

# Using Backlight Intensity for Device Identification

Jeremiah J. Neubert\*

Tom Drummond†

Department of Engineering  
Cambridge University

## ABSTRACT

This paper presents a method for identifying handheld devices (*e.g.* smart phones and pocket PCs) to facilitate the use of these devices as tangible interfaces for desktop augmented reality systems. The proposed system leverages the ability of these handheld devices to programmatically control their backlight intensity to display a binary code. The codes produced are non-intrusive, require no specialized hardware, and can be generated with most handheld devices. This technique is shown to accurately and robustly identify up to 16 different devices in under 500 msec and is easily expandable to 256 or more devices.

## 1 INTRODUCTION

Handheld devices are cheap and ubiquitous, making them attractive for use as tangible interfaces to desktop augmented reality systems like those outlined in [5], [6], and [7]. Additionally, they are more flexible than traditional block interfaces allowing users to download data or applications [6]. Unfortunately, existing systems do not allow for more than one such device because there is no desirable method to identify these devices.

Methods exist for reliable identification of a group of tangible interfaces similar in appearance using mechanisms like paper fiducial markers [2] or infrared LEDs [1, 7, 4]. IR-tags, which utilize infrared LEDs, are attractive because they emit signals that, while not visible to the human eye, are bright and easily detected with a camera. Unfortunately, these methods are undesirable because handheld devices may be the personal property of the user; thus it is unlikely that users would be willing to mount IR-tags or fiducial markers on them. In addition, it is difficult to place markers on handheld devices because they can interfere with functionality and may require modifications to the device's hardware. For these reasons there is a need for an identification method that can operate on most handheld systems without additional hardware or markers.

One such method proposed in [3] varies the hue of the device's screen to emit a signal, but only uses one half the frame rate of the camera and requires the execution of an application that consumes the entire screen. Our method also uses the device's screen to emit an identifying signal, but varies the backlight brightness to produce the signal at the frame rate of the camera. This allows the device to be identified in a shorter period of time without consuming the display, minimizing interference with its usability.

## 2 THE SYSTEM

The identification scheme outlined here is created for use with an augmented desktop. While these systems may vary in construction a typical implementation is shown in Figure 1. A central computer

system is used to project augmentations onto the desktop while using a camera to monitor interactions with these augmentations. The server is able to identify the PDA by requesting that the device identifies itself using the method described below, which is a series dark and bright screens.

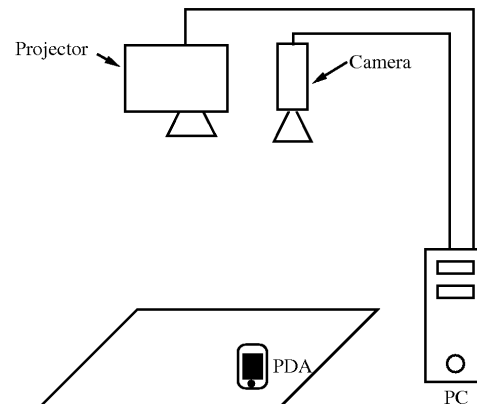
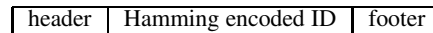


Figure 1: Typical augmented desktop system.

### 2.1 Binary Signal

The system is limited to a binary signal since we require a system that can be used with a variety of handheld devices thus only standardized APIs can be employed. Unfortunately, the only standardized method for controlling the backlight intensity on the Pocket PC is through the power management system, which only allows the screen intensity to be toggled between two states.



The structure of the binary code employed a (7, 4) Hamming code with a two bit header and footer. The header and footer consist of two low bits to denote the beginning and end of the signal, allowing the system to detect false signals as they are unlikely to be the proper length. This message structure provides 16 unique codes, the ability to detect and correct errors, and can be expanded to 256 devices with the (12,8) Hamming code. This structure supplies robustness and scalability not present in simpler schemes.

