Real-Time Mobile Augmented Reality Using Markerless Subject Tracking

Robert Mahieu
Department of Electrical Engineering
Stanford University
Stanford, CA 94305 USA
rmahieu@stanford.edu

Hershed Tilak
Department of Electrical Engineering
Stanford University
Stanford, CA 94305 USA
htilak@stanford.edu

***Abstract*—A real-time markerless subject tracking system for projecting 2D imagery into an augmented reality space is proposed using shape and color features to accomplish high computational efficiency and robust detection. The system is implemented on Android devices and is able to run consistently at a speed of around 30FPS. Results suggest that the algorithm is reasonably robust to variation in viewing distance and angle.**

I. Introduction

The field of augmented reality (AR), especially in recent years, has become a prominent topic of interest in the public eye as techniques have become more accurate and effective. As a result, consumer market interest has grown substantially. Research into AR shows many promising applications in fields ranging from medical [5] to industrial manufacturing [2]. With these prospective applications in mind, new AR technology needs to be adaptable to varied environments with minimal user interaction.

The fundamental operation of any augmented reality system is based on its ability to track camera pose through estimation using some form of visual cues. Early work into this problem primarily focused on utilizing pre-defined markers, placed in the camera frame by the user, to track movement, while more recent efforts into markerless tracking involve automatically detecting natural features and matching them over successive frames [8]. However, each method has its own disadvantages: marker-based systems require some amount of user interaction with the environment, while markerless systems typically require higher amounts of processing to perform real-time feature detection and matching with comparable accuracy. While most current research lies within the domain of creating efficient markerless methods, in practice, marker-based methods seem to be more abundant due to their reliability.

This paper suggests an effective method of projecting 2D imagery into a scene using a lightweight markerless tracking method that makes use of very specific features of the target surface. Through an understanding of the shape and color of the subject of interest (here the scope is limited to rectangular surfaces), it will be shown that this operation can be completed in real-time using standard Android hardware.

II. Previous work

A. Marker-based Tracking

Early marker-based systems include HOM, IGD, SCR, and perhaps the most commonly used ARToolKit (ATK), which have since been compared and evaluated by Zhang et al [7]. More recent marker systems have utilized QR codes to assist in tracking and to store links to the specific 3D data that is to be superimposed in the scene [4].

B. Markerless Feature-based Tracking

Some of the most popular markerless tracking systems utilize methods such as object shape contour detection combined with edge tracking [1] or texture feature detection and edge tracking [6].

Markerless systems can generally be classified into two categories: corner detectors and blob detectors, depending on what sorts of feature points are desired. Corner detectors of course look for corners that exist within the image, while blob detectors generally look for shapes [3]. The success of a markerless tracking system can be assessed based on its computation time, robustness with respect to variation in lighting and blurring, robustness to view angle variation, and scale invariance. Most research unfortunately accomplishes only one of either computational efficiency or detection robustness, not both.

Through a combination of corner and blob detection methods, the technique proposed by this paper is able to accomplish both high computational speed and robust detection.

III. System flow

A. System Initialization

Before the algorithm can run, two pieces of information need to be collected from the user. The first piece of information is the image that the user wants to project. We allow the user to choose any photo currently saved to their phone. After the image is selected, we subsample it to account for the memory constraints of the mobile device and then load it into RAM. The second piece of information is the subject onto which the stored image will be projected. We allow the user to select the subject via touch input and store the location and RGBA color data of the pixel that the user touches.

The second step of the system initialization phase is to apply a color-balancing algorithm to the user image in the HSV color space so that the user image will look more natural when projected into the scene. The color-balancing algorithm first calculates the range of the value and saturation channels for the camera frame, then shifts and scales the value and saturation channels of the user image to cover the same range. The algorithm is re-run every time the user selects a new subject to project onto.

B. *Frame Update*

After initialization has been completed, each time the program receives a new frame from the device camera, the following steps are taken:

1) Color Thresholding: The input camera frame is thresholded using the RGB color data of the subject that was stored during the initialization phase. A percentage based threshold of plus or minus 50% is applied to each color channel of the input camera frame. The ouput is a frame where all pixels that were within the specified range are set to white and all other pixels are set to black.

2) Isolate Connected Component: A connected components algorithm is run on the thresholded camera frame in order to isolate the subject from other objects in the scene that may also lie within the color threshold range. This function returns only the connected component which contains the point in the frame that the user had selected during the initialization phase. The algorithm takes the complement of the binary image and, starting at the user-selected location, swaps surrounding zeros for ones until it reaches the ones that form its region boundary. It then returns a binary image containing only the pixels which have been altered. The output is a black frame which contains a single, connected region of white pixels.

3) Low Pass Filter: The frame is filtered with a 5 pixel by 5 pixel Gaussian filter with a 20 pixel standard deviation. This smooths out small deformations in the subject and primes it for detection which occurs in the next step.

