A Markerless Augmented Reality System for Mobile Devices

Alex Ufkes
Ryerson University
Department of Computer Science
Toronto, Canada
aufkes@ryerson.ca

Mark Fiala
Ryerson University
Department of Computer Science
Toronto, Canada
mark.fiala@ryerson.ca

***Abstract*—Augmented Reality (AR) combines a live view of a real world environment with computer-generated virtual content. An AR system poses unique challenges including requiring a high quality camera pose estimate and operating on resource-limited platforms. We present a full system based on a hybrid approach using ORB binary features and optic flow that is able to run on a consumer tablet device in near real time. The system uses an adaptive map of features designed to extend the usable range possible with ORB features. We demonstrate how real time performance is possible with platform specific optimizations, and propose the use of two different existing pose estimation algorithms to further improve speed and extend the usable tracking range.**

Keywords-Augmented Reality; Markerless; Mobile; Real Time; Feature Detectors; ORB; Hybrid Tracking

I. INTRODUCTION

Augmented Reality (AR) is a sub-class of Virtual Reality (VR). In true virtual reality applications, all sensory input experienced by the user is simulated. In augmented reality, virtual content is combined with real world sensory input in order to create a blended version of the environment. This is most commonly achieved by using computer vision techniques to analyze live video input from a hand-held camera or Head-Mounted Display (HMD) in order to determine the location and orientation (pose) of the camera in 3D space. Virtual augmentations can then be inserted seamlessly into the real-world video feed using this pose information.

Traditional AR systems such as ARTag [1] utilize fiducial markers or "tags", over which virtual objects are rendered. Marker-based systems are ideal for applications involving a static or fixed environment, or situations where the desired virtual space does not extend beyond the tag itself. In the case of simple applications, users must print and carry the tags in order to use the AR application. For more complex, large scale applications, significant infrastructure is required in the form of dozens or hundreds of markers placed throughout the environment [2].

For more ubiquitous and ad-hoc applications, *markerless* AR can be used. Instead of relying on artificial markers, markerless AR systems make use of natural features already present in the environment. The biggest challenge of performing markerless AR is overcoming the daunting computational requirements of camera pose estimation through natural feature detection and matching. This issue is even more critical in handheld mobile devices[1], where processor speeds and memory are limited.

[1] Handheld mobile devices include smartphones, tablets, PDAs, etc. Laptop computers are not considered handheld in the context of this paper.

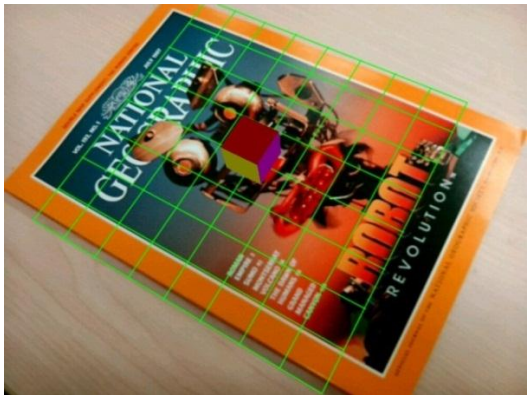

Figure 1 - Sample output of our mobile AR system recognizing a scene and drawing simple augmentation overlays (grid lines and cube).

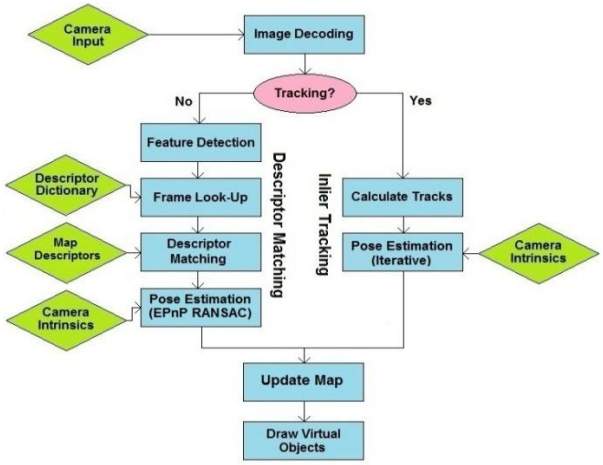

Figure 2 - AR pipeline overview.

In this paper, we present an end-to-end markerless AR pipeline based on the binary ORB descriptor by Rublee et al [3]. The system is capable of real time performance on current generation consumer grade mobile devices. The system requires nothing more than a calibrated monocular camera; every other process is performed online and in real time, including map creation. The computational restrictions of mobile devices are addressed at each stage in the pipeline, and an Android implementation is used to evaluate the system as a whole. Figure 1 shows our system recognizing a magazine cover and drawing the X-Y plane and a solid 3D cube.

