Center for Embedded Computer Systems
University of California, Irvine

Study of Android-Based Multi-Projector Mobile System

Apik Zorian

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-2620, USA

{zoriana}@uci.edu

CECS Technical Report 12-05
July 17, 2012

Table of Contents

List of Figures

Abstract

Study of Android-Based Multi-Projector Mobile System
By Apik Zorian

University of California, Irvine, 2012
Professor Fadi Kurdahi, Chair

As mobile technology continues to increase its role in the world's day-to-day activity, hand-held devices such as smart phones and tablets are becoming increasingly commonplace within corporate offices and meetings. These devices contain sufficient storage capacity and operating units to handle presentation material, yet their mobile design leaves little space to incorporate projection. Pico projectors hold a promising role in the future of mobile display, allowing imaging projection from handheld devices to be accomplished on a level small enough to be integrated on the ever-minimizing mobile platform.

The Graphics ICS department at UC Irvine has made great strides in working with pico projectors, including the iCVideo project, which uses these projectors to achieve an interactive collaborative video through a distributed system [9]. Using four separate sets of BeagleBoard-xM development boards, pico projectors, and imaging cameras, a synchronized video playback is achieved on a tiled display. A distributed registration algorithm is used for spatial calibration of the system, while the video frames are synchronized through a visual feedback-based video synchronization.

The goal of this research effort is to improve mobile use of this system by applying the system onto two Nvidia Tegra tablets. To achieve this goal, a detailed model was established to replicate this system on an Android platform, using Nvidia's Android Software Development Kit to accomplish the necessary steps in creating this display. Design variables included camera sensitivity, projection input, and image blending. Improvements in power and frequency usage were also analyzed.

Chapter 1: Introduction

The growing possibility of projecting images from mobile devices has lead to the emergence of a variety of applications, for pico projectors, from projecting images or videos stored on a device to even utilizing hand and finger tracking to control an operating systems via simple gestures captured from a camera. Currently, pico projectors' limitations in resolution and brightness have proven to be the biggest setbacks when comparing them to standard projectors [1]. The idea of utilizing multiple devices to collaboratively display either a tiled or superimposed display is a viable solution to this issue, as the combined projection would allow the power of multiple pico projectors to be used in achieving a seamless display. For this system to be practical on a mobile level, calibration and synchronization must be achieved on the devices, rather than relying on an external system.

1.1 Background and Motivation

From the academic community to corporate businesses, professionals are witnessing the dependence on computers slowly start to be substituted with tablets, smart phones, and other mobile devices. The same meetings that merely 10 years ago saw paper and pen begin to be replaced with PCs and Macs, have now fully embraced the desertion of computers for the simplicity of iPads and Kindles. It's no secret that the world is starting to welcome the fact that tasks such as web browsing, e-mail, and other jobs we once relied on our computers to do, can be achieved just as well by using today's handheld devices. They provide a fully-functional, lightweight alternative to the personal computer that generally proves to be more aesthetically pleasing as well. As mobile devices become more commonplace in the professional society, the need for this technology to incorporate projections for instructional and collaborative spaces has never been more imminent.

Figure 1.1: Social Usage of Pico Projector [2]

Today, most people's perception of projectors does not necessarily fall under the "mobile" category. The common image is a loud fan, numerous cables, and perhaps a remote to flip through a PowerPoint slideshow. Yet by achieving a lightweight, handheld marriage of these two technologies, the possibilities of real-time data sharing become endless. As strides are being made to find an effective way to incorporate the two technologies, pico projectors are emerging as a viable solution for image projection on handheld devices. From portable pie charts that can be updated during meetings, to interactive whiteboards that allow students to solve algebra equations using hand gestures, pico projectors and mobile devices have the potential to revolutionize how media can be shared in every-day life. Figure 1.1 shows the use of pico projectors in social scenarios.

1.2 Overview of Pico Projectors and Existing Research

Given that today's tablets and smart phones are more than capable of handling and storing significant amounts of data, pico projectors are providing a means of displaying this information without adding overhead of display screens. These projectors consist of a miniaturized hardware and software, allowing digital images to be projected from the device onto any surface, such as a desk or wall. When mounted, this compact system goes through a series of steps to efficiently manipulate and display the

image, operating with the same ease as a conventional projector. Figure 1.2 shows how a device can be connected to a pico projector and used as a source for the display.

Figure 1.2: Device Using Pico Projector to Display Image [3]

Research efforts from students in the Graphics ICS department at UCI have given light to the potential of mobile projections, using distributed multi-projector systems to create collaborative displays. However, there is a critical issue that must be resolved to effectively accomplish these systems onto handheld devices: For such a distributed system to operate on a multi-projector design, the steps for registering and synchronizing the devices must be achieved on the device without depending on an external source, so that the "mobile" aspect is not compromised.

1.3 Research Goals and Approach

This report presents a unique application of mobile collaborative image projection with real-time registration and synchronization. To operate as a fully-mobile system, calibration and synchronization of the devices must be carried out in real-time. This can be achieved by making all of

these steps executable on a single, hand-held device that is independent of exterior systems and support. The mobile devices used in this effort are two Tegra development boards by Nvidia, whose ultra low power graphing processor unit (GPU) allows real-time image processing on the device, incorporating energy efficient technology to accomplish high quality performance at minimal power. With this basis, the objectives of this research effort include the following goals:

1) Achieving registration for each mobile device in the projector system
2) Synchronizing the devices to guarantee a seamless resulting display with no delay
3) Displaying the corresponding frames in the projection to achieve a flawless image

This report consists of six chapters, including the background, motivation, and objective of the research effort, all of which are covered in Chapter 1. Chapter 2 contains related studies and information on mobile device projection, exploring projects done by other researchers on similar topics. This includes a review of recent achievements by ICS image processing projects, as well as an in-depth look at other research done in this field. Validation of the proposed model for real-time image projection from two Tegra tablets is presented in Chapter 3.

Chapter 4 covers the steps for developing the project detailed in Chapter 3 on the Android platform. Experimental results are presented in Chapter 5, as well as suggestions for prospective research on the topic. A detailed description of the benefits of low-power performance on the Tegra tablet is also presented in this chapter. Finally, Chapter 6 provides a summary of the project and some concluding remarks.

Chapter 2: Literature Review

To understand how a distributed mobile projection can be achieved, we must first evaluate the building blocks of such a system, analyzing the most efficient and relevant components to make up this display. Aside from the hardware, this section also analyses existing work in the utilization of distributed systems for the collaboration of multiple mobile devices. Student research in UCI's Graphics ICS department has made great progress in studying the area of mobile projections.

2.1 Pico Projectors

A pico projector system contains five main components: scanning mirrors, a combiner optic, a battery, electronics, and laser light sources. When the system is fed an image to display, it first translates the image into an electronic signal which, in turn, is used to drive laser light sources. These light sources vary both in color and intensities and each travel down differing paths until they reach the combiner optic. Here, these light paths are joined into one collective light path which essentially serves as the color pallet. With the required colors available, the mirrors are then used to copy the image pixel-by-pixel and finally project it onto a surface [4]. Figure 2.1 shows an illustration of how these components interact. The pico projector system can essentially be viewed as a SoC, since the entire system is compressed onto a single, tiny chip that effectively controls the image display process.

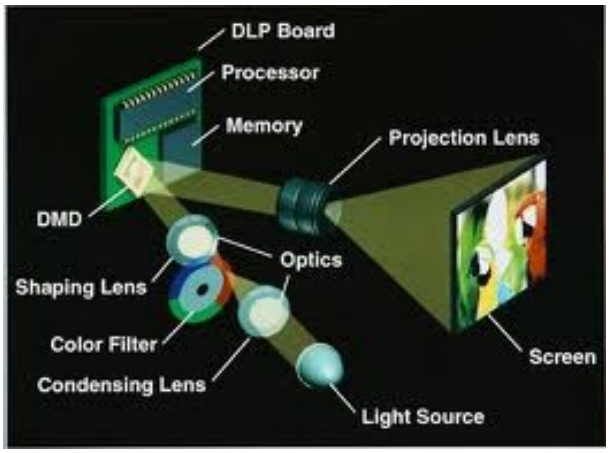

Figure 2.1 Layout of pico projector system components [5]

Though their small size makes them ideal for use with handheld devices, pico projectors are currently under scrutiny for their insufficient display brightness. This deficiency makes them impractical for daily use, as their displays cannot be effective in normally lit rooms. There have been recent efforts in improving the resolution of current pico projector chipsets, such as Texas Instrument's DLP technology. The DLP's design incorporates an array of nearly 2 million microscopic mirrors onto a single chip to produce a high quality projection [6]. By shifting our focus to the mobile level, one unique approach can be considered: Combining several pico projectors to collaboratively create one image, achieved as either a superimposed or tiled display. Superimposition would require multiple projectors to display the same image on the same location at once, with the assembly of projections on top of each other achieving a stronger image than if just one were used. On the other hand, a tiled display would require each projector to display a portion of the image, combining the projections to create a larger display. The collaborative projection would be done through a distributed system, where each component would independently identify its role in the display, format its portion of the end image, and project onto its designated location, fusing with the other projections to achieve a seamless result.

2.2 Distributed System

Large multi-projector planar display walls are common in many visualization applications. With mobile devices, rather than using one pico projector to display an image, a wider display can be achieved by utilizing a tiled distributed system, where a group of devices project fractions of the image and combine to form one collaborative display. For this to be achieved, devices must be implemented in a distributed system, where the image is divided between each device and respective portions and positions of the image are processed by the devices themselves.

