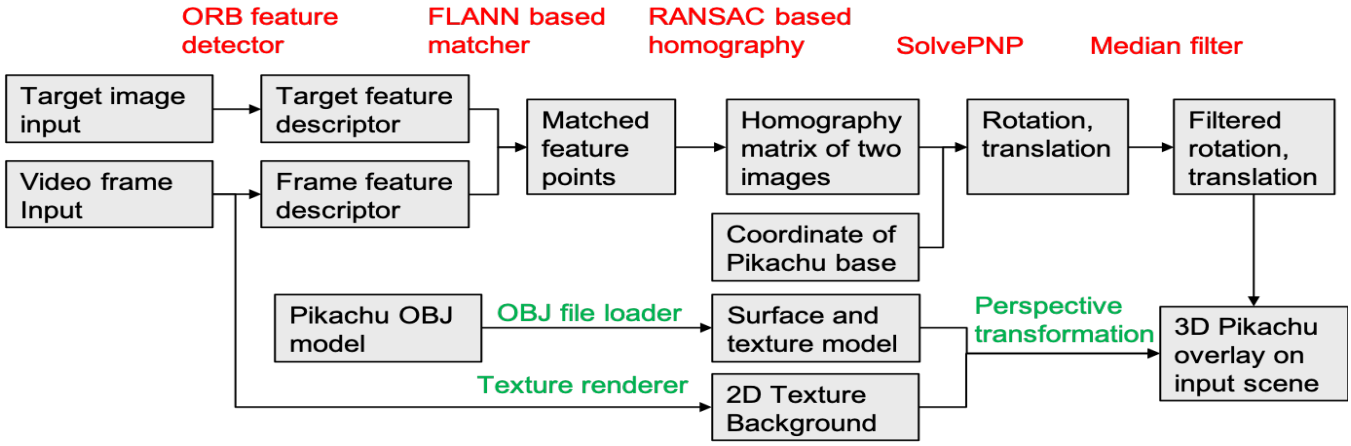

Fig. 2. System Overview

As the name suggested, ORB feature detection is composed of a combination of Features from Accelerated Segment Test (FAST) feature detector and Binary Robust Independent Elementary Features (BRIEF) feature descriptors. The FAST feature detector provides a good method for feature detection that is suitable for real-time application on system with limited resources for its computational efficiency. On the other hand, the BRIEF feature descriptor provides a more compact representation of the features detected to save storage spaces.

In principle, the FAST feature detector works by comparing the pixel intensity values of a point of interest (denoted as p in Figure 3) to the 16 pixel values forming a circular pattern around the pixel of interest (denoted with number labels in Figure 3). The test is performed by comparing the absolute difference d_i of the intensity values between point p and the labeled 16 points. A threshold value of T and number of points N (commonly chosen as 12) should be set so that a corner is detected if the number of labeled points satisfying $|d_i| > t$ is no less than N. Higher-speed test can also be performed initially on pixels labeled as $2, 6, 10, 14$ in Figure 3 with 2 and 10 checked first. At least 3 out of the 4 points tested should have $|d_i| > t$ before we continue to test more points so that we can save resources when compute the rest of the pixels.

For feature descriptors, conventional SIFT and SURF utilize 512 bytes or 256 bytes with floating number respectively due to their high dimensions. Compression algorithms such as hashing or principal component analysis are utilized to reduce size of storage. The BRIEF feature descriptor provides a shortcut to find compressed binary strings directly without finding the descriptors in the first place.

To improve the rotational performance of feature detection, the ORB algorithm added the notion of orientation by computing the weighted center of intensity. The vector from the center of the patch p to the weighted center of intensity is utilized as orientation and used to rotate the BRIEF descriptor around for improved rotational performance. More information regarding the ORB feature detection algorithm

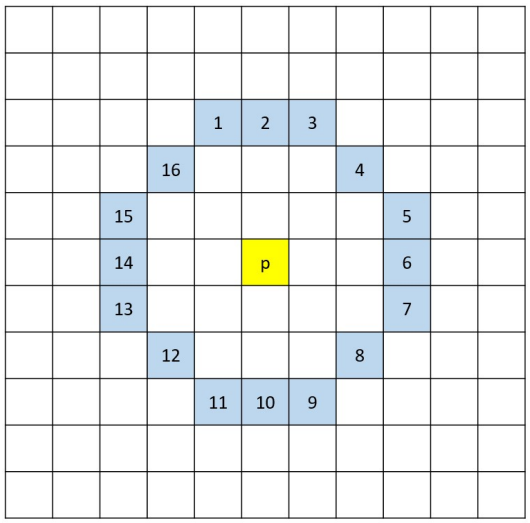

Fig. 3. FAST corner detection algorithm

can be found on the OpenCV documentation and in the reference papers [2, 3, 4].