Contributions

This work is primarily a proof-of-concept that robust markerless AR can be performed in real-time (10-15Hz) on

978-0-7695-4983-5/13 $26.00 © 2013 IEEE
DOI 10.1109/CRV.2013.51

modern consumer mobile hardware. To accomplish this, we make the following improvements to the traditional AR pipeline:

- Hardware SIMD implementations of image decoding and descriptor matching that yield an approximate threefold increase in efficiency in both cases.
- Adaptive pose estimation based on whether the system is matching or tracking in order to balance execution speed and pose accuracy. *Hybrid* systems (using both feature detection and tracking) usually have the same pose determination algorithm in both modes.
- Implementation of a binary bag-of-features scheme for recognizing multiple scenes to augment.

II. Related Work

There has been myriad work done in the field of AR over the past decade. In this paper we focus mainly on work related to markerless AR, mobile AR, and their relevant computer vision algorithms.

The best known work in monocular camera localization and mapping is the MonoSLAM system developed by Davison et al [4]. The authors successfully applied SLAM methodologies to the vision domain using a monocular camera as the sole input device. MonoSLAM maintains a sparse map of oriented planar textures (~12 for any given camera view) centered on Shi-Tomasi corners. The system is initialized using a planar target containing four known features before dynamically updating the map. They present results in the AR domain, and achieve real-time operation at 30Hz while rendering augmentations.

The Parallel Tracking and Mapping (PTAM) system, introduced by Klein and Murray in 2007 [5], uses a multi-threaded approach in order to simultaneously track interest points by morphing and matching image patches, and maintain a map of these patches. PTAM uses a stereo pair of images of a planar scene for initialization, and uses bundle adjustment to determine and update the 3D locations of interest points.

Taehee and Hollerer also propose a multi-threaded approach in [6]. They use a hybrid tracking method which extracts SIFT features [7] (instead of image patches) and then tracks them using optic flow. They achieved real-time performance by only extracting and matching SIFT features periodically in a thread separate from the tracker. They also perform scene recognition by recognizing previously recorded SIFT features.

On the mobile front, attempts have been made to adapt SIFT and its speedier counterpart SURF [8] to mobile devices. Wagner et al proposed a hybrid between FAST corners [9] and a reduced version of the SIFT descriptor in [10]. Rather than compute descriptors every frame, they also adopt a hybrid approach and track existing features using SAD patch correlation. This method achieves upwards of 20Hz while extracting ~150 features per frame (320x240). Chen et al propose a modified version of SURF in [11] that achieves a roughly 30% speed-up over the original SURF algorithm, but nevertheless falls far short of real-time operation on mobile devices.

The first self-contained AR system to run on a consumer-grade cell phone was presented in 2004 by Mohring et al [12]. Their system recognizes different markers via circular bar codes and detecting gradient changes in the red, green, and blue color channels. Their entire pipeline achieved a frame rate of 4-5fps, at a camera resolution of 160x120.

A reduced version of the PTAM system has been adapted to the iPhone [13], but results showed severely reduced accuracy and execution speed. PTAM is intended for use in small AR workspaces, and suffers reduced performance as the map gets bigger and bigger due to the bundle adjustment process being cubic with respect to the number of features in the map ($O(n^3)$).

As a continuation of their work in [10], Wagner et al propose a more complete version of their feature detection and matching system in [14]. They use a similarly modified version of SIFT and outlier rejection that runs at approximately 26Hz. However, when AR-related overhead (image retrieval from camera, rendering, etc.) is taken into consideration, the speed drops to 15Hz. They combine this with patch tracking to greatly improve speed to 8ms per frame (not including AR overhead) instead of using SIFT on every frame. Their system runs on 320x240 imagery during the SIFT phase, and down-samples further to 160x120 while tracking. In addition, a maximum of 100 features are tracked at any given time.

To the best of our knowledge, there exists no complete AR system that incorporates all the elements presented in this paper while achieving real-time operation on mobile devices. Most notably absent from existing systems are online map creation and multiple map support.

III. Technical Approach

This section describes our system. The stages of the AR pipeline shown in Figure 2 are discussed. For each stage we discuss which algorithms are used, and any modifications that were made to improve efficiency. Quantitative results of these speedups are presented in Section IV.

Map Overview

We define a *'map'* as a list of feature descriptors and their corresponding 3D world coordinates. In our system, matching features in an input video frame to a map is accelerated by using a tree structure. When a map is created, the descriptors are placed into a clustering tree with eight branches as done by Muja and Lowe in [15]. Each map has an associated 3D bounding box consisting of four world coordinates. This bounding box can be projected into the image frame once camera pose has been estimated. Additionally, the system maintains a 100-word descriptor vocabulary. Each map is matched to this vocabulary to create unique response histograms.

Platform Specific Acceleration

To achieve real time speeds on a resource limited mobile devices, platform specific adaptations are used. This system makes use of the NEON instruction set, which utilizes the

SIMD engine present in the ARM Cortex-A series of processors[2]. This is achieved through the use of 'intrinsic' functions added to the C/C++ code.