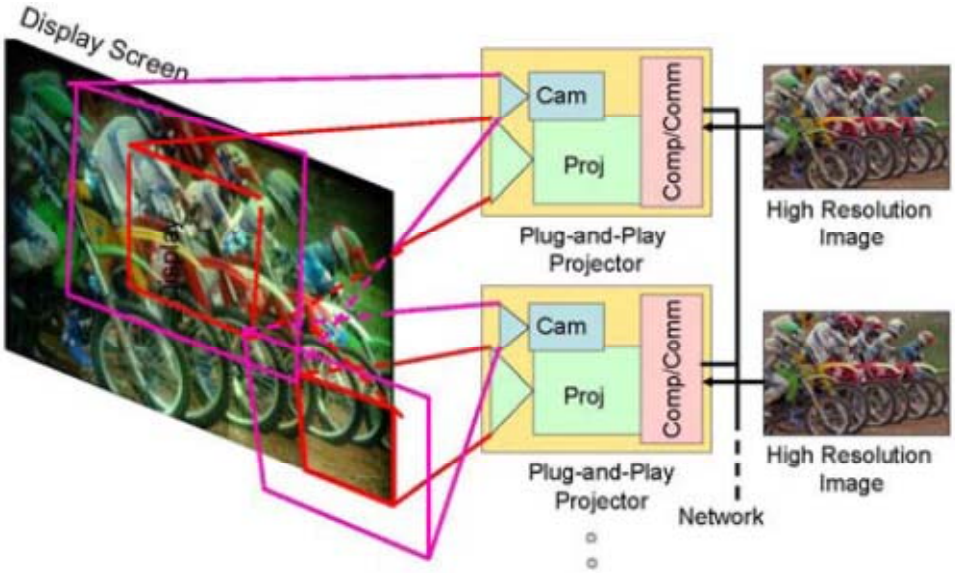

Figure 2.2 Architecture of Distributed Multi-Projector System [2]

There are a number of steps that must be achieved within the mobile system to accomplish this distributed display, beginning with registration. In this step, each device must first register itself in terms of what portion of the image it will be displaying, the sequence of frames it will be displaying, and the coordinates for the location of its projection. Secondly, synchronization must be achieved among the devices to guarantee a flawless display. This entails a run-through of the frames that each will be displaying and scheduling of projections to account for delays. Finally, formatting of each image must be done so that each component is displaying the correct portion and size of the image.

Auto-Registration

In a study conducted in UCI's Graphics ICS department, mobile device calibration was achieved through a distributed multi-projector system. An assembly of pico projectors was used to build a large scale and high resolution tiled display of plug-and-play projectors (PPP) [2]. The objective was to make the display scalable to multiple projectors and users, allowing painless collaboration between varying numbers of participants. For this to work, each component of this distributed system would have to self-calibrate and reconfigure itself when necessary. The approach was to design a scalable interaction paradigm that could scale to multiple projectors; thus, distributed geometric calibration, interaction, and rendering algorithms were developed to assure an unblemished image when these projectors were combined.

By mounting individual cameras to each projector, the distributed calibration algorithm used visual communication to achieve registration among the projectors. Each projector was able to identify its position in the end display, the number of projectors that made up the display, as well as other geometric calibration factors. This feedback helped assure a faultless display, as each projector was able to recognize its parameters and avoided overlapping with its neighbors. The projector system also used augmented infrared illuminators and filters for experimenting with gesture detection. The setup of the PPP system can be seen in Figure 2.3.

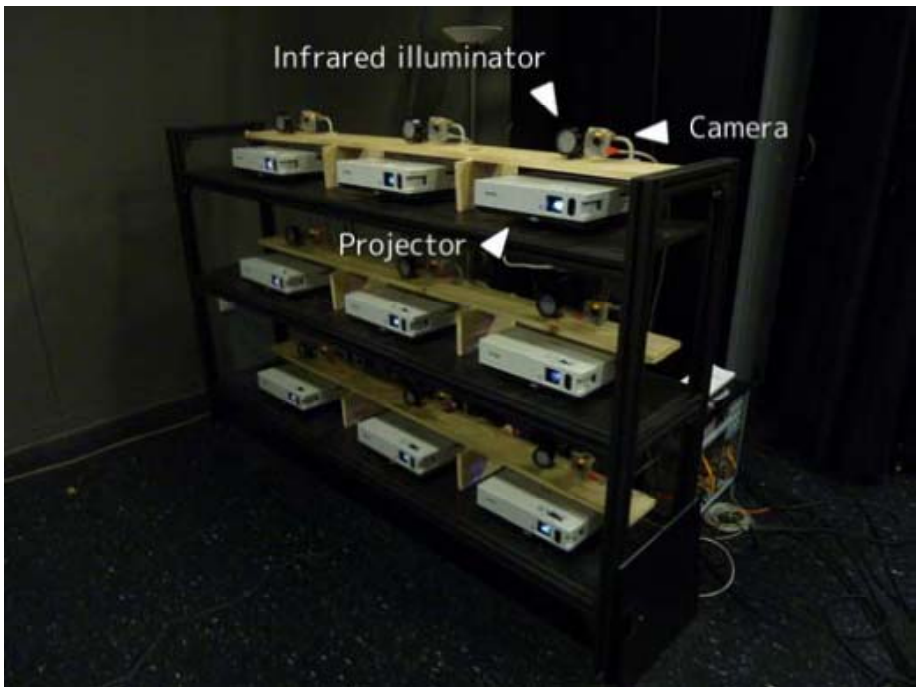

Figure 2.3 Setup of the PPP network [2]

This novel calibration method allowed each projector to use a single pattern made of specially augmented Quick Response (QR) codes to discover the display configuration and self-register simultaneously in constant time. A QR code is a 2D barcode which can be embedded with a certain number of bits of data, depending on its size. By using the cameras to achieve visual communication, each projector was able to detect its neighbors. The QR codes were encoded with the IP address and port of the respective projector, the location of the QR code, and its size. The codes were also augmented with Gaussian blobs embedded in the surrounding region of the code [7]. A blob is commonly known as a detection point in an image which, in contrast to its surroundings, differs in color or brightness. The blobs surrounding the QR codes were used to identify correspondences across the projectors and cameras for calibration. Figure 2.4 shows a QR code in standard form and augmented with Gaussian blobs. Using these registration techniques, the projectors were able to discover their neighbors, the total number of projectors in the display, as well as its own coordinates in the array. Once this was done, the projectors could self-register with the rest of the system, producing a seamless image.

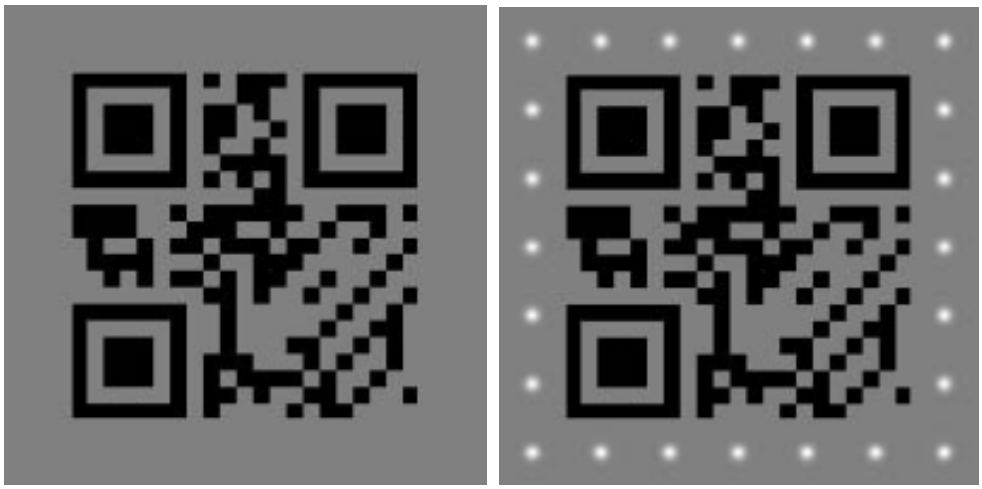

Figure 2.4: QR code in standard form (left) and augmented with Gaussian blobs (right) [2]

Synchronization

The concept was introduced to use embedded cameras in capturing the projection area to register frames spatially. More recently, one of Dr. Majumder's research groups began experimenting with visual feedback to achieve collaborative video playback on mobile projectors [8]. The idea was to use the calibration algorithms discussed above with a novel visual feedback- based video synchronization algorithm. This would allow an ensemble of projector-enabled mobile devices to collaboratively present a full video stream, consisting of multiple sub streams, each streamed to a different mobile device comprising the assembly.

Single patterns of specially augmented QR codes were used for each projector, as before. In a distributed scheme, each unit was equipped with a camera and independently ran the synchronization algorithm through the visual feedback from its camera. The display created a 19" diagonal display which could easily come in the field of view of camera. Visual feedback from embedded cameras facing the projection area was used to both register frames spatially and synchronize them temporally. The significance of the synchronization step was that it did not use any wireless network infrastructure, making it independent of network congestion and connectivity. Each projecting unit would autonomously adjust itself to achieve synchronization, without the need for a communication network

between the boards. This meant that synchronizing could be achieved without depending on a wireless network, making the system more mobile-friendly in cases where devices cannot access a network. It also relieved the system of relying on a mobile network that is already heavily congested, as the requirement of 30 frames per second for video synchronization would require large data transmission.