B. Feature Matching

Fast Library for Aproximated Nearest Neighbor (FLANN) is utilized to compute the Hamming distance of the ORB descriptors, where a count of number of different elements in the binary string is computed similar to the XOR operation. As a number of potential matches can be found for a single feature, a selection needs to be performed to find strong matches without ambiguity. This is done by comparing the first and second smallest computed distance measurement and a good match is found if the distance ratio is less than 0.3, which indicates that the match has good correspondences.

C. Homography Matrix Estimation

For two images of a plane viewed from different orientations in 3D, a 3×3 homography matrix H that transforms the pixel locations from one scene to another can be estimated.

In order to compute the homography, at least 4 pairs of corresponding points are needed. In practice, more corresponding points can be identified from feature matcher to form an over-determined system where a least squares approach can be utilized to obtain the optimal parameter estimation.

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{H} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \quad (1)$$

In this case, we utilize the RANdom SAmple Consensus (RANSAC) algorithm for estimation of homography matrix parameters. Being an iterative approach, the RANSAC approach is more robust to outliers than conventional least square approaches that utilizes all the data points. This is done by fitting least square models to multiple randomly selected subset of the data and return the model with the best fitting. As the inliners tend to provide better fit than the outliers, the model we picked would have a better performance than using all the data points that include outliers.

D. Pose Estimation

In order to create a 3D model overlay, we need to identify the rotation and translation of the tracked object and apply the movement with proper scaling to the loaded 3D model.

The first approach is to decompose the homography matrix estimation to obtain the translation and rotation as illustrated in [5]. However, this method can produce four possible solutions and needs further constraints to identify the real translation and rotation.

Another approach is to utilize the solvePnP function to find the object pose from 3D to 2D point correspondence. As a minimum of 4 corresponding points are needed for estimation, we assume the rectangular base of the 3D Pikachu model coincides with that of the target image and have height $Z = 0$. The four corners of the base of the 3D model give 4 points in 3D world. To obtain the 2D points in the image, we apply the homography matrix estimated previously to the target images and find the 4 corners of the 2D surface. The translation and rotation vectors can then be estimated with the 4 point correspondences.

Notice that we can also conduct homography matrix estimation directly using the solvePnPRansac function with corresponding features detected on the images from the feature matcher. This would skip the step of homography matrix estimation but put more computation work on the pose estimation.

E. Noise Reduction

The parameters estimated with the video frame input can be noisy and variations in the translation and rotation vectors cause jiggling of the 3D model overlay. For practical purpose, we can apply a 10 points median filter to the estimated vectors to smooth out the motion of the model. This non-linear filtering provides a good result from the demonstration as significant outliers are eliminated.

IV. Visualization

In order to create an augmented reality effect, the camera video input is utilized as the far clipping background in the scene. A 3D model of Pikachu is then loaded into the scene and overlaid on the video input.

A. Scene Setup

As shown in Fig. 4, the far clipped is entirely filled with the video stream input. The axis orientation setup for prospective projection in OpenGL framework is also demonstrated.

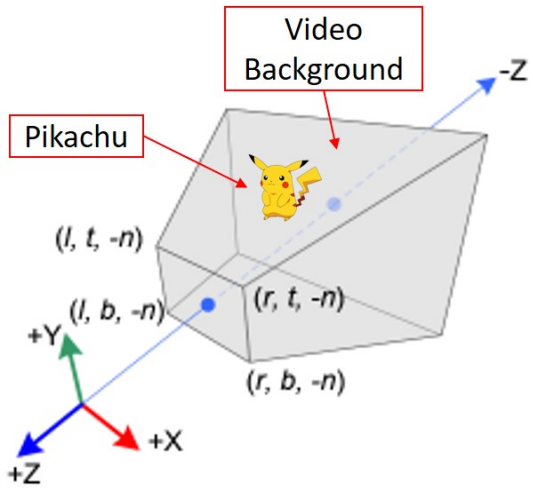

Fig. 4. OpenGL scene setup for Pikachu AR effect.

