Detecting Planes in Real-Time for Camera Display Communications

Independent Study Report

Revan Sopher

May 5, 2015

Abstract

We implement a prototype Android system for extracting a message from a subtly intensity-encoded carrier image. The system is novel in that it requires no camera calibration in color, time, nor space: it is tolerant of variation in ambient light, temporal synchronization, and observation angle. The system demonstrates $> 90\%$ accuracy running on consumer hardware.

1 Introduction

1.1 Visual MIMO

The aim of the Visual MIMO project thus far has been to embed messages within images by varying the intensity of patches of the image, and to reliably extract these messages from pictures of the image displayed on a screen.

The existing software system is capable of detecting the computer monitor, in order to to consider only the displayed image, but encounters difficulties with the variance in photometry dependent on camera and display type, and the spatial positioning of the two. This is addressed via radiometric calibration: nearly invisible patches are created on the corners of the image using histogram equalization, creating calibration data.

Although the Visual MIMO project is intended to make use of the ubiquity of handheld cameras, previous work has been reliant on a precise, static configuration of high quality SLR camera and a desktop computer, with processing performed post hoc.

The purpose of this independent study independent study is to implement this algorithm on a mobile device, as in Figure 1.

1.2 Mobile Challenges

This requires special considerations and optimizations to compensate for the significantly weaker processor and average cameras: the processing limitations can be mitigated by lower level programming and memory management using the Android Native Development Kit, but the lower quality image and light sensors has no obvious fix.

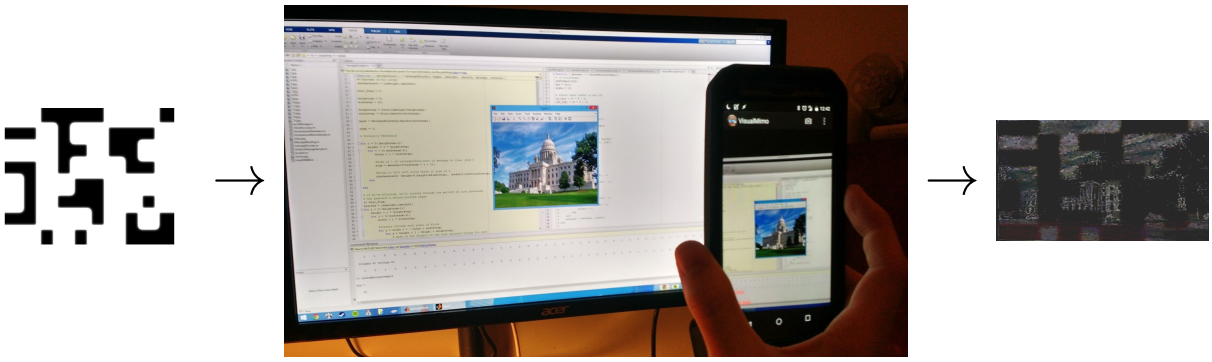

Figure 1: The final system in action. A computer monitor displays a message embedded in a carrier image, and an Android phone extracts and decodes the message visually.

The primary focus of this study is in addressing the issues that arise from the spatial differences between the camera and the screen: although radiometric calibration can mitigate such differences, the resulting calibration is only valid for the current position, resulting in a fragile setup unsuitable for casual demonstrations.

The goal is to increase the accuracy of imperfect positioning, without requiring explicit calibration.

2 Message Encoding

The source message is generated in MATLAB. The bit string message to be encoded is turned into an 8x10 grid where each sector is either "on" with an intensity of α, or "off" with an intensity of 0. Higher values of α are more accurate, but more visible. The edges of the blocks are blended to reduce the visual impact of the sharp edges.

This grid is added and subtracted to the carrier image to create two frames (Figure 2). These frames are displayed alternating on the computer screen.

3 Mobile Image Processing

3.1 Plane Tracking with Vuforia

The Android application uses the Qualcomm Vuforia SDK [4], a library meant for facilitating the construction of Augmented Reality applications. Vuforia is used for its image tracking capabilities: known image targets are trained ahead of time, and are quickly and reliably detected in real-time.

Upon detection, Vuforia provides the pose matrix. Combined with the known size of the target image camera parameters, we can calculate the position of the image corners, allowing us to process only the relevant part of the image.