Processing Stages

Our system is a 'hybrid' system, in that it either uses feature detection and matching or optic flow tracking to find corresponding points between each input frame and the stored maps. Which method it uses depends on if the system is 'lost' or not. The stages depicted in Figure 2 are described below.

A. Image Decoding

On mobile devices it is often necessary to convert the image formats for use in AR. The input camera format and output screen format are usually incompatible and require a conversion step. The vast majority of smartphone and tablet cameras return images in some variation of the YUV format. The common YUV420 format is convenient for vision algorithms as the luminance data block can be used directly as a grayscale image. However, in order to display a color image on the screen (using the Android API) or upload it as an OpenGL texture, it must be converted to an RGB format. A resolution of 640x480 is used throughout the system, which the camera is able to return directly without the need for software down-sampling. See Appendix A for conversion formulae.

B. Feature Detection (performed in 'lost' hybrid mode)

Our system is based on the ORB descriptor, which is an oriented version of the BRIEF descriptor [17]. Both ORB and BRIEF are binary descriptors, meaning the descriptor itself is a binary string rather than a floating-point vector as is the case with the SIFT and SURF algorithms. To build the ORB descriptor, interest points (typically FAST corners) are detected in the image and a series of pixel intensity comparisons are carried out between the interest point and some distribution of nearby pixels (256 comparisons in the case of the 32-byte descriptor). A single bit is required to store the result of each comparison, and each comparison is very fast to compute. This makes ORB very well suited for applications where memory and computation resources are at a premium.

The most notable downside to the ORB descriptor is that it is not scale invariant. This can be compensated for by extracting features at different image scales in the input image and/or in the stored map but this was found to reduce execution speed too greatly. Instead, the system requires that maps be detected at or near their home position.

There are two general cases considered when extracting features from the new camera frame. First, if the system is not already localized to a specific map (the 'lost' state,) features are extracted from the entire image and then used to determine which map, if any, the camera is looking at. The second case occurs when the system is already localized to a map, but existing feature inliers are not being tracked (conditions for tracking may not have been met). In this case, features are only extracted from within the bounding box of the map as it appeared in the previous frame. This bounding box is updated each frame during the map update process.

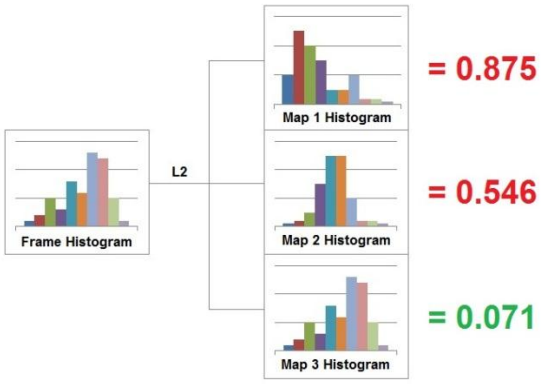

Figure 3 – Histogram comparison for map recognition

C. Frame Lookup (performed in 'lost' hybrid mode)

Newly detected features in the camera frame are matched to the map set using a bag of words scheme. Normalized response histograms are created for both frame and map by matching their descriptors to a descriptor vocabulary. These histograms are compared using their L2 distance, and the map whose histogram gives the lowest distance is returned as the most likely to be present in the frame (Figure 3).

One notable shortcoming is that we do not apply a hard threshold on whether or not a map is present at all. Even if there is no map present in the scene, the system will always return the most likely candidate. The reason for this is that we could not find a value that reliably separated the two cases. The system relies on the descriptor matching phase described in the next section to determine if the map is actually present. Once the most suitable map is found, it is matched to the camera frame.

D. Descriptor Matching (performed in 'lost' hybrid mode)

Comparing binary descriptors is done by calculating their Hamming distance. This consists of an exclusive-or operation followed by a population count of the result. This can be done quickly using built in XOR and POPCNT instructions, but we achieve a further speed increase by once again using the SIMD engine. Rather than iterating through the 256-bit descriptor one integer at a time (8 integers total), they are stored across two 128-bit registers and processed in only two parts. This yields an approximate threefold increase in speed, even over the POPCNT instruction.

Frame descriptors are first matched to the roots of the candidate map, and then to the branch whose root yielded

[2] This precludes non-ARM based or pre-NEON ARM devices from benefiting from some of these speedups, but as ARM currently holds 95% of the smartphone market [16], we feel this is a fair restriction.

the best score. Matches are filtered using two thresholds: an absolute threshold of 75 for the best Hamming score (this value was picked based on the distribution of true positive scores in [17]), and a relative threshold of 0.75 for the ratio between the best and second best matches. Additionally, a threshold of 20 is set as the minimum number of good matches that must be found for the map to be deemed detected. This number was determined empirically, based on the observed average number of false matches that occur when there is no map present in the frame. Using this threshold the system ignores frames that do not contain a map, regardless of the best map returned in the lookup phase.