2.3 iCVideo Research Project

One of the current research projects that is experimenting with distributed multi-projector systems is the iCVideo Research Group. Dr. Majumder's team uses a group of pico projectors set in tiled a arrangement to generate higher resolution displays when compared to what a single mobile projector could offer. In this case, the QR codes are encoded with the frame number that each device will display, as well as a pattern of blobs that surrounds the code, which are used to determine spatial calibration [9]. Using the distributed calibration algorithm developed in [3], each device captures the QR code and detects the position of the blobs, storing this information for its potential projection. Figure 2.5 shows the QR codes being projected using pico projectors.

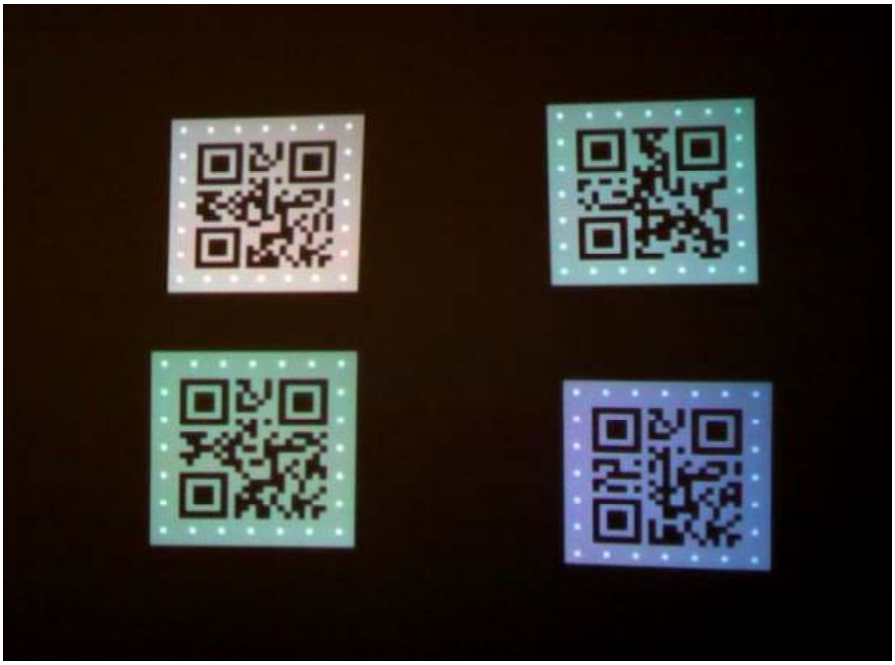

Figure 2.5 QR Codes Displayed For Registration [9]

A set of four TI BeagleBoard-xM development boards serve as the target mobile devices in this system. The BeagleBoard-xM is a single-board computer, about 82.55 by 82.55 in size, with 512 GB of RAM and a CPU clocking at 1 GHz. The board contains a 4 port USB hub, Ethernet jack, and a DVI-D that is HDMI compatible, allowing the simple connection to peripheral pico projectors [10]. Figure 2.6 shows the layout of the system with the boards connected to their respective projectors and cameras.

Figure 2.6 Setup of iCVideo Multi-Projector System

The main advantage of this project comes from the integration of the registration and synchronization steps. Utilizing the aforementioned registration algorithm, each board projects a set of QR codes augmented with Gaussian blobs and the cameras capture and decode them to determine the display configuration. However, since both registration and synchronization steps are carried out before playback, the two can be combined using empty channels in the QR code. Here, frame number information can be embedded into available areas, thereby integrating registration and synchronization into a single process.

Chapter 3: Validation of Concept

The following chapter details the hardware and software translation of the iCVideo project's multi-projector system implemented through an Android mobile system. The proposed design uses two pico projectors and a pair of Nvidia Tegra 3 developer boards. The project will achieve geometric registration in real-time and will eliminate the need to use BeagleBoards, which require external computer support for calibration and synchronization. The new design allows most of the hardware used in the iCVideo project, including the boards, cameras and their external machines, to be scaled down to a Tegra board and pico projector.

3.1 Hardware Translation

Our group was fortunate enough to receive two Tegra 3 Developer tablets from Nvidia for our research. The tablets, which feature Nvidia's next-generation Tegra processor Kal-El, come equipped with a front-facing camera, two back facing cameras, and sensors for measuring acceleration and orientation. Figure 3.1 highlights the camera and sensor locations on the tablet. The Tegra eliminates the need to use external cameras by providing the two rear facing cameras integrated into the tablet, allowing a portable medium for capturing images. The sensors become useful when programming the per-frame camera parameters within FCAM, which we will discuss later in Section 4.5.

Figure 3.1: Camera (a) and Sensor (b) Locations on Tegra Board

The tablet also comes with an SD Card Slot and Port Expander, a utility extension for the board that contains ports for USB and HDMI connections. The SD card slot and USB port can be used to load data to the board, such as the QR codes and frames that will be projected. The HDMI port allows easy mounting of an external display source, which in our case would be the pico projector. There are volume control buttons and a headphone jack on the side of the board, as well as a micro USB port [11].

Using a USB A-to-micro-A/B cable, the tablet can be connected to an external computer through a USB port, allowing locally developed applications, or apps, to be loaded onto the board. Finally, a debug board with a connector is included; this component connects to the board's processing circuitry and can be used to implement a forced recovery mode. The debug process requires physically removing the back cover of the tablet to connect the debug board and is necessary for tasks such as updating the system software. Figure 3.2 shows a detailed illustration of the board's supplemental technology.

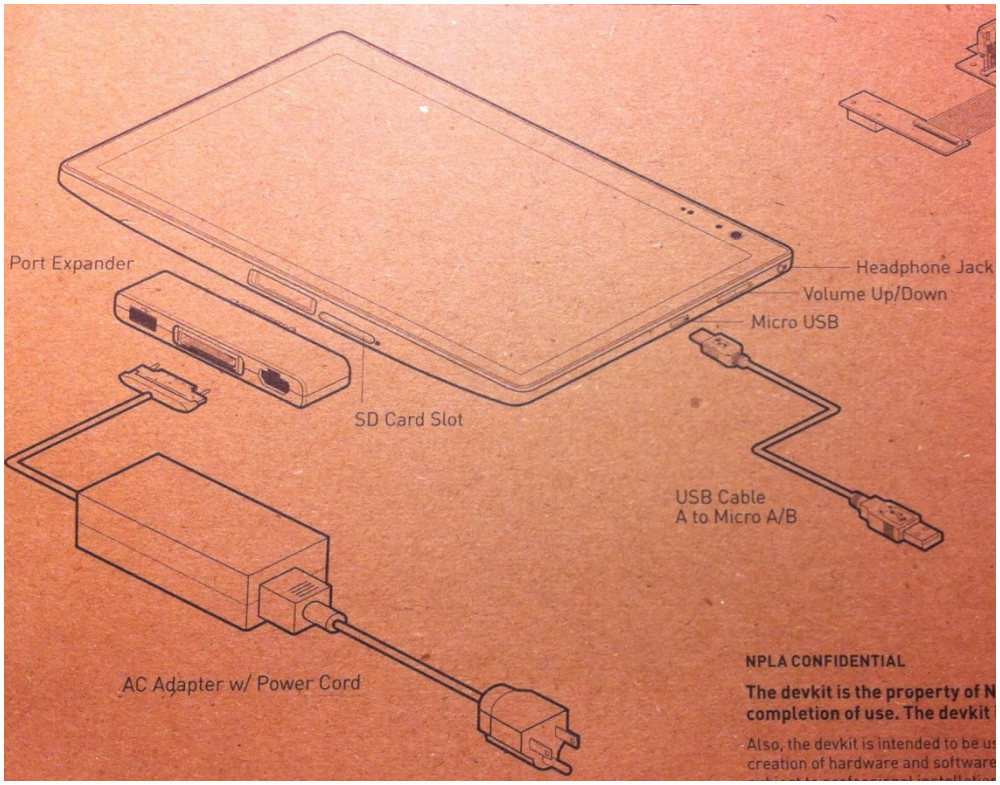

Figure 3.2: Illustration of Nvidia Developers Kit Technology

3.2 Software Translation

The Tegra boards operate using Android OS; therefore, the programming of the device is done almost entirely through an Android application programming interface (API). Applications were written in a variety of languages including Java, C++, and XML, using the Eclipse Integrated Development Environment (IDE). Nvidia also provided access to Android's Software Development Kit(SDK), detailed in section 4.2, which gives programmers the ability to develop Android applications for Android-powered devices. Along with the sample applications and references included in the SDK, an Android Native Development Kit (NDK) is also available for Nvidia developers to download. The NDK allows portions of an application to be programmed in native-code languages, such as C and C++. This toolset provides a simple means to reuse existing libraries written in these languages, though using native code in Android normally results in increased application complexity [12]. Thus, developers should resort to coding in

native languages only out of necessity, not because of a preference to C/C++. CPU-intensive operations such as signal processing and physics simulations are viable candidates for using native-languages, as they are generally self-contained processes. Installing the NDK requires modifying the environmental building path for whatever API is being used. By doing so, the environment immediately knows to use the NDK when compiling the native-portion of the respective app, executed every time the app is built. For the NDK build to work, a version of GNU Make 3.81 or later must be downloaded, as well as Cygwin 1.7 or higher for Windows users. Without these programs, the NDK will not compile the native-components of the app.