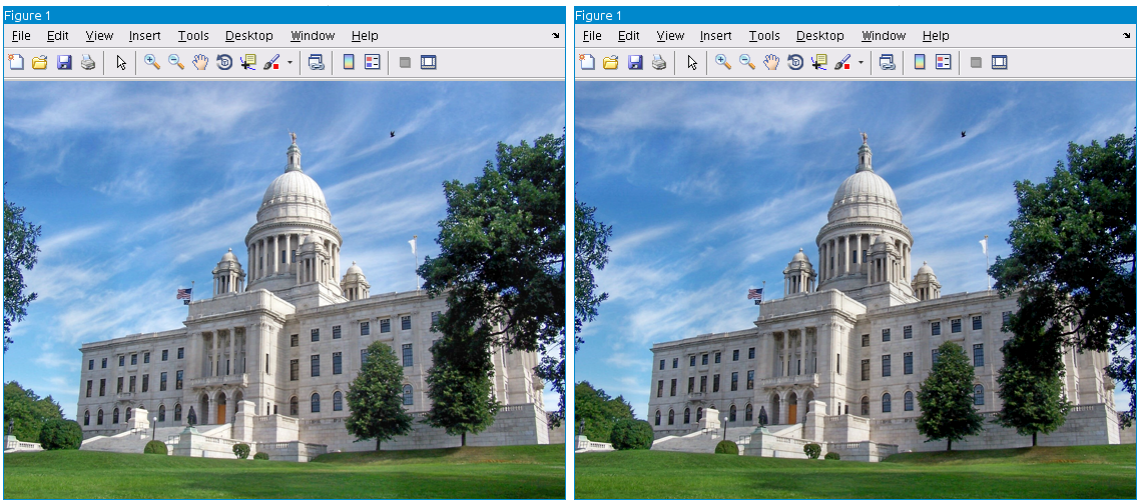

Figure 2: An example of message embedding: the message is turned into a subtle grid pattern, then added and subtracted to carrier image to produce two frames.

3.2 Skew Correction

The usage of a mobile device for image capture necessarily forecloses angular calibration. As such, the image captured is usually not perfectly head on. In this step, we use a projective transformation to correct for skew.

First we obtain the 3×3 transformation matrix, such that:

$$\begin{pmatrix} t_i x_i^* \\ t_i y_i^* \\ t_i \end{pmatrix} = \text{transformation\_matrix} \cdot \begin{pmatrix} t_i x_i \\ t_i y_i \\ 1 \end{pmatrix}$$

That is, we map each corner to the proper rectangular positions.

Having calculated the corner positions previously and knowing the shape of the original image, this is easily accomplished with OpenCV[1].

Next, we apply the transformation by multiplying the found matrix with each point in the image. This is again straightforward with OpenCV [3], yielding Figure 3.

3.3 Histogram Equalization

In the histogram equalization step, we improve the image contrast and transfer faithfulness by "stretching" the pixel intensity histogram to resemble a normal distribution (Figure 4).

We perform histogram equalization on both the source image and the captured image, which should increase message fidelity.

Figure 3: An image viewed at an oblique angle, then the image, skew-corrected and extracted by projective transformation.

Optimally we apply the same transformation to both images, however this is not especially straightforward using OpenCV so we rely on the two images having similar transforms and run the histogram equalization function twice.

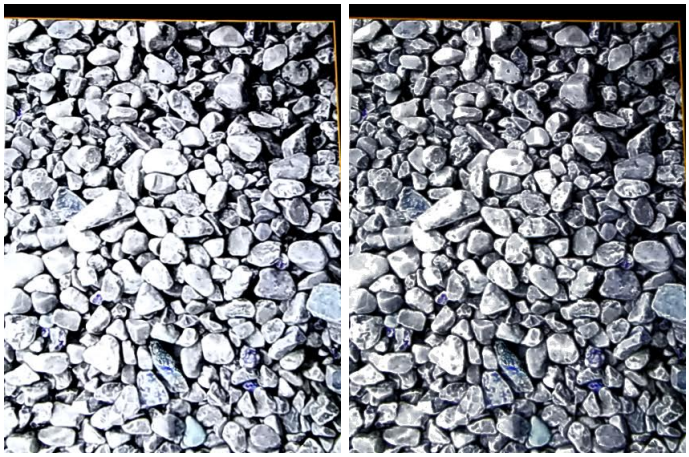

Figure 4: An example of histogram equalization of an image: original on the left.

3.4 Subtraction

With both targets skew-adjusted and normalized, we subtract one image from the other:

$$\text{dst}[I] = \text{saturate}(\text{src1}[I] - \text{src2}[I])$$

where I is the image matrix[2].