E. Pose Estimation

If a suitable number of good matches are found, the system attempts to estimate the pose of the camera using the known 3D coordinates of the map and their corresponding 2D frame locations. We have found that improved performance can be achieved by using two different pose estimation algorithms, one for each of the hybrid modes ('lost' or 'tracking'). When determining pose, two cases are considered, one for each mode of the hybrid system: pose from descriptor matches, and pose from tracked inliers (discussed below). Despite the various limits and thresholds applied during the matching process, it is still possible for false positives to be present. These bad matches can comprise a significant portion of the correspondences, and, if not rejected, degrade the accuracy of the recovered camera pose. The goal is to use only correct matches in the calculation of the pose. To do this, the EPnP [18] technique is used inside a RANdom SAmple and Consensus (RANSAC) loop [19].

The maximum number of iterations was determined empirically. Since the RANSAC algorithm is not deterministic, the integrity of the camera pose depends largely on the probability of selecting four accurate correspondences within the maximum number of iterations. 5000 iterations may have an infinitesimal chance of failing to find inliers, but the running time would be far too high. Therefore, this probability is weighed against the iteration speed in order to find a middle ground between accuracy and efficiency. In practice, when the correct map is present in the camera image, the number of outliers was typically below 25%. However, under extreme conditions this number was observed to climb to 50% (or higher), so the maximum number of iterations was calculated based on a 50% outlier ratio. Additionally, to determine the number of iterations, the desired probability of selecting four good inliers was arbitrarily set to 99%. The number of iterations was set to 72 using Equation (3) in appendix B.

There is one main downside to pose estimation using feature matches: The same features are not always detected in every frame, which means the set of inliers is always different. This causes the camera pose to differ slightly each frame, which manifests itself as jitter in the displayed augmentation. To combat this, an inlier tracking method is used.

F. Inlier Tracking (performed in' tracking' hybrid mode)

In our system we utilize an optic flow [20] tracking mode, both for speed performance and to allow for a greater range of motion without losing pose. Inlier tracking begins when the number of inliers reaches 50% of the total map size or higher. This value was obtained by observing that the average number of matches under ideal conditions is typically two thirds of the total map size (66%), and the average number of inliers under ideal conditions is 75% or more, which combine for 50% or more of the total map size. This requires the camera to be moderately close to its original position when the map was created, but when tracking it is desirable to begin with as many inliers as possible.

When tracking begins, instead of extracting and matching new features at the start of the next frame, inliers from the previous frame are tracked in the new frame using optic flow. This provides a number of benefits. First, tracking existing features is computationally faster than extracting new features and matching them. Second, because these are inliers, they are already known to be accurate matches. Rather than performing RANSAC-based pose estimation on the tracked inliers, the system skips straight to least-squares pose estimation. This time, however, an iterative pose estimation method based on Levenberg-Marquardt optimization [21] present in OpenCV is used instead of EPnP. The reason for using the iterative method is that it produces more consistently robust results at extreme viewing angles. Although the iterative method is slower in general, performing least-squares iterative pose estimation is still much faster than using RANSAC based EPnP. Using optic flow and iterative pose estimation, the system is able to track the map plane until it is nearly perpendicular to the camera plane and still achieve a robust, stable pose.

This represents one of the contributions of this paper, the observation that a RANSAC loop not needed in the tracking mode if properly initialized, and that furthermore the iterative pose mode provides better visual stability for AR.

Ideally, every inlier in the previous frame will be tracked into the new frame. This is not realistic, however. Each new frame will result in fewer and fewer successful tracks due to occlusion, difficult viewing conditions, motion blur, or even camera noise. Eventually there will be too few tracks remaining to obtain a reliable pose. When this occurs, the system assumes the map has been lost, extracts new features, and goes back to the frame lookup step. The lower limit on the number of tracks is set to 20. This number is based solely on observation of augmentation stability.

G. Updating the Map

The final process in the pipeline is to dynamically update the map. First, the projection matrix is computed

from the pose and camera parameters. It is used to re-project the 3D bounding corners of the map into 2D image space to provide the region of interest used in feature detection.

Secondly, if features were extracted from the current frame, inlier descriptors are added to the map. In this way, each 3D world point in the map can have more than one descriptor associated with it. In order to prevent the map from growing overly large, the number of descriptors that can be associated with each point is limited. When the limit is reached, older descriptors are removed in favor of the newer ones with the exception of the original descriptor, which is never removed. This allows map features to be more consistently detected from different viewpoints.