### 2.2 Extracting the Code

Recovering the binary code from the screen intensity occurs in two parts: extracting the signal from the data stream, and classifying screen intensities. The data stream consists of the average intensity of pixels corresponding to a device's screen extracted from images captured by the overhead camera shown in Figure 1. The vector  $\vec{p}_d = [p_{d0}, \dots, p_{dn}]$  contains the last  $n + 1$  screen intensities for device  $d$ , where  $p_{d0}$  is the oldest element in  $\vec{p}_d$ . The number of elements in  $\vec{p}_d$  needs to be large enough to contain the signal and a buffer on either end such that the code can be distinguished from a random sequence. The search for a valid code begins when

\*e-mail:jjn27@eng.cam.ac.uk

†e-mail:twd20@eng.cam.ac.uk

0	x	1	x = 0	initial value during frame capture is 0
1	x	0	x = 1	initial value during frame capture is 1
0	x	0	x = 1	An unknown between two 0's or 1's indicates the state changed between them
1	x	1	x = 0	
x	x			multiple unknowns dictates the code is ambiguous and can not be classified
x	x			

Table 1: Rules for addressing ambiguous intensities. The right column lists possible neighbors of the unclassified intensity  $x$ . The second column is the most likely value for  $x$  in given its neighbors

$p_{d(k-1)} - p_{d(k)} > \tau_{sig}$ , where  $\tau_{sig}$  is a user defined threshold and  $k$  size of the buffers. Assuming that the device's display is in the bright or high state, a positive difference between  $p_{d(k-1)}$  and  $p_{dk}$  indicates that  $p_{dk}$  and  $p_{d(k+1)}$  may be the header of a binary code.

Using the initial position estimate of the binary signal in the data stream, a more intensive search is conducted to obtain the most likely location  $i$  for the signal, which minimizes

$$\beta_i = \underbrace{p_{di} + p_{d(i+1)}}_{header} + \underbrace{p_{d(i+9)} + p_{d(i+10)}}_{footer}. \quad (1)$$

Once the signal is located tests are performed to reject random or corrupted codes. The first such test evaluates  $\vec{p}_d$  to ensure that the signal does not exceed the specified length. If the values of  $p_{d(i-1)}$  and  $p_{d(i+11)}$  are not significantly higher than those in the header and footer the signal is rejected as too long. In addition, if the footer or header contain values that are significantly higher than  $\min(\vec{p}_d)$ , which suggests a high bit, the signal is rejected.

When a valid sequence is detected at  $i$  the signal  $\vec{s}_d = [p_{d(i+2)}, \dots, p_{d(i+8)}]$  is extracted and the binary code is recovered by classifying each intensity value as a high or low bit. Classifying the values in  $\vec{s}_d$  could be done using any number of techniques. The simplest method would be to use a fixed threshold. However, this neglects the blurring generated by the exposure time of the camera and the time the screen requires to change state, which causes intensity values of the two classes to overlap. Bearing in mind the overlap of the two classes, two threshold are employed,  $\tau_{high}$  and  $\tau_{low}$ . The values of  $\tau_{high}$  and  $\tau_{low}$  are determined from  $\vec{p}_d$  to make the system robust to the variations in lighting generated by the projection of augmentations. The threshold are calculated using the equations

$$\tau_{low} = \lambda_1 * range(\vec{p}_d) + \min(\vec{p}_d) \quad (2)$$

and

$$\tau_{high} = \lambda_2 * range(\vec{p}_d) + \min(\vec{p}_d). \quad (3)$$

The values of  $\lambda_1$  and  $\lambda_2$  are fixed and reflect the spread of intensity values in their respective class, with  $\lambda_1 < \lambda_2$ .