For this design to work, the application would be responsible for running a tedious amount of tasks: While projecting the QR code from its respective tablet, the application would simultaneously capture and decode the displayed QR codes, register the tablet's frame sequence, and project the stored images that coincided with the QR data. The image would be scaled and positioned based on the coordinates of the blobs surrounding QR code. By utilizing image processing techniques libraries such as OpenCV, these coordinates can be deciphered to allow the application to read the edges of the displayed code and project the image onto the respective area. Open CV is a library of programming functions focused on real time image processing. With its latest updates, Open CV has become a staple in the mobile graphics industry in developing applications that utilize real time computer vision functions. It is a cross-platform library that functions on Windows, Android, Linux, Mac OS, and iOS, to name a few. The library, originally written in C, is portable to platforms such as DSP, and has had wrappers developed for other languages such as Python and Java [13]. Through these signal-processing techniques, OpenCV could be utilized in detecting the Gaussian blobs surrounding the QR codes and storing the coordinates for the frames that would eventually be displayed.

To control the activity of the Tegra's camera, such as managing its capturing action and parameters, an open-source C++ API called Fcam was utilized. Through its integrated libraries, FCam

allows full low-level control of all device cameras and parameters that are available on a per-frame basis. These manipulations include reconfiguring the camera's auto-focus routine, synchronizing the lens and flash, and most importantly, the ability to capture bursts of images with varying parameters [14]. With this last feature, the device could be programmed to capture a large amount of images at once and send them to the QR reader for decoding. The most important advantage, however, comes from the fact that FCam can be run as a separate thread on the device without having to be open. This means that while the QR codes are being displayed, the FCam is able to capture images successively without needing to occupy the display.

Chapter 4: Android Design

4.1 Experimental Parameters and Strategy

Before beginning the preliminary steps of the design, a setup for the display was first established. Two pico projectors would be mounted, each facing a wall and each connected to a Tegra tablet. Registration would then be accomplished through the following steps:

(1) Each tablet will display a series of QR codes, with each code consisting of its frame number, as well as a border of blobs for detecting position.

(2) The program would activate manually from each Tegra tablet to display these frames. While these frames are being displayed, the program will also access the device camera and command it to capture a burst of images, saving the captured QR codes as .jpg files.

(3) The saved photos will then be decoded using a QR reader source. The program will also detect the blobs around each frame and store the coordinates of each frame.

(4) Once all the frames have been QR codes, the tablets will display their portions of the film, based on the frame numbers detected. The frames will be displayed in the positions relative to the blobs detected during registration.

4.2 Android Software Development Kit

Several months were spent learning how to develop Android applications and familiarizing with the API. I chose to use the Eclipse IDE as it was readily available on my machine and presented a relatively simple interface for development. Along with all of the hardware that came with the prototype tablet, Nvidia also provided developers with the Android SDK. By accessing the development kit through Eclipse, I was able to utilize the Eclipse environment to not only create Android applications, but to test them through either an emulator, known as an Android Virtual Device (AVD), or by simply installing them on the tablet from my PC. The SDK provides a program called AVD manager, which

allows local downloads of AVDs for any version of Android's 16 different API operating systems. Similarly, the SDK provides a program called SDK manager, which downloading of SDK platforms and sample applications for all of the different API operating systems, as well as API-specific USB drivers for mounting devices. As the prototype tablets were running on Android 3.2 OS, I accessed the necessary downloads and began creating and testing some basic applications.

The SDK also provides instructions on how to run some of the more basic samples and breaks down the different Java components of an Android application. Through these samples, I was able to learn the structure and programming of the elements that comprise an application, such as formatting of the source code, building views for application's User Interface (UI), assigning and accessing Android IDs for different components in the program, organizing the layout of an application through XML, and many other basic functions. With each sample came a new learning opportunity that would come in handy when in the long-run of the project.

Above all, the process of learning how to use the API presented a chance to refresh my skills in Java programming, as well as the opportunity to learn additional languages, such as XML, that proved critical in developing Android applications. Although my experience in Java had been outweighed by my knowledge of C/C++ programming, I was able to build on my verdant understanding of Java programming, learning how to operate using different classes and different constructors throughout the program. Additionally, Android has a large variety of unique classes specifically targeting Android application development. The SDK provides an online catalog of these various classes, as well as instructions and reference examples that describe how to import, initialize, and employ the classes in applications. While the majority of an Android application's core tasks are written in Java, developing the application's layout interface required programming in XML. Once I felt comfortable with developing applications in the Android API, I was ready to move on to the beginning steps of the project.

4.3 Projecting the QR frames

The first task was to experiment with simple projections from the Tegra using a pico projector. As described in section 3.1, the Tegra board comes with a Port Expander which, when connected to the dock connector, allows USB and HDMI connection to the board. Using an HDMI cable, the board was connected to a DLP V2 pico projector and effectively displayed tablet's home interface. Test photos and videos were saved to and opened on the tablet and as long as it remained connected to the projector, the display on the tablet's screen would successfully be displayed from the pico projector. A set of 100 QR code were created using an online QR code generator and saved to the tablet's SD card. Each QR was encoded with a number from 100 to 199, with each code translating to a single frame in a 100 frame display. The SD card provided over 13 GB of free space (more than enough memory to store a set of .jpg files that, each at around 1.5 KB in size) and was loaded easily through the SD card slot.

Next, I developed a simple app that would display each QR file(QR100.jpg – QR199.jpg) with a 1 ms delay between each file. Initializing an array in the application's primary activity, each file was loaded from the SD card and stored into the array. Then, a timer method was created to start counting once the app was launched. By importing the android.os.timer library, a timer class was easily developed as most of its methods were detailed in the SDK. When launched, the app would create a bitmap which would initially display the first file in the array of QR images and simultaneously would start a timer that would count down for 200 milliseconds (ms). After 2 ms had passed, the next file from the array would be loaded onto the bitmap, then replaced by its preceding file after another 2 ms had passed. This would go on until the timer ran out, resulting in all 100 QR codes being displayed.

4.4 QR Reader

One of the many advantages of programming in Android is the abundant access to existing open source projects. Android applications can be downloaded directly to a device through the Marketplace or as a .apk file from an external source. However, for most application developers who have easy

access to an API, a source folder of the application serves as the primary option for truly understanding the different components of the app, how they function, and the various manipulations that can be executed. A source folder generally contains a variety of directories that make up the application, such as SRC (containing the main source code for the program), GEN (containing the R.Java file that handles the ID's of the project's variables), RES (includes the .xml files for the layouts of the application), and LIBS(containing the various libraries referenced throughout the application). Once the source folder is saved to the local desktop, one can use a development environment such as Eclipse to create a new project from the existing source. From here, the application is free to be edited at will and can be loaded to a device to test or simulated using an Android emulator.

For the QR reader, I was able to access the source code for ZXing, a popular Android QR reader application. While more basic mobile QR reader applications allow decoding of a simple encoded URL and the option to launch the respective website, ZXing has the capability of decoding a wide range of embedded content [15]. E-mail addresses, phone numbers, and even calendar events that are encoded can not only be displayed after reading, but the user can then save this information to their device by simply asking the program to launch the native address book or calendar. By studying the source code, I was able to understand how ZXing was able to differentiate between these contents and launch the respective applications. Although this application did satisfy a number of needs for my project, there were many other layers that would need to be included in my application, such as projecting the frames and detecting the blobs. One idea was to use an Intent in my native application to launch ZXing, then switch back when the QR reading was done. However, launching an Intent and switching applications back and forth would consume time and make the process more complex, as the information decoded would be difficult to transfer from app to app. Thus, I decided to integrate the ZXing QR reader into my display application, allowing all of the actions to be done without the need to switch between applications.

There was, however, one critical concern that promised to give issue in the long run. When the device would be mounted to a projector, the projector would display only what showing on the tablet's screen. Thus, if an app was running on the board and displaying from the projector and one decided to switch to another application, the projector would cease to display the previous application and would switch to the new one. This would be a significant obstacle in the registration process, as the displaying and decoding of the QR codes was meant to be done simultaneously. With each of these steps needing full access of the screen to operate, the two could not be open concurrently. Thus, an approach would need to be taken to access the device's camera and control its capturing activity without the need to exit out of the QR display.

4.5 Capturing – Fcamera

The original plan for the registration step was to have the display, capture, and decoding of the QR codes simultaneously executed on the device. However, once it was brought to our attention that the Tegra tablet did not have the capability to run one app while displaying a different one on the projector, a new strategy was necessary for accessing the device camera. The solution to this problem came in the form of the FCam API, discussed in section 3.2.

The FCam package contains several examples including the FCameraPro sample application. This app performs capturing and displaying of an image stream, but one could modify it to function in several different parameters. The app contains a Java component for rendering the UI and a C++ component to take care of the capture. Though running the app was fairly similar to previous apps that I had developed, FCameraPro also required installing a new system image (fcam-cardhu-img) to the tablet. This allowed the system binaries to match the FCam library to prevent the app from crashing. Flashing the system image required putting the tablet into debug mode and utilizing the supplementary debug board to apply this final step. Once the app was able to run correctly, I was able to operate FCameraPro as well as the other sample apps and capture bursts of images at different times, frames,

and parameters by manipulating the C++ code. The various sample apps each executed unique functions of the Fcam libraries, but most revolved around capturing images at varying parameters [14]. For instance, one example app would capture two consecutive images, one with flash and the other without, and store these images on an ash shell “/data/fcam”, which could later be accessed.