H. Drawing Augmentations

Once the map has been updated, the final task is to pass the recovered pose back to OpenGL to be used to create a model view matrix (Appendix C). With a model view matrix obtained via an accurate camera pose, any 3D augmentation can be drawn. With sufficient 3D graphics experience, one could design complex augmentations that integrate seamlessly with their real world surroundings.

In the next section, quantitative and qualitative evaluations of the described system are performed.

IV. Experimental Results

In this section, the components of the AR pipeline are analyzed in detail. The algorithms used by our system are compared to other candidate algorithms, with particular attention being paid to execution speed. Our mobile test platform is an Asus TF201 tablet (1.4GHz). Two modern laptops are also tested; an average performance Lenovo W520 (2.2GHz) and a more powerful Alienware M17 (3.6GHz). The laptop tests are done in order to illustrate the performance difference between modern laptops and smartphones/tablets.

No multi-threading or GPU acceleration is used in the laptop trials, just a single core of the processor. Unless otherwise noted, every trial was averaged over 1000 frames, at a resolution of 640x480. Implementations of feature detectors, pose estimation algorithms, and the optic flow algorithm are from the OpenCV library version 2.4.3.

A. Image Decoding

Image decoding speed is measured for three implementations: floating-point and integer based conversions performed on the CPU, and our NEON-accelerated integer-based conversion. Table 1 shows an approximate 70% reduction in computation time on the tablet when using NEON acceleration.

B. Feature Detection

Speed trials were conducted to compare the chosen ORB feature detector to SIFT and SURF on each platform. In each trial, 300 features were extracted from a still image. Table 2 shows the average time required per image for each detector. An important thing to notice about these results is how the computation time scales between the laptop and tablet trials. While the feature detectors scale at a similar rate on all platforms, laptop speeds are an order of magnitude or more faster than the tablet despite having a clock speed that is only 2-3 times faster.

An unexpected result of this test was that SIFT ran faster than SURF across all platforms, despite SURF having been developed to improve upon the slow running time of SIFT. According to the OpenCV developers, this is due to their highly efficient implementation of the SIFT algorithm [22]. Regardless, these trials indicate that both SIFT and SURF are unsuitable for real-time applications on mobile devices, typically requiring between 0.5 and 1.0 seconds per frame.

C. Frame Lookup

Frame lookup consists of matching the frame's descriptors to the vocabulary, creating a histogram, and comparing it to each map histogram. This is not a costly operation and takes fewer than 2ms in all cases (Table 3).

D. Descriptor Matching

The comparison speed of floating point descriptors and binary descriptors are measured in this trial. These results were then compared to the NEON-accelerated implementation on the tablet. Table 4 shows the comparison speed for a single pair of descriptors averaged over one million trials. On the tablet, ORB descriptor comparison is approximately 3.5 times faster than SURF, and, even when using ARM's built in population count instruction, adding NEON acceleration speeds up binary descriptor comparison by an additional factor of 3.25.

Additionally, the speed of NEON-accelerated linear search matching (brute force) is compared to the clustering tree method employed by this system. In this test, 300 frame descriptors are matched to 500 map descriptors using each method. In each trial, all matching checks and criteria are performed. Table 5 shows the results.

E. Inlier Tracking

Since Optic Flow is independent of the detector used to initially find the tracked features, it is straightforward to measure. Table 6 shows the tracking speed for 300 features on each test platform.

F. Pose Estimation

Both pose estimation methods used by the system (Iterative and EPnP) were measured in terms of speed. This is straightforward for the iterative method, since it is only used in least-squares estimation while tracking. For EPnP the computation time depends largely on the number of RANSAC iterations that are performed before a suitable pose is found. Table 7 presents speed trials for least-squares pose estimation using both the Iterative and EPnP methods in order to provide a baseline. In addition, a worst-case scenario of 72 RANSAC iterations is presented for EPnP. In all cases, a set of 300 correspondences is used.

Table 1 - YUV to RGB decoding speed.

Platform	Clock Speed (GHz)	Average Decoding Speed (ms)			NEON Speed-up
		CPU (Float)	CPU (Integer)	NEON (Integer)	
M17	3.6	5.05	**1.42**	N/A	N/A
W520	2.2	8.20	**2.81**	N/A	N/A
TF201	1.4	16.0	9.02	**4.82**	1.87x

Table 2 - Detection and description speeds of SIFT, SURF, and ORB.

Platform	Clock Speed (GHz)	Extraction Speed (ms/frame)		
		SIFT	SURF	ORB
M17	3.6	66.5	88.1	**3.60**
W520	2.2	84.7	160	**5.87**
TF201	1.4	694	1034	**41.0**

Table 3 – Frame lookup speed

Platform	Clock Speed (GHz)	Lookup Speed (ms)
M17	3.6	0.816
W520	2.2	1.58
TF201	1.4	1.81

Table 4 - Average descriptor comparison speed.