As the non-message part of the images should match up, this should leave only the message (Figure 5).

$$\frac{(\text{frame} + \text{message}) - (\text{frame - message})}{2} = \text{message}$$

Figure 5: An example of subtraction at $\alpha = 10$. 96.25% accuracy on the retrieval of the embedded message "abcdefghijk". Brightness and contrast of the resulting image were increased for ease of viewing in this document.

3.5 Information Extraction

Having isolated the encoded pattern, we must then decode the information inside.

We divide the difference image into a grid, and compare the average intensity of each sector against the average intensity of the entire image. Assuming a mostly even distribution of 1's and 0's, the average intensity of the entire image will be approximately halfway between the "on" and "off" intensities. Thus we interpret any sector as a 1 or 0 depending on if its average intensity is greater or less than the general average.

We assemble the bit string, row by row.

4 Synchronization

Earlier iterations of the system had no synchronization measures in place, so the two frames chosen for subtraction were often not properly matched. Combination such as both frames displaying the message added, or both subtracted, lead to nearly resulting images.

The system would also break when encountering frames that are between clean updates on the screen: there is a switching period involved in updating the source display.

4.1 Multisampling

This synchronization method involved taking four frames rather than just two, and selecting the best two frames to perform the subtraction and extraction on. For each frame, we calculate the average intensity, and select the combination of two frames which maximizes the absolute value of the difference of their intensities.

We estimate that the refresh speed of the camera is 20 FPS, so we fix the refresh speed of the encoded message at 10 FPS, the Nyquist frequency for this camera.

The reasoning for this method is based on the idea that the transitions between frames on the display are non-instantaneous, causing inaccuracies in decoding. By sampling at twice the rate of the display, we can be reasonably sure that we will capture at least one clean picture of both frames, (img+msg) and (img-msg). (Figure 6).

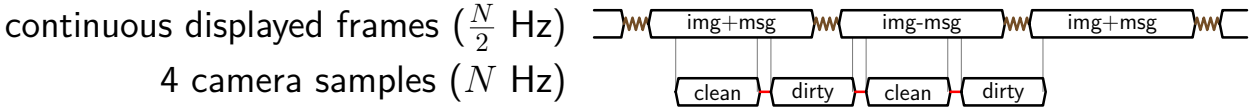

Figure 6: Multisampling technique compensates for synchronization and switching times to reliably provide a clean image of both the positive and negative frames

5 Results

We test the message recovery accuracy for the message "abcdefghijk", the longest text that can fit in the $8 \times 10 = 80$ bits. Unless otherwise noted, tests were at $\alpha = 10$, ≈ 1 meter distance, head-on.

5.1 Various Carrier Images

We first test fifteen carrier images (Figure 7) to find average accuracy (Table 1). These images are trained for Vuforia, but some are more easily tracked than others; images 2, 9, and 11 could not reliably be tracked by Vuforia, making them unfeasible for the system. Results show that, ignoring images 3 and 12 which were dark and without color, the system achieves $> 90\%$ accuracy.

5.2 Various α Values

We then test the effects of various values of α on the best performing carrier images. Higher values of α are easier for the system to process, but also easier to detect by a human: we try to minimize the value. We show that at $\alpha = 5$, the accuracy degrades below 90% (Table 2).

5.3 Various Viewing Angles

We now test the effects of differing viewing angles on accuracy. Previous trials were captured at approximately 0°, that is, head on. This experiment shows that message accuracy does not

Table 1: Various Carrier Images, $\alpha = 10$

	Accuracy Per Trial						
Image	1	2	3	4	5	AVG	
1	0.9375	0.975	0.9625	0.975	1	0.97	
2	—	—	—	—	—	—	could not track
3	0.675	0.5625	0.625	0.5875	0.5625	0.6025	black and white
4	0.8375	0.8625	0.9125	0.9	0.9	0.8825	
5	0.9625	0.95	0.95	0.975	0.9375	0.955	
6	0.7625	0.775	0.825	0.8125	0.8125	0.7975	
7	0.95	0.95	0.9	0.95	0.95	0.94	
8	0.9125	0.7875	0.9125	0.9125	0.9125	0.8875	
9	—	—	—	—	—	—	could not track
10	0.8875	0.8875	0.8625	0.8875	0.8875	0.8825	
11	—	—	—	—	—	—	could not track
12	0.6125	0.55	0.6125	0.55	0.6375	0.5925	black and white
13	0.9375	0.8375	0.9875	0.975	0.9625	0.94	
14	0.9125	0.975	0.95	0.95	0.95	0.9475	
15	0.975	0.9375	0.9875	0.975	0.9875	0.9725	
					Total avg:	0.8642	
					Color avg:	0.9175	

appreciably change as we increase the obliqueness of the view, however the reliability of Vuforia's target tracking drops off approaching 45°, to the point that target acquisition was noticeably slow for most images. (Table 3).