Chapter 5: Results and Discussion

5.1 Experimental Results

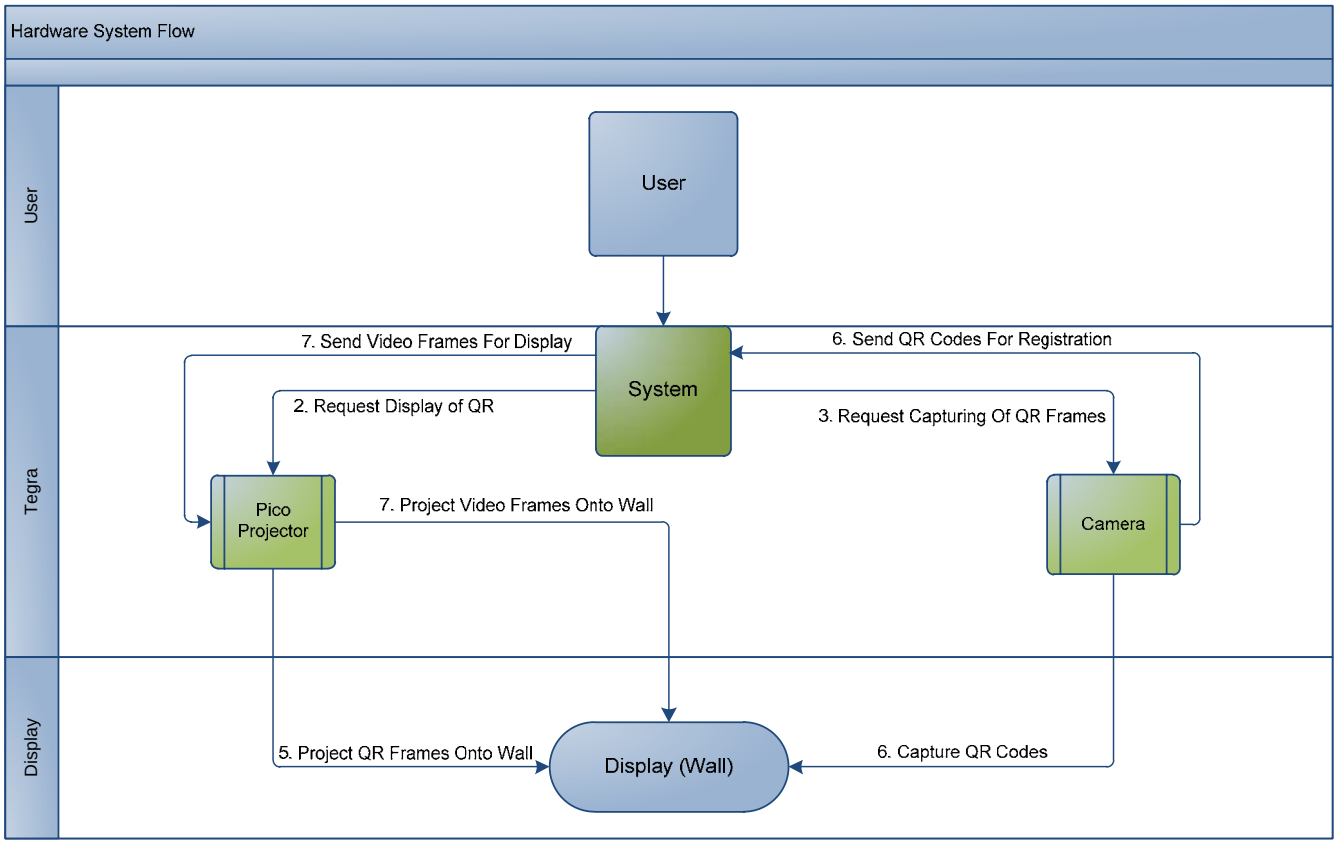

Figure 5.1: Hardware System Flow Chart

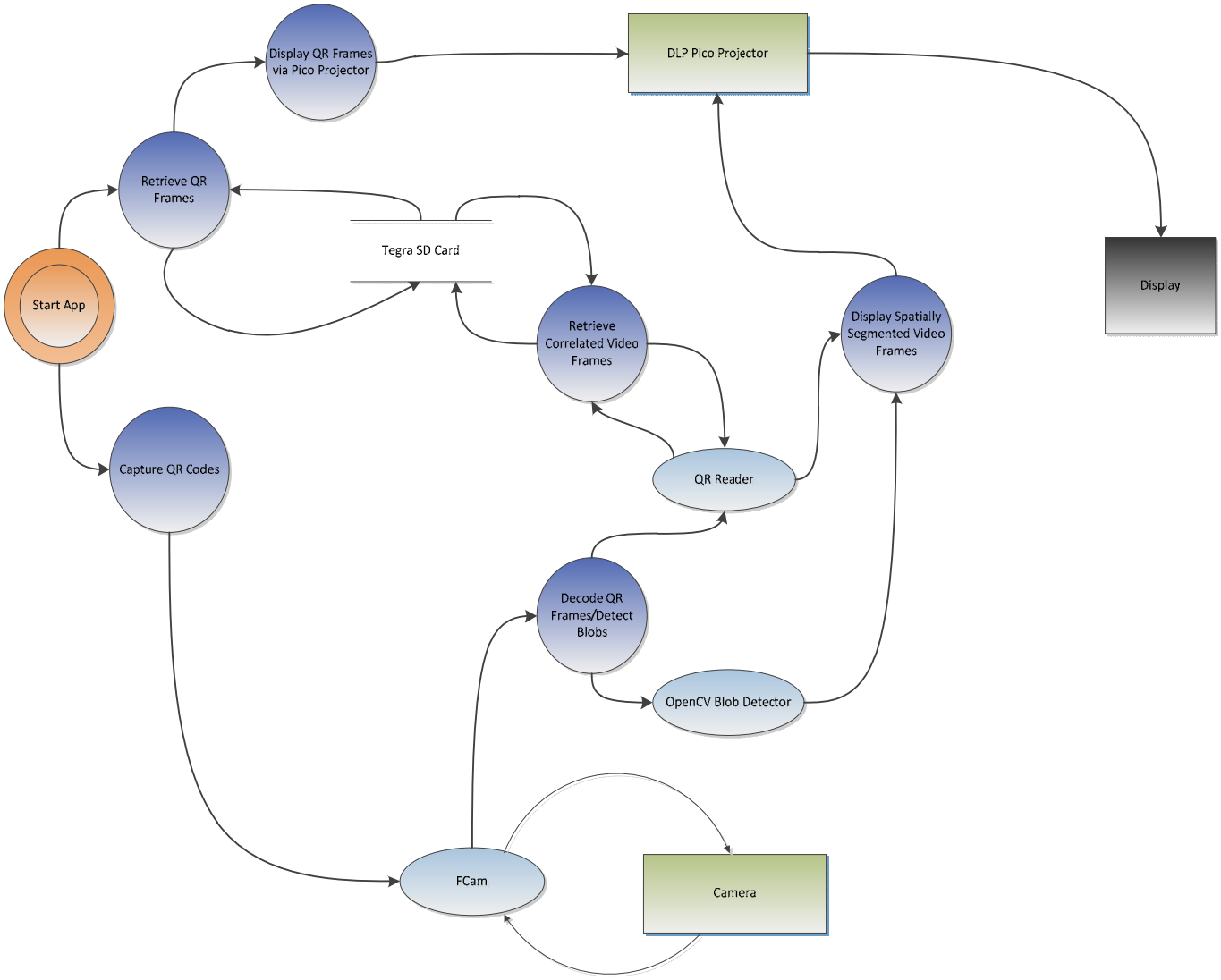

Figure 5.2: Software System Flow Chart

The goal of this project was to take the iCVideo multi-projector design and emulate the system using an Android platform, shrinking the necessary components such as the BeagleBoard and its external units, down to a Tegra board and pico projector. For each of the various steps required to achieve this distributed system, a range of experiments were completed to ensure that the Tegra could indeed function for all levels of the process. From reading and storing of the codes, to registering the devices, to assessing the physical position of the projections, applications were developed and tested to satisfy each requirement. Figure 5.1 and 5.2 show the hardware and flow charts for the steps taken in this project, respectively.

For the display activity, I developed a number of applications that used an ImageView to load stored files onto a bitmap and display the specified images. QR codes were developed for each tablet, embedded with the frame number of the image the tablet would be displaying, as well as its IP address, and augmented with Gaussian blobs for determining the display configuration. These codes were saved to the tablet's memory card and served as the initial files used to test the display. Using a counter, the application would load a new QR code to the bitmap with every ms that would pass, until each of the QR codes had been displayed, as described in Section 4.3.

After a few trials, the program was edited so that the timer would run using the "onResume" class rather than "onCreate" so that the display action would pause if another app was called during the display. To make the process even easier, I added a button below the bitmap that, when clicked, would initialize the timer and start the slideshow. By doing so, the user would not be rushed to have the QR reader running immediately when the QR display was called. With the button acting as a regulator for the timer, the slideshow of QR codes was able to run smoothly on the tablet and displayed perfectly when connected to the pico projector. Figure 5.3 shows the tablet mounted to a pico projector, displaying the QR codes onto a wall.