Platform	Clock Speed (GHz)	Comparison Speed (µs/pair)				NEON Speed-Up
		128-float SIFT	64-float SURF	256-bit ORB	256-bit NEON	
M17	3.6	0.362	0.172	**0.0237**	N/A	N/A
W520	2.2	0.466	0.230	**0.0435**	N/A	N/A
TF201	1.4	0.886	0.455	0.132	**0.0406**	**3.25x**

Table 5 - Speed of linear search matching to clustering tree matching.

Platform	Clock Speed (GHz)	Matching Speed (ms/frame)		Speed-Up
		Linear Search	Clustering Tree	
M17	3.6	4.50	**0.921**	**3.95x**
W520	2.2	8.30	**1.97**	**4.21x**
TF201	1.4	18.8	**4.25**	**4.42x**

Table 6 - Optic Flow tracking speed for 300 tracked features.

Platform	Clock Speed (GHz)	Tracking Speed (ms)
M17	3.6	7.49
W520	2.2	13.1
TF201	1.4	49.5

Table 7 - Least-squares Iterative and EPnP pose estimation, as well as worst case RANSAC EPnP.

Platform	Clock Speed (GHz)	Pose Estimation Speed (ms/frame)		
		Iterative	EPnP	EPnP (RANSAC)
M17	3.6	2.99	0.230	8.35
W520	2.2	3.17	0.263	9.99
TF201	1.4	10.7	1.69	20.9

G. *Putting it All Together*

Based on the incremental results from the previous sections, the figure below shows the overall execution speed of our ORB-based system while matching (Figure 4) and tracking (Figure 5). This data is based on a slowest possible scenario, with each component being tested using the maximum possible number of features, correspondences, and RANSAC iterations. In practice, these maximum values will almost never occur. Therefore, a series of end-to-end tests were performed on two recorded videos and the overall speed of the system is measured frame by frame.

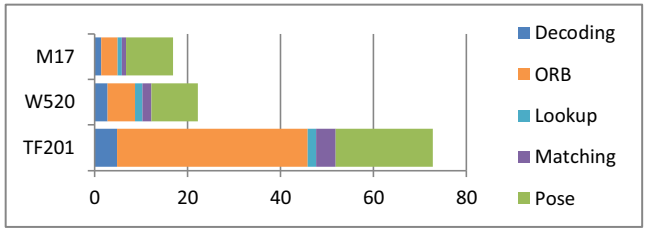

Figure 4 - Overall Speed while matching (ms).

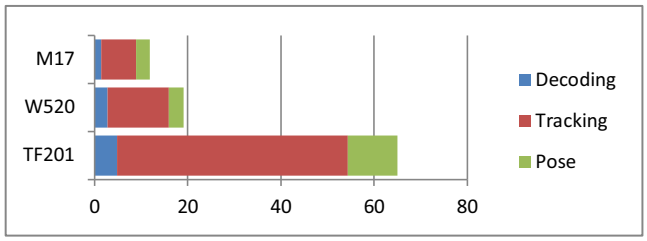

Figure 5 - Overall speed while tracking (ms).

The first video is of a plain tabletop environment containing six different maps (Figure 6), and the second is of a noisy lab environment with 10 textured surfaces throughout the lab being used as maps (Figure 7). Results are displayed in Figure 8 and Figure 9. For clarity, the results are displayed as a moving average with a period of 30 frames.

The speed differences between periods of tracking and matching are most clearly visible in Figure 8. A consistent, minimum valley is visible across the entire duration for all platforms, with peaks interspersed throughout the data. These peaks correspond to the system having lost the map and extracting and matching new features. The valleys correspond to periods of inlier tracking. In Figure 9 these peaks and valleys are less defined, owing to the fact that the lab environment is far noisier than a series of objects on a plain table top. Additionally, the lab video contained a much higher percentage of frames that did not contain the scene from a map.

H. *Qualitative Results*

Since the goal of any AR system is to create a seamless and immersive user experience, a series of images below show the system in action from the point of view of the user through live screen captures (Figure 10). The top row shows a 3D cube being augmented at three different scales, as well as an oblique angle. The bottom row shows three different maps being recognized, with different colored triangles being drawn depending on the map that is being tracked.

V. Conclusion and Future Work

This paper presented a markerless augmented reality system designed to operate in real time on current generation mobile devices. The system was tested on two laptops and a tablet, and on the tablet was shown to operate at speeds of approximately 30Hz while tracking, and 15Hz while extracting and matching features.

Figure 6 - Plain tabletop environment.

Figure 7 - Panorama of noisy lab environment.

The most immediate future work will involve implementing the ORB feature detector using NEON and the SIMD engine. If the system's other NEON implementations are any indication, this will greatly improve the worst-case execution speed of the system and perhaps allow feature matching to be performed on top of optic flow tracking in order to provide a more robust pose estimate.