6 Future Work

6.1 Multi-frame Messages

The implementation described thus far only allows for as much information as can be encoded into a single image. In order to scale this prototype arbitrarily, we must be able to encode and decode a series of messages sequentially.

To this end, the last few blocks of the message will be reserved for an index giving the position of the message relative to the other messages, in a Sliding Window implementation. In such an implementation, the source display would loop through the series of messages, displaying the (img+msg) and (img-msg) of each frames once each. The mobile device would begin collecting frames from an undefined point, using the encoded index to restructure the original message.

Synchronization in this design becomes problematic, however, as we must beware of attempting to difference mismatched frames.

Figure 7: The fifteen carrier images tested. Images 1, 5, 7, 13, and 15 were the best.

6.2 Google Glass

Porting to Google Glass poses several problems. Primarily, the processing capabilities are limited: not only is the processor weaker, but the heat generated by performing heavy computations locally causes the device to overheat. Therefore, the "proper" approach to such an application on Google Glass would consist of offloading all processing to a server, however this means such an implementation is less technically interesting.

References

[1] *OpenCV Documentation: getPerspectiveTransform*. 2014. URL: `http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#getperspectivetransform`.

[2] *OpenCV Documentation: subtract*. 2014. URL: `http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#subtract`.

Table 2: Various α Values: at $\alpha = 5$, the accuracy degrades to below 90%

		Accuracy Per Trial							
Image	Alpha	1	2	3	4	5	AVG	avg/image	avg/alpha
1	20	1	0.9875	1	0.9875	0.9875	0.9925	0.9675	0.9535
	10	0.9875	0.925	1	0.9875	0.975	0.975		0.955
	5	0.875	0.975	0.8625	0.975	0.9875	0.935		0.8819
5	20	0.95	0.95	0.95	0.95	0.95	0.95	0.9225	
	10	0.95	0.95	0.95	0.9375	0.975	0.9525		
	5	0.9625	0.925	0.8375	0.9	0.7	0.865		
7	20	0.95	0.95	0.7875	0.95	0.95	0.9175	0.9217	
	10	0.95	0.9375	0.95	0.9375	0.925	0.94		
	5	0.8	0.9375	0.9	0.95	0.95	0.9075		
13	20	0.9375	0.9375	0.925	0.9625	0.9375	0.94	0.909	
	10	0.9375	0.95	0.875	0.95	0.95	0.9325		
	5	0.875	0.9125	0.875	0.8625	0.7475	0.8545		
15	20	0.95	0.9625	0.9875	0.975	0.9625	0.9675	0.92083	
	10	0.975	0.9625	0.975	0.9625	1	0.975		
	5	0.9375	0.85	0.6625	0.825	0.825	0.82		
						Total avg:	0.928		

[3] *OpenCV Documentation: warpPerspective*. 2014. URL: `http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#warpperspective`.

[4] Inc. Qualcomm Connected Experiences. *Vuforia Developer Portal*. 2014. URL: `https://developer.vuforia.com/`.

Table 3: Various Viewing Angles: Increasing angles causes no change in accuracy, but Vuforia tracking degrades.

		Accuracy Per Trial						
Image	Angle	1	2	3	4	5	AVG	avg per image
1	0	0.9875	1	0.9875	0.975	0.9875	0.9875	0.9708
	30	0.975	0.9625	0.975	0.9625	0.9875	0.9725	
	45	0.975	0.975	0.9625	0.9625	0.8875	0.9525	
5	0	0.9625	0.9625	0.9625	0.9625	0.925	0.955	0.9608
	30	0.9625	0.9625	0.925	0.95	0.975	0.955	
	45	0.975	0.9625	0.975	0.975	0.975	0.9725	
7	0	0.9375	0.95	0.95	0.925	0.95	0.9425	0.9533
	30	0.975	0.95	0.95	0.9375	0.9	0.9425	
	45	0.975	0.975	0.9875	0.975	0.9625	0.975	
13	0	0.975	0.9625	0.8625	0.975	0.9625	0.9475	0.9417
	30	0.9	0.9375	0.925	0.9625	0.9375	0.9325	
	45	0.9375	0.925	0.95	0.9625	0.95	0.945	
15	0	1	1	0.9875	0.9875	0.9625	0.9875	0.98725
	30	1	1	0.935	1	1	0.987	
	45	—	—	—	—	—	—	could not track
						Total avg:	0.961	