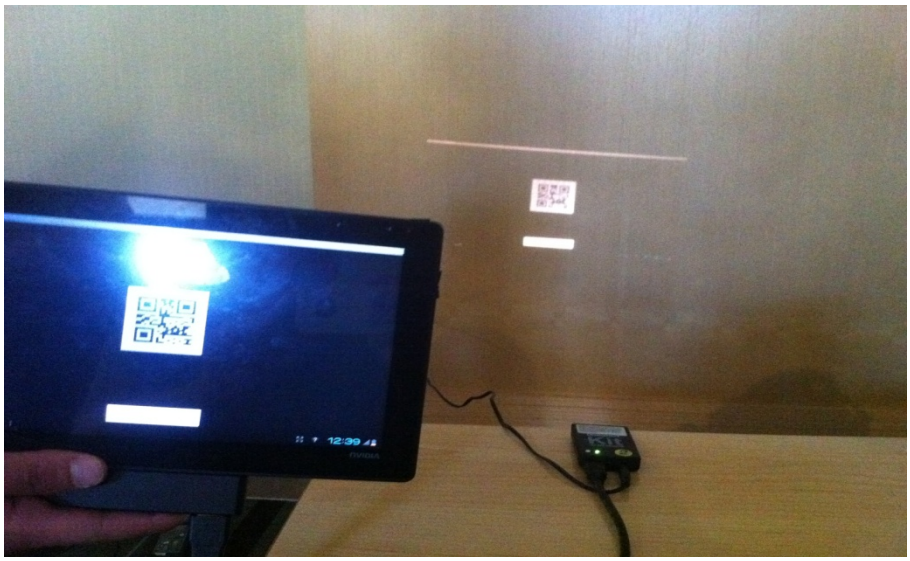

Figure 5.3: Pico Projector Displaying View from Tegra Tablet

Next, I moved onto the QRReader. The reading action of the application was set to "bulk reading" rather than individual codes. This option relieved ZXing from its duties of asking the user how to handle each decoded QR and simply stored the content in an AdapterArray. Using the ZXing open source directory discussed in section 4.4, I designed my display application to incorporate the QR reader by modifying my layout to support a tabular format, with tabbing capabilities to switch between the QR reader and the frame display. Once the calibration process would begin, the convenience of keeping both of these actions within the same application would prove essential for transferring data. A "history" tab would also be available to keep track and store each code that was read. Using ZXing's "Results" class, an array adapter would store the integer value and details of each result from the QR reader and display them in a ListView layout in the History section. Using an inflated layout, the list would automatically add a new line every time a new code was captured. The history list provides a usefully way of keeping track of the series of codes, as well as a mode of initiating the adapter. Figure 5.4 shows the interface of the QR reader.

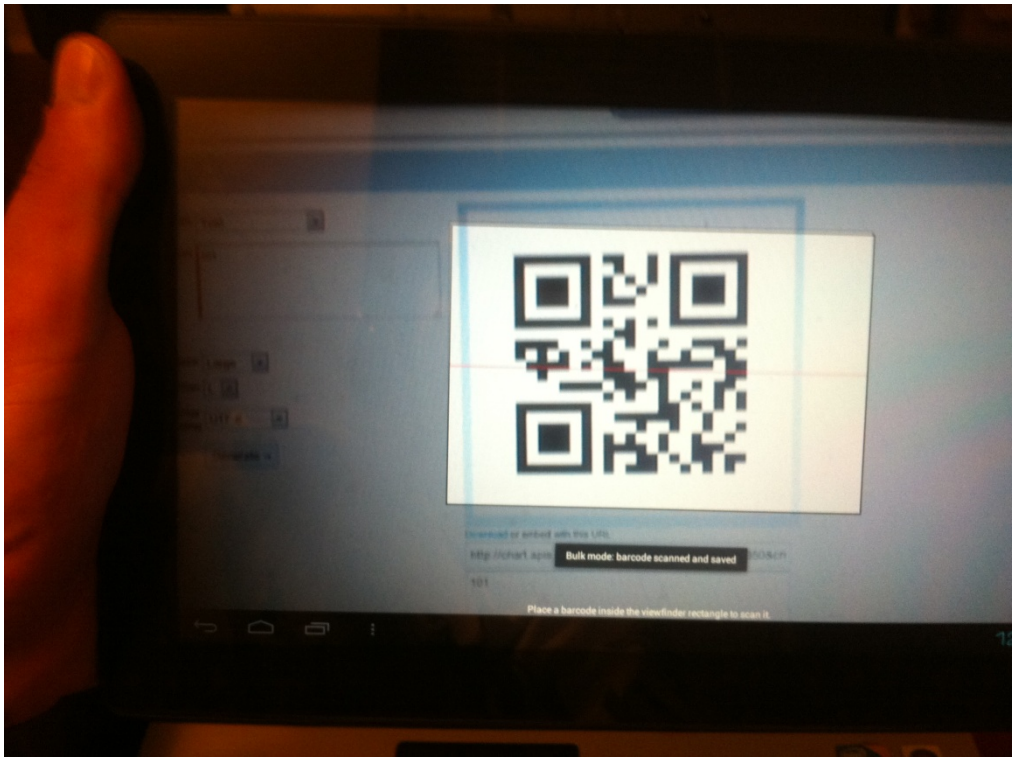

Figure 5.4: QR Reader Interface

In the application's History activity, the AdapterArray was set to store only new QR codes so that if it were to decode the same QR multiple times in a row, it would not store a duplicate copy. Finally, a sharp beeping sound was used in the Capture activity to symbolize when a QR was decoded. Since the QRReader was not required to pause after each reading, this sound acted as a confirmation of decoding after each QR was read. To test these modifications, a series of QR codes, containing several repeats, were displayed using a QR code generator and decoded by the QR reader. The app did not once pause between readings and effectively beeped after each code was read. The History list showed a list of each code that was read and excluded all repeats within the series, proving that the adapter was indeed working and was able to detect and exclude duplicate readings. Figure 5.5 shows the History item list, which has the first 10 frames stored without duplicates.

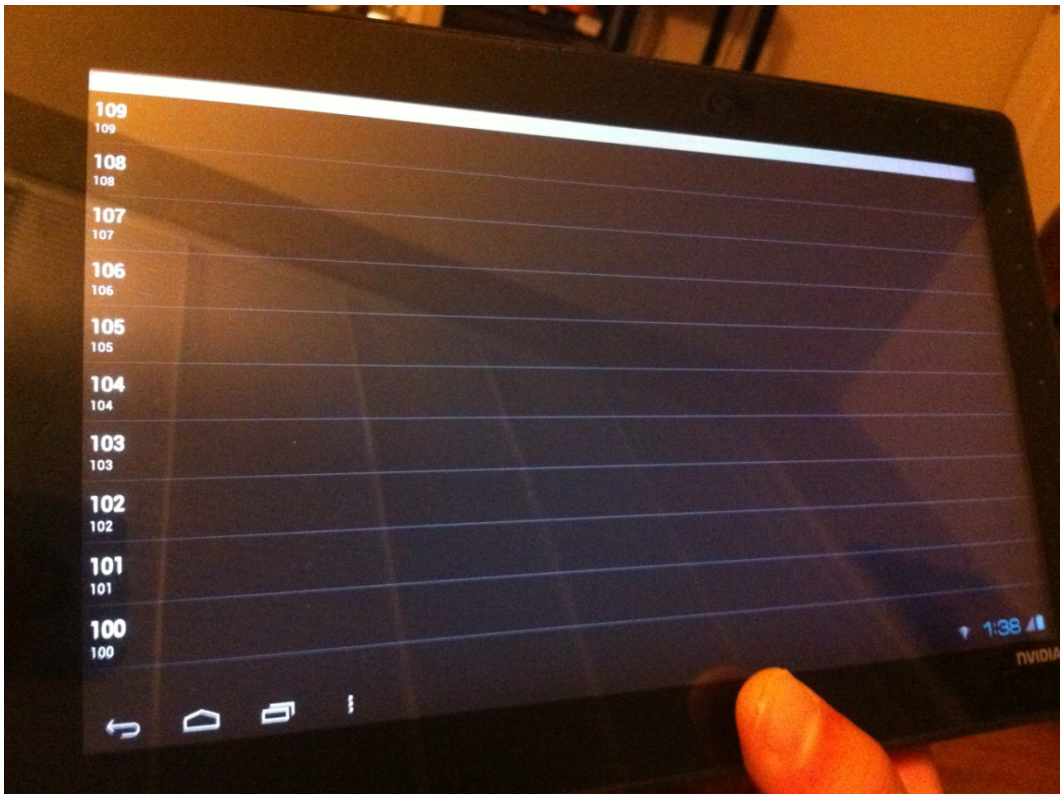

Figure 5.5: History Item List Containing First 10 Frames

Since the end role of the QR codes would be to have each one represent a frame, the next step was to load 30 frames of a generic video clip and assign each value of the QR codes to a specific frame. In the ICVideo project, the video frames used were in RGB format. An RGB image file has the capability of formatting its image, a quality that would be used in the formatting step when each device, once identifying its role in the tiled projection, would modify its projection to include only the specific portion of the display for which it would be responsible. Accordingly, 30 frames from a test video were stored on the SD drive of the Tegra. In the Display activity, these frames were stored in a string array at positions 100-129, with each position correlating to the frame's relative QR value. Once the series of codes were read by the QR reader, the Display activity would retrieve each value from the AdapterArray, convert the objects to integers, and save them to a local array. Once the display was initiated, the application would load frames to the bitmap, with one frame being chosen with every passing ms. The order of the frames would be based on the order in which the QR codes were read. A number of

experiments were run to test this app. QR codes embedded with numbers 100-129 were read and stored to the history. Once the Display was initiated, the app successfully displayed the 30 frames of the movie in perfect order. For good measure, another experiment was run where the order of the QR codes were read in random order. Once again, the application behaved correctly, displaying the 30 frames in the exact order with which the codes were stored.

