Project Kidprint

Albert Tang, Eric Chang

Spring 2016

1 Abstract

One of the challenges that doctors in developing countries face is identification - namely, how to keep track of patients that they have treated and how to store their information. This becomes even more of a problem when the patient does not understand or remember their medical information - when they are infants, for example. This makes even the most basic treatments, such as vaccinations, very difficult - and a mistake is potentially fatal.

Project Kidprint aims to solve that problem by going back to one of the earliest, but most reliable forms of identification - fingerprints. Our goal, in conjunction with researchers and graduate students at the Qualcomm Institute at U.C. San Diego, is to create a modular, intuitive, and cheap way of reliably reading the fingerprints of infants. While other attempts have been made, the results have either been unsuccessful, or far too expensive and inaccessible. We hope to succeed where others have failed by taking a different approach to solving this problem - rather than take a specific technology and mold it to our needs, we are choosing to experiment with multiple technologies and picking the ones that are best suited to the task.

2 Introduction

In developed countries such as the United States, identification is something that is easy and oftentimes taken for granted. Any citizen on the street usually has multiple ways of proving their identity, whether it is a drivers' license or passport or birth certificate, etc. Those who live in impoverished or developing countries oftentimes to not have that luxury. Those who are poor may not be able to afford getting an official means of identification, and with poor bureaucracy and inconsistent standards, the types of identification they do have may not be sufficient. Additionally, many people have no means for identification because it does not affect their day to day life - they have no incentive.

This presents many issues when it comes to determining who is who, and keeping track of people and their information. One of the groups of people it affects is doctors who are working in third world countries. While this is not necessarily a problem for local doctors, who know their patients personally, for doctors that travel a lot or have a lot of patients, it is hard for them to remember them all.

One of the major problems of insufficient identification is being able to keep track of vaccinations. Patients often do not remember their vaccination, and doctors have a hard time keeping track of medical records without a reliable form of identification. The low rates of vaccination present serious problems in third world countries with 1.5 million people still dying by vaccine preventable diseases each year. [1].

The purpose of Project Kidprint is to provide a solution to that problem by making it easy for doctors to keep track of their patients and what vaccines they have already taken. The idea is to develop a device that can identify an infant based on their fingerprints. This device will be easy to use as well as modular and affordable. We hope for it to become the widely accepted standard.

Identifying infants - and people in general - by their fingerprints provides a variety of advantages. Firstly, and most importantly, everyone has fingerprints. Unless the individual has suffered some form of injury, everyone has a unique set of fingerprints that are distinct and unchanging, providing a reliable and consistent method of identifying them. People cannot lose them, and for the most part, they cannot be destroyed. Other people cannot steal fingerprints or fake identification.

You may ask, why has this not been done already? And indeed it has been tried, but results have been unsuccessful. There are many complications that come with fingerprinting infants that are not present when fingerprinting adults. The most obvious one is size. Infants have smaller fingers, making it more difficult to extract the details and differences between them. Another problem is that a infant's skin is not quite as developed. Unlike the skin of someone who has lived a few years, the fingers of infants are oftentimes still very soft and as a result the details are not quite as An often overlooked problem is whether or not the technology is trusted. Fingerprint machines are often intimidating and as a result, not only do the infants refuse to get close to them sometimes, even the adults may fear their children getting hurt by the machine or using an improperly sterilized scanner.

Project Kidprint is different in that it approaches the problem differently. Rather than take a specific technology and attempt to mold it to do what we want it to, we look at the problem first and choose the right technologies for the job. By being flexible with what we use, we oftentimes consider options that others may not have.

Much progress has already been made, and results so far are very successful. The following is the data that we have already gathered. We are currently looking to expand our research and move forward in developing a prototype.

Images Captured	170
Quality Images	137
Matched Images	136

3 Technical Material

3.1 Autofocus

Implementing the autofocus turned out slightly more involved than expected. However, we divided the autofocus into smaller subproblems, and the task became much more managable. The individual modules did not take too long to implement and were easy to test and debug using Python. We implemented two main modules based on OpenCV's C++ implementation of autofocus. One module takes an image as a parameter and rates the image based on how sharp the edges are. This assumes that the sharper the edges, the more focused the image is. This image gets passed through a Gaussian blur to reduce noise followed by a Canny Edge Detection to identify and rate the sharpness of the edges. Another module takes in