4) Harris Corner Detection: A Harris corner detector is applied to the resulting image. The returned corners are ordered based on corner strength as determined by the detector. The location of the Harris corner with the largest strength is selected as the first corner and all other returned Harris corners within a 10 pixel radius of that location are zeroed out to avoid repeat detection of the same corner. This process is repeated until all four corners have been identified. To determine which points correspond to which corner of the subject, the points are first separated into left and right corners based on each of their x-coordinate's relation to the median of the four points' x-coordinates. These subsets are then separated into top and bottom corners by simply comparing which point has the larger/smaller y-coordinate. Thus at the end of this step, we have the four corners and their relative position on the subject.

5) Update Subject Location: The four Harris corners determined in the previous step are used to estimate the size and location of the subject in the current frame. The width is taken to be the pixel distance between the leftmost and rightmost Harris corners, the height is taken to be the pixel distance between the topmost and bottommost Harris corners, and the location is taken to be the center of the four Harris corners. If the estimated subject location is within one estimated subject size of the previously stored subject location, the stored subject location is updated. If not, the new location estimate is disregarded.

6) Compute Homography: Using the coordinates of the four stored Harris corners and the corresponding four corners of the user image, a projective homography transform is calcuated between the user input image and the subject onto which we want to project.

7) Overlay Image: The user image is transformed using the homography matrix. The resulting output image is overlain on the camera scene, with the binary image resulting from step 2 used as a mask.

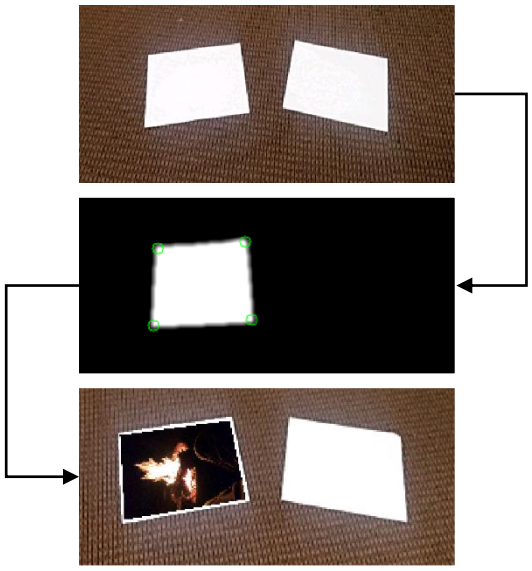

Fig. 1. Algorithm pipeline: (top) camera frame, (center) thresholded, isolated component, with corners detected, (bottom) user image overlaid on camera frame.

The steps taken by the algorithm are summarized above in Figure 1. The current frame is taken from the camera, then the subject of interest is thresholded, isolated, and sent through a Harris corner detector. Using the resulting corners, the user-selected image is transformed and drawn onto the camera frame.

IV. RESULTS

A selection of experimental outputs are shown below in Figure 2. It can be seen that the system is successful for a variety of angles and distances.

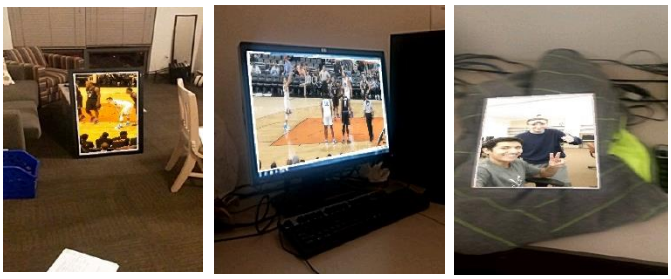

Fig. 2. Select examples of system output in varying environemnts.

The perceived quality of our system is directly related to the robustness of our Harris corner detection. To evaluate this quantity we created the following metric:

$$R = \frac{c_4 - c_5}{c_1} \quad (1)$$

Where c_i is the strength of the i^{th} strongest corner as determined by the Harris corner detector. This metric tells us the difference between the fourth and fifth strongest Harris corners, which provides a measure of the ambiguity in our detected corners. As this quantity decreases, the chance of our system returning a false corner on the next input frame increases. Our metric is normalized by the strength of the strongest Harris corner. This penalizes situations where the strength of the fourth strongest Harris corner is much weaker than the first. Ideally we want our system to return four strong Harris corners of near equal magnitude.

Figure 3 shows the value *R* as a function of angle for our system. The angle is reported with respect to the surface normal. All measurements were made at a distance of 36in from the center of the subject. The maximum angle for which the system was able to successfully recognize the object and discern four distinct corners was at an angle of 85 degrees. Figure 3 shows that the robustness of our detected Harris corners decreases as the angle from the surface normal increases. Subjectively, the system appeared to be reasonably stable up to about the maximum angle of 85 degrees, at which point the zeroing of the local corner areas, from Step 4 of the system flow, overlapped other true corners, making them undetectable. It was also noted that the subject location update step of the system flow becomes unreliable at this extreme angle due to the small area of the thresholded component.