To resolve the tablet's necessity to only project the app that was occupying the tablet's interface, I turned to Fcam and its accompanying sample applications. The FCam sample applications all contained a C++ component that would take care of the capturing, as well as a Java component to render the UI. Thus, the NDK was referenced and utilized in each of these apps to compile the C++ component; the Java component was almost identical across all the apps. Upon opening one of these applications on a device, the interface would display a TextView, with the screen reading a line such as "FCam Example 1 is starting" when the activity would begin and corresponding lines of text for when the app was running and done, after which the app could be closed by simply tapping on the screen. While this text is being displayed, FCam accesses the camera of the device and begins capturing images based on the quantity and parameters specified in the code. The images are copied to the shell and dumped onto a machine by using the "adb pull /data/fcam" command on a local computer, the local directory being specified.

With the ability to manipulate the UI through the Java code, I created my own Fcam application, FCamQR, that would capture a set of images and, rather than displaying text on the UI, would display a stream of QR codes. This way when the app was open and displaying from the pico projector, the projection would be a stream of QR codes while the device would simultaneously capture the images being projected. Starting with the Java script of an example application that captured one image, I removed the TextView and all of its components, replacing it with the code from my previous QR

display application, including the counter, bitmap, and all the necessary libraries. I also brought in the .xml file from my previous app, assuring that the UI would be synchronous with the Java script.

Configuring the actual capturing activity of the app required editing its C++ component, stored in the app's source folder as a .cpp file. For my app, I decided to apply the .cpp code of one of the examples that captured 1 photo and alter it so that when running, the app could capture a variable number of frames with differing parameters. These parameters, which included width, height, exposure, and gain, could be set at different values for each frame captured. However, I applied constant values for all of the frames so as to maintain consistence with the captured images. Once captured, the images would be saved as .jpg files, with the file names varying with each frame captured.

5.2 Future Work

While the efforts presented in this report show great potential in applying a distributed projection on an Android platform, the goal of emulating the iCVideo project is still a few stages away. This section discusses the strides needed to be taken to finalize a distributed multi-projector system using Android.

Blob Detection

The Gaussian blobs embedded around the QR codes are used to find homography across the adjacent devices. In the iCVideo project, a radically cascading method is used to register the images across the multiple projectors geometrically and the homography can achieve an edge blending across the overlaps [9]. As discussed in Section 3.2, Android applications can be built using various image processing libraries, most of which are available for developers online. OpenCV was mentioned as a possible candidate, as it comes with many equipped with a few sample applications, such as blob detector and edge detector, that may be used with calibrating system.

Synchronization

The viewing experience of a video in a system such as the one described relies heavily on the synchronization of the frames across the devices. The iCVideo project implements a synchronization method that uses visual feedback as its foundation and is independent of congestion, delay, and connection to a network. This network independence is critical to the mobile aspect of the project, as it relieves users of the variable constraints that come with a network. Furthermore, videos require 30 frames per second synchronization, a strict condition given the already congested mobile network. Therefore, it will be essential to integrate network independence when developing a synchronization method for the Android, preferably through a visual feedback from the device camera.

In a distributed synchronization scheme, each unit needs to be equipped with a camera and it independently runs the synchronization algorithm through the visual feedback from its camera. In this scheme each device does its own image capture. In this system, each device must be equipped with its own camera, identifying itself using the embedded device IDs and also identifies the device with the most lag in time (with smallest frame number) using the embedded frame numbers [9]. Using the captured frame number difference between itself and the lagging device, it stalls for the given number of frames, until the lagging device catches up.

Gesture Recognition

Once these essential features have been accomplished, a novel addition to the project would be integrating gesture recognition into the system. The basis of this technology involves designing algorithms to interpret human gestures using visual feedback from cameras. Interest in the field of gesture recognition has been significantly growing in the past few years, with studies being done in analyzing human facial expressions and hand motions through camera-recorded images, and using them as input for a given system [20]. Figure 5.6 shows gesture recognition being used to control a video

game. To incorporate this feature into our current design, a motion sensing algorithm could be instituted to identify hand gestures and let users modify the applications activity from the projection itself, rather than the device. For example, while displaying a movie onto a nearby wall , the user could pause or rewind the video by performing a given set of hand gestures on the actual projection showing on the wall. Given that the project already utilizes visual feedback in the registration and synchronization steps, an algorithm could be designed to detect human hand gestures from the camera and translate the motions into system commands. As long as these motions are being performed in the vision field of the camera, the algorithm should have no problem detecting and decoding the hand movements, opening a whole new window of possibilities for our multi-projector system.

Figure 5.6: Gesture Recognition Technology Being Used In Video Game [21]

5.3 Advantages of Tegra Based System

By implementing the proposed design, a few advantages become evident in moving the platform of the project from BeagleBoards to the Tegra 3 tablet, one of which resides in the Tegra 3 SoC. Originally codenamed the “Kal-El”, it operates as a SoC with a quad-core CPU, all cores being Cortex-A9s, and includes a fifth "companion" core. The companion core is manufactured with a special low power silicon process and is built using low-power transistors. Though it uses less power at low clock rates, this

fifth core does not scale well to high clock rates, hence limiting it to only 500 MHz. The purpose of this extra core is to allow a mobile device to power down all the normal cores and run on only the companion core [16]. This process, normally implemented during standby mode or when the CPU is otherwise underutilized, uses comparatively little power and is achieved through special logic that allows the running state to be quickly and transparently transferred between the companion and any of the other cores.

The traditional impression of a multi-core CPU has historically been that it consumes more power than a single core unit. However, the Kal-El's ability to provide variable symmetric multi-processing has made its quad core CPU more power-efficient and more able to delivery higher performance per watt than competing single and dual core processors. By distributing the system's workload across multiple cores, the each can run at lower voltages and frequencies to achieve single or multiple tasks. As a result of their low power and frequency requirements, each core operates at a lower power when compared to a single core CPU. By using digital voltage and frequency scaling (DVFS), the Kal-El strays away from the traditional single supply voltage for a region of logic, instead assigning variable supplies based on the task's performance level. While some tasks may require maximum performance from the device, others require much less performance and can be executed with less power. When lower-performance tasks are identified, reducing performance and thus the power consumption of the system, results in immediate gains in energy efficiency [17].

As the end goal of this research effort is to mobilize this multi-projector system, the convenience of the Tegra package also serves as a noteworthy advantage. Rather than relying on an external computer for executing calibration or mounting cameras to the BeagleBoards, the Tegra tablet readily provides the hardware and software on the tablet for executing nearly all of these functions without supplementary connections.

Despite these critical advantages in energy savings, the most significant gain in terms of our project is the ability to achieve real-time registration and synchronization using the Tegra GPU. Through their Tegra series, Nvidia was the first manufacturer to release a tablet with an integrated GPU. Previous projects at UCI that used BeagleBoards as their mobile device were required to capture each QR, send the information to the external PC, and remotely decode the QR code and detect the surrounding blobs. With Nvidia's GeForce GPU readily working on the Tegra tablet, both the registration and synchronization processes can be carried out in real time on the device [19].

By now it is clear that establishing registration and synchronization in a distributed system such as the one discussed involves a variety of tasks to perform concurrently. The registration step alone, for instance, requires projecting, capturing, storing, and decoding of a multitude of QR codes, all of which must be carried out simultaneously. Here, we can visibly see the benefits of Tegra's quad-core architecture over the BeagleBoard, as well as any other single core mobile system. Under heavy multi-tasking circumstances, such as the need to simultaneously run multiple applications each with its own tasks, a single-core CPU may run out of power when attempting to service the multitude of processes. Additionally, it is required to operate at peak frequency in order to accommodate the intense work-load, resulting in poor performance and excessive power consumption. On the other hand, a multi-core CPU can handle queued up requests from the abundance of tasks by servicing these requests to its various cores. If, for instance, two of the cores are occupied with handling processes with long tasks, the OS can immediately allocate time-sensitive tasks to one of the available cores, assuring faster response [17]. Figure 5.7 graphically depicts the performance speedup that the Kal-El's quad core CPU provides when running a series of demo games, and compares its performance to an equivalent CPU-based dual core platform. Here, the significant performance boost offered by the Kal-El can be readily observed, proving that the quad-core CPU can provide superior performance for graphics processes on mobile devices.