B. Object Loading and Positioning

A Pikachu 3D model in the format of OBJ file with texture information is loaded into the scene with an object loader. Translation and rotation estimated previously are applied to the model with proper scaling factor to create the overlay effect.

V. Software Implementation

As briefly mentioned in previous sections, OpenCV and OpenGL libraries are chosen to implement the AR effect for their well established built-in functions and cross platform package portability. This allows the original C++ code to be used both in Java and Python.

VI. Python Implementation

The Python programming language is chosen for PC implementation due to its convenience in coding. In addition to Python version of OpenGL and OpenCV, packages like Numpy, Pillow and Pygame are utilized for vector manipulation, image manipulation and visualization. An OBJ file loader from Pygame open source software is also utilized for loading the downloaded Pikachu 3D model. A full implementation of the aforementioned AR functionality is done in Python for demonstration.

VII. Java Implementation

Java is used for Android Studio implementation as it is the fundamentally supported language. Notice that NDK can be installed also to enable direct C++ programming with the OpenCV and OpenGL packages. In addition, the OpenCV Manager APK needs to be installed on the phone that will run the App. For Android, only pose estimation is implemented for the feasibility of demonstration.

VIII. Results and Discussion

The demonstration of the code functionality is done by recording videos of the Pikachu AR effect using both OpenGL and OpenCV as shown in Fig. 5 (a) and the simple rectangle overlay done purely in OpenCV as shown in Fig. 5 (b) and (c).

Fig. 5. Augmented reality effect demonstration (a) Pikachu AR in Python (b) rectangle overlay in Python (c) rectangle overlay in Java on Galaxy S4 Android phone

A. Python PC Demonstration

The Python implementation can run at a speed of around 15 to 20 FPS on an PC with i7 core and 32GB RAM. Although it is less than the standard 25 FPS video rate, the tracking effect is reasonably good for practical purpose.

B. Android Studio Demonstration

As shown in Fig. 5 (c), the Android rectangle overlay is done by tracking a selected previously taken target image in the camera image library. With the limiting processing power of Samsung Galaxy 4, the frame rate is around 3 to 4 frames per second for image processing.

IX. Conclusion

In this project, we implemented the computer vision algorithm for planar textured object tracking and pose estimation using OpenCV in both Python and Android studio. An augmented reality effect with Pikachu 3D model overlay using OpenGL in Python is also implemented for functionality demonstration.

X. Future Improvement

For Python implementation of the AR effect, additional functionality for multiple object tracking and pose estimation can be an interesting application. It can be challenging with objects of similar texture patterns as the feature matcher can be confused. As an initial thought, combining machine learning based technique such as convolutional neural networks for object detection with the pose estimation algorithms can be helpful as suggested in [6]. For implementation purpose, rewriting the current code structure into object oriented style can be helpful than the currently procedure based code.

For testing performance of Android implementation, we can benefit from using a newer generation of Android cell phone for improved frame rate. Implementation of additional camera calibration routine would also be helpful to improve the estimation accuracy. Further development of applications that utilize the estimated translation and rotation for not only AR effect but also panorama stitching and other potential applications can also be interesting.

ACKNOWLEDGMENT

The author would like to acknowledge the help from documentation of OpenCV and OpenGL official site.

References

[1] *AprilTags AR Effect*. `https://blog.sagipl.com/introduction-augmented-reality-technology/`. Accessed: 2018-11-30.

[2] E. Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2564–2571.

[3] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features". In: *Proceedings of the 11th European Conference on Computer Vision: Part IV*. ECCV'10. Heraklion, Crete, Greece: Springer-Verlag, 2010, pp. 778–792. ISBN: 3-642-15560-X, 978-3-642-15560-4.

[4] Edward Rosten and Tom Drummond. "Machine Learning for High-Speed Corner Detection". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443.

[5] Ezio Malis and Manuel Vargas. *Deeper understanding of the homography decomposition for vision-based control*. Research Report RR-6303. INRIA, 2007, p. 90.

[6] Paul Wohlhart and Vincent Lepetit. "Learning Descriptors for Object Recognition and 3D Pose Estimation". In: *CoRR* abs/1502.05908 (2015). arXiv: `1502.05908`.