In addition, as hardware evolves, acceleration methods used in PC implementations will become increasingly relevant to mobile hardware. A popular way to improve algorithm efficiency is to parallelize through GPU acceleration. Nvidia recently announced their new Tegra 4 processor which will contain a programmable 72-CUDA core GPU [23], allowing CUDA implementations to be used on mobile devices.

As efficiency increases, additional functionality can be added. Currently, the system cannot identify multiple maps in a single camera frame; it will pick whichever map returned a better lookup score. Future work will involve adding functionality to recognize multiple maps in a scene and render unique augmentations for each map. This could be done by accepting multiple maps whose look-up scores are above some threshold, or within a certain percentage of one another. Once potential maps have been selected, the descriptor matching and pose estimation processes would be applied independently to each one. The matching process is fast enough through NEON acceleration that it could be performed multiple times without a large impact on performance. And, though pose estimation is much slower, the effect of doing it more than once could be absorbed by having sped up feature detection using NEON or GPU acceleration. This would yield multiple camera poses, each of which would be sent back to OpenGL where separate augmentations would be drawn.

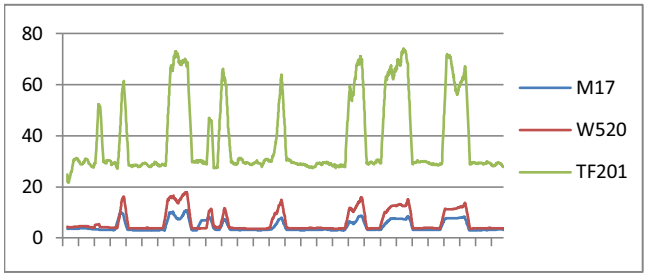

Figure 8 – Processing speed (ms) of tabletop video (2775 frames).

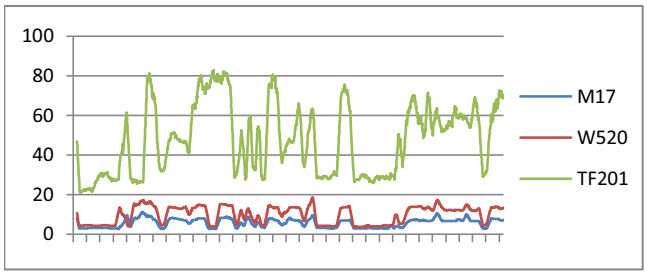

Figure 9 - Processing speed (ms) of lab video (3233 frames).

ACKNOWLEDGMENT

This work was funded by NSERC Discovery Grant 356072

REFERENCES

[1] M. Fiala, "ARTag, a fiducial marker system using digital techniques," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005, pp. 590-596 vol. 2.

[2] M. Fiala and G. Roth, "Magic Lens Augmented Reality: Table-top and Augmentorium," presented at the ACM SIGGRAPH 2007 posters, San Diego, California, 2007.

[3] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, pp. 2564-2571.

[4] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 29, pp. 1052-1067, 2007.

[5] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, 2007, pp. 225-234.

[6] L. Taehee and T. Hollerer, "Hybrid Feature Tracking and User Interaction for Markerless Augmented Reality," in *Virtual Reality Conference, 2008. VR '08. IEEE*, 2008, pp. 145-152.

[7] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 1999, pp. 1150-1157 vol.2.

[8] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Computer Vision–ECCV 2006,* pp. 404-417, 2006.

[9] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," presented at the European Conference on Computer Vision, 2006.

[10] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose tracking from natural features on mobile phones," presented at the Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, 2008.

[11] W.-C. Chen, Y. Xiong, J. Gao, N. Gelfand, and R. Grzeszczuk, "Efficient Extraction of Robust Image Features on Mobile Devices," presented at the Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007.