Figure 4 shows the value *R* as a function of distance. All measurements were made at an angle of 0 degrees from the surface normal. Our measurements show that the detected corners are reasonably robust to changes in distance from the subject.

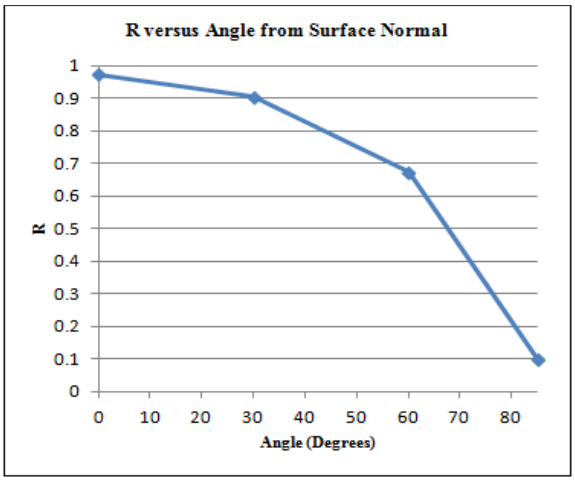

Fig. 3. Robustness of system against variation in angle to subject

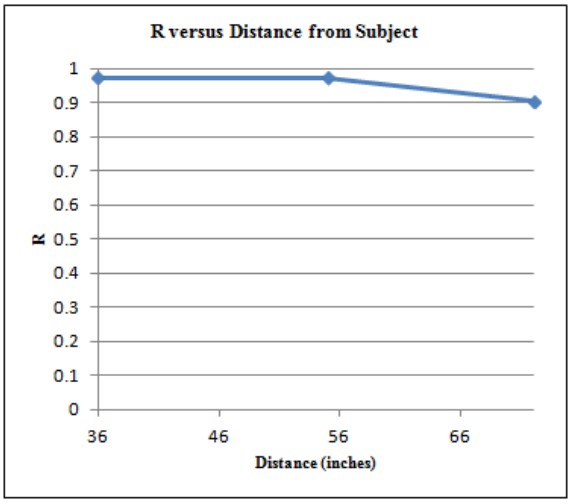

Fig. 4. Robustness of system against variation in distance from subject

V. Conclusion & Future Work

The system performed well in environments where there was a strong color contrast between the subject of interest and its immediate surrounding area. We were able to track the rectangular subject in various environments and successfully project a user image with correct perspective. Using a Samsung Galaxy S6 phone for testing, the Android application, developed using the OpenCV mobile library, was remarkably quick when running on x64 hardware and was able to maintain a frame rate of around 30FPS (frames per second) during all processing.

In the future, improvements could be made to the color balancing algorithm to make the projected image look more natural in its surrounding environment. The HSV matching function implemented here was relatively basic and was not particularly successful in most cases. Due to memory issues, this matching is also only able to run when the user makes a touch selection. Additionally, this algorithm is by no means robust to occlusion, so it would be beneficial to explore methods for handling these cases. This would enable further

augmented reality applications where the user needs to interact with the subject of interest while an image is being projected.

REFERENCES

[1] A. I. Comport, E. Marchand, and F. Chaumette. A Real-Time Tracker or Markerless Augmented Reality. In International Symposium on Mixed and Augmented Reality, Tokyo, Japan, September 2003.

[2] M. Fiorentino, R. Amicis, G. Monno, and A. Stork, "Spacedesign: A Mixed Reality Workspace for Aesthetic Industrial Design," in Proc. of International Symposium on Mixed and Augmented Reality, pp. 86‐94, 2002.

[3] B. Furht, Handbook of augmented reality. New York: Springer, 2011.

[4] T. Kan, C. Teng, W. Chou, "Applying QR code in augmented reality applications," 8th Int'l Conf. on Virtual Reality Continuum and its Applications in Industry, 2008.

[5] N. Navab, T. Blum, L. Wang, A. Okur and T. Wendler, "First Deployments of Augmented Reality in Operating Rooms", Computer, vol. 45, no. 7, pp. 48-55, 2012.

[6] L. Vacchetti, V. Lepetit, and P. Fua. Combining edge and texture information for real-time accurate 3d camera tracking. In Proc. ISMAR 2004, Arlington, VA, USA, November 2‐5 2004. IEEE

[7] X. Zhang, S. Fronz and N. Navab. Visual marker detection and decoding in AR systems: a comparative study. In ISMAR '02, pp. 97-106, 2002.

[8] F. Zhou, H. B.-L. Duh, and M. Billinghurst, "Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR," in Proc. of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, pp. 193‐202, 2008.

APPENDIX

The work for this project was distributed equally:

Hershed – Co-developed algorithm, laid groundwork for algorithm through Harris corner detection. Debugged final code. Wrote report.

Robert – Co-developed algorithm, laid groundwork for algorithm from Harris corner detection through system output. Debugged final code. Wrote report.