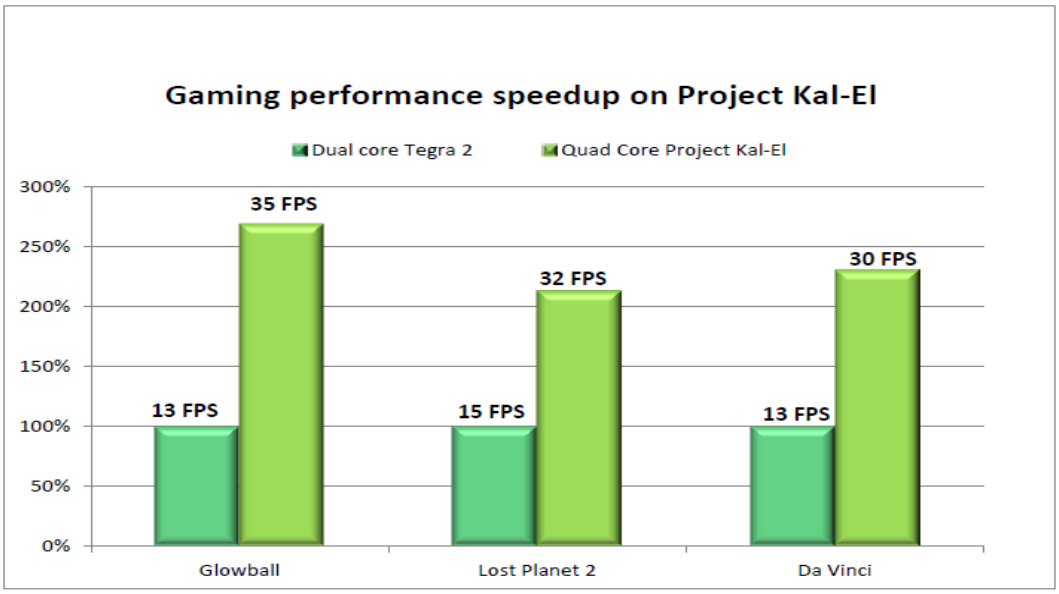

Figure 5.7: Performance Benefits From Quad-Core vs. Single-Core Processor

Chapter 6: Conclusion

As modern technology continues its paradigm shift towards miniaturization, mobile devices are rapidly taking a significant role in today's business and social environment. With studies showing that around 69% of Americans access the mobile internet on a daily basis, smart phones and tablets have quickly gone from luxury to necessity [20]. As these smart devices become increasingly commonplace, users will need a way to easily share media with each other without huddling around a screen. By integrating a display into these smart devices, users would have the opportunity to visually share information or media with one another. A perfect candidate for such a display would be a pico projector, as it is small enough to be included in the hardware of present day mobile technology. Though the brightness and resolution quality of its projections are still being improved, a collaborative system could be established where multiple devices could combine their projections either as a tiled display for a wider view, or multiple projections superimposed to render better quality.

Multiple research groups in UCI's ICS Graphics Department have experimented in establishing such a system, using BeagleBoards as the standing mobile device and pico projectors as the means of display. Dr. Majumder's students have embarked upon a series of projects, designing methods for registration, synchronization, and image formatting for these mobile devices, allowing each BeagleBoard and projector to contribute to the final image being displayed as either a tiled or superimposed collaboration. Most recently, the iCVideo project has utilized visual feedback from embedded cameras facing the projection to both register and chronologically synchronize frames [9] . Although the project achieves this distributed display, the BeagleBoard's role as the mobile device contributes to various setbacks in establishing the display, such as its inability to independently execute calibration steps and a processing unit that struggles to perform multiple tasks simultaneously. In this paper, an alternative

design was suggested that replaces the BeagleBoard platform with an Nvidia Tegra tablet, allowing a more independent, responsive, energy friendly means for establishing a mobile distributed system.

Excluding the pico projectors, the proposed design eliminates the need for most of the hardware that went into the iCVideo display, such as the BeagleBoard, camera, and external computer, scaling down to a simple Tegra tablet. The Tegra has three different cameras available on the device, operating on a quad-core CPU that utilizes a fifth "companion core" for power efficiency. In terms of software, the Android SDK allows apps to be developed and run on the device that perform similar tasks for registration and synchronization as the BeagleBoard, without the need for any external operating system. Valuable libraries, such as OpenCV and FCam, can be used for developing applications geared towards processing images and modifying a camera's capturing parameters, respectively. There also exists a plethora of open source applications that can be downloaded and modified, allowing tasks such as QR reading to be easily observed and incorporated within any application.

With the Tegra serving as the central mobile device in a multi-projector distributed system, the process of registering the frames and synchronization can be achieved in real-time. While the BeagleBoard is incapable of independently executing these steps, the powerful ULP GeForce GPU allows the Tegra board to perform image processing tasks, such as decoding QR codes or detecting blobs, at very high speeds with instantaneous results. The GPU incorporates 12 cores, allowing it to operate at ultra-low power (ULP) and produce efficient results by resourcefully allocating tasks among the available cores. Meanwhile, the CPU disperses its work in a similar way among its 4 cores, with its fifth core acting as a low-power substitute that allows the rest of the system to save energy. The idea is to have the companion core be the only core running during modes that require little or no CPU utilization, allowing the other 4 cores to power down and save energy. The CPU also utilizes DVFS to ensure efficient voltage and frequency scaling when executing low-performance tasks.

With its supreme ability to process graphics in real-time, coupled with its endless configurations for energy conservation, the Tegra tablet emerges as the perfect candidate for a multi-projector distributed system. Aside from the necessary HDMI connection to the pico projectors, the board holds all the necessary hardware in a compact package. Ideally, there will come a time when a group of mobile-device-owners can unite and create a collaborative presentation or assemble a projection of a game, simply by running the calibration programs on their respective devices. Judging by the exponentially increasing number of mobile devices being purchased across the world, it's beginning to seem as if everywhere we look, somebody is on their smart phone or tablet. As this market continues to grow, it will on be a matter of time until the concept of collaborating from these devices becomes a reality.

References

[1] Livolsi, Bill, “Pico Projectors for Entertainment,”*http://www.projectorcentral.com/favi_e3_pico_projector_review.htm?page=Limitations-and-Conclusion* (ProjectorCentral.com, 2011).

[2] P. Roman, M. Lazarov, and A. Majumder. A saclable distributed paradigm for multi-user interaction with tiled rear projection display walls. *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[3] Pico-Projector Info, “Pico Projector: Introduction,” *http://www.picoprojector-info.com/introduction* (Pico-Projector Info, 2009).

[4] Microvision, “How it Works,” *http://www.microvision.com/technology/picop.html* (MicroVision, Redmond, WA).

[5] NextGenLog.Blogspot, "MEMS: Pico projector to obsolete handheld's LCD?," *http://nextgenlog.blogspot.com/2009/02/mems-pico-projectors-to-obsolete.html* (NextGenLog, 2009).

[6] DigiKey, “Pico Projector 2.0,” *http://www.digikey.com/product-highlights/us/en/texas-instruments-dlp-technologies/688* (DigiKey, Thief River Falls, MN).

[7] Tec-It, “2D Barcode: QR-Code,” *http://www.tec-it.com/en/support/knowbase/symbologies/qrcode/Default.aspx* (TEC-IT Datenverarbeitung GmbH, Austria).

[8] E. Bhasker, P. Sinha, and A. Majumder. Asynchronous Distributed Calibration for Scalable Reconfigurable Multi-Projector Displays. *IEEE Transactions on Visualization and Computer Graphics, 2006.*

[9] Kiarash Amiri, Shih-Hsien Yang, Fadi Kurdahi, Magda El Zarki and Aditi Majumder “Collaborative Video Playback on a Federation of Tiled Mobile Projectors enabled by Visual Feedback”, *ACM Multimedia Systems* 2012.

[10] Coley, Gerald, “BeagleBone Rev A5 System Reference Manual,” *http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf* (BeagleBoard).

[11] Nvidia, “Tegra Development Kit Android Setup Experience,” *http://developer.download.nvidia.com/tegra/docs/android_setup_experience_public_20111122.pdf* (Nvidia, Santa Clara, CA, 2010).

[12] Android, “What is NDK,” *http://developer.android.com/tools/sdk/ndk/overview.html* (Creative Commons Attribution 2.5, 2012).

[13] Sourceforge, “Open Source Computer Vision Library,” *http://sourceforge.net/projects/opencvlibrary/* (Sourceforge, 2012).

[14] FCamera, “FCam: What is it?,” *http://fcam.garage.maemo.org/* (Maemo).

[15] ZXing, “ZXing (Zebra Crossing),” *http://code.google.com/p/zxing/* (Google Project Hosting, 2010).

[16] Hildenbrand, Jerry, “Nvidia Announces Tegra 3,” *http://www.androidcentral.com%2Fnvidia-announces-tegra-3-pc-class-performance-comes-android-tablets/* (Android Central, 2011).

[17] Nvidia, “The Benefits of Quad Core CPUS in Mobile Devices,” *http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911a.pdf* (Whitepaper, 2011).

[18] Nvidia, “Nvidia TEGRA MOBILE PROCESSORS,” *http://www.nvidia.com/object/tegra-3-processor.html* (GLBenchmark 2.0 Egypt, 2012).

[19] Sterling, Greg “Study: 69 Percent Access The Mobile Internet Daily,” *http://marketingland.com/study-69-percent-access-the-mobile-internet-daily-4371* (Search Engine Land, 2012).

[20] Omron Global “OMRON Develops Hand Gesture Recognition Technology,” *http://www.omron.com/media/press/2012/05/e0528.html* (Omron Corporation, 2012)