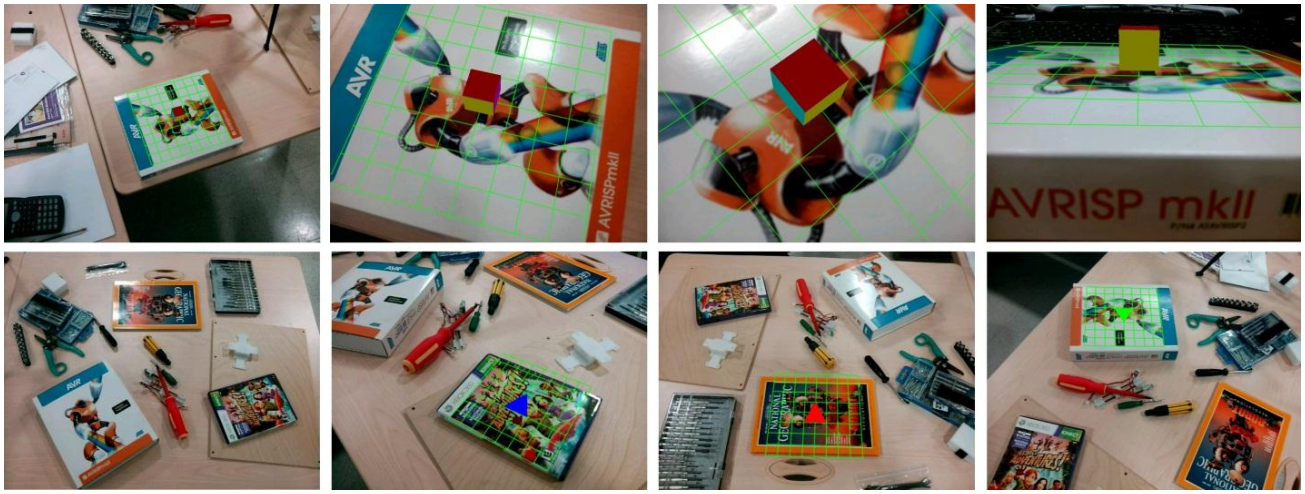

Figure 10 - System screen captures using multiple maps under various viewing conditions.

[12] M. Mohring, C. Lessig, and O. Bimber, "Video See-Through AR on Consumer Cell-Phones," presented at the Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality, 2004.

[13] G. Klein and D. Murray, "Parallel Tracking and Mapping on a camera phone," in *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, 2009, pp. 83-86.

[14] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-Time Detection and Tracking for Augmented Reality on Mobile Phones," *Visualization and Computer Graphics, IEEE Transactions on,* vol. 16, pp. 355-368, 2010.

[15] M. Muja and D. G. Lowe, "Fast Matching of Binary Features," in *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, 2012, pp. 404-410.

[16] T. P. Morgan. (2012). *ARM Snags 95 Percent Of Smartphone Market, Eyes New Areas For Growth*. Available: http://www.crn.com/news/components-peripherals/240003811/arm-snags-95-percent-of-smartphone-market-eyes-new-areas-for-growth.htm

[17] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," *Computer Vision–ECCV 2010,* pp. 778-792, 2010.

[18] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o (n) solution to the pnp problem," *International Journal of Computer Vision,* vol. 81, pp. 155-166, 2009.

[19] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM,* vol. 24, pp. 381-395, 1981.

[20] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," Carnegie Mellon University1991.

[21] K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *Quarterly of Applied Mathematics,* pp. 164-168, 1944.

[22] W. Garage. (2012). *OpenCV Change Log*. Available: http://code.opencv.org/projects/opencv/wiki/ChangeLog

[23] J. Hruska. (2013). *Nvidia's Tegra 4 demystified: 28nm, 72-core GPU, integrated LTE, and questionable power consumption*. Available: http://www.extremetech.com/computing/144942-nvidias-tegra-4-demystified-28nm-72-core-gpu-integrated-lte-and-questionable-power-consumption

APPENDICES

A. YUV Conversion

We modify the traditional floating-point conversion (1) to a reduced integer approximation (2) in order to avoid costly floating point arithmetic and better accommodate the SIMD engine.

$$\begin{aligned} R &= 1.164(Y-16) + 1.596(V-128) \\ G &= 1.164(Y-16) - 0.813(V-128) - 0.391(U-128) \\ B &= 1.164(Y-16) + 2.018(U-128) \end{aligned} \quad (1)$$

$$\begin{aligned} R &= 75(Y-16) + 102(V-128) \gg 6 \\ G &= 75(Y-16) - 52(V-128) - 25(U-128) \gg 6 \\ B &= 75(Y-16) + 129(U-128) \gg 6 \end{aligned} \quad (2)$$

While it is common practice to use integer coeffients, our formula in (2) further restricts the conversion so that at no point will an intermediate result exceed 16 bits. This allows eight pixels to be processed simultaneously in a 128-bit SIMD register as opposed to four in the case of using 32-bit integers. This effectively cuts the conversion time in half, and the resulting image is nearly indistinguishable to the human eye.

B. RANSAC Iteration Formula

This formula was derived based on randomly selecting four inliers (n = 4) from a data set containing 50% outliers (R_o = 0.5) with a 99% confidence ratio (R_c = 0.99).

$$Iter_{MAX} = ceil\left[\frac{\log(1-R_c)}{\log(1-{R_o}^n)}\right] \quad (3)$$

C. Pose to OpenGL Model-view Matrix

Equation (4) shows how a rotation matrix and translation vector can be used to create the OpenGL model-view matrix.

$$M = \begin{bmatrix} R_{0,0} & -R_{1,0} & -R_{2,0} & 0.0 \\ R_{0,1} & -R_{1,1} & -R_{2,1} & 0.0 \\ R_{0,2} & -R_{1,2} & -R_{2,2} & 0.0 \\ t_{0,0} & t_{1,0} & t_{2,0} & 1.0 \end{bmatrix} \quad (4)$$