The values in  $\vec{s}_d$  above  $\tau_{high}$  are considered high bits, those below  $\tau_{low}$  are classified as low bits. The remaining values which fall between  $\tau_{high}$  and  $\tau_{low}$  are classified using rules contained in Table 1 which reflect an unclassified value's most likely class given its neighbors. When the value has two neighbors of the same class, the variation in  $\vec{s}_d$  was likely generated by a change in the displays state between the two known bits; thus the bit assumes the opposite class. When there is an unclassified value between two different classes, which has likely been caused by sampling across a change in the screen's state, the unknown intensity is assigned the prior value's class because the sampling event began with the device in that state. When  $\vec{s}$  contains two neighboring unclassified values it is deemed ambiguous and the message must be repeated.

	Error Types			
	trials	corrected	detected	false id
Ideal	447	5	20	0
Desktop	320	0	21	0

Table 2: There were three types of errors reported those detected and corrected using the parity bits, those detected requiring redisplay, and misread codes.

### 3 RESULTS

The system outlined above was tested using two sets of experiments. The first were conducted under ideal conditions, where there was little variation in lighting. The second was carried out with the augmented desktop described in [6], where dynamic augmentations were present. The experimental setup remained similar in both experiments. The camera was  $\approx 1$  m above the device and the intensity values used in  $\vec{p}_d$  were extracted from a gray scale image with 8 bit pixel depth captured using a Trust 380 USB Spacecam with a frame of 30 Hz. The vector  $\vec{p}_d$  contained 15 elements with buffers of size  $k = 2$ . The device used to generate the binary codes was a Dell x51v Pocket PC with Windows Mobile 5.0. The Dell x51v displayed codes on the request of the observing computer. Each of the sixteen Hamming codes was used with equally frequency and the results are summarized in Table 2. The values of  $\lambda_1$  and  $\lambda_2$  were 0.125 and 0.25 respectively. This reflects our observations that the handheld's screen was able to dim almost instantaneously, while brightening the screen took significantly longer.

The results in Table 2 show that the device was correctly identified 95.5% of the time under ideal conditions and 93.7% while being used with dynamic augmentations, exceeding the 90% accuracy reported in [3] under similar condition. The parity bits were only needed 5 times to correct classification errors. The uncorrectable errors detected were mostly due to the header and footer not being located with sufficient robustness. The remainder of the errors were due to state changes of the screen occurring during camera capture events. When the binary code being displayed is a series of alternating highs and lows the resulting sequence can not be classified because multiple intensities are between the  $\tau_{low}$  and  $\tau_{high}$ . Despite the presence of errors the robust construction of the code prevented falsely identified devices that could cause data to be distributed to the wrong device.

### REFERENCES

- [1] H. Aoki and S. Matsushita. Balloon tag: (in)visible marker which tells who's who. In *Intr. Symp. on Wearable Computers*, pages 181–182, 2000. IEEE.
- [2] H. Kato, M. Billingham, and K. Tachibana. I. Poupyrev, K. Imamoto. Virtual object manipulation on a table-top ar environment. In *Proc. of the Intr. Symp. on Augmented Reality*, pages 111–119, 2000.
- [3] K. Miyaoku, S. Higashino, and Y. Tonomura. C-blink: a hue-difference-based light signal marker for large screen interaction via any mobile terminal. In *Proc. of the Symp. on User Interface Software and Technology*, pages 147–156, 2004. ACM.
- [4] L. Naimark and E. Foxlin. Encoded led system for optical trackers. In *Proc. of Intr. Symp. on Mixed and Augmented Reality*, 2005. IEEE.
- [5] W. Newman and P. Wellner. A desk supporting computer-based interaction with paper documents. In *Proc. of the Conf. on Human Factors in Computing Systems*, pages 587–592, 1992. ACM.
- [6] G. Reitmayr, E. Eade, and T. Drummond. Localisation and interaction for augmented maps. In *Proc. of Intr. Symposium on Mixed and Augmented Reality*, 2005. IEEE.
- [7] B. Ullmer and H. Ishii. The metadesk: Models and prototypes for tangible user interfaces. In *Proc. of the Symp. on User Interface Software and Technology*, pages 223–232, 1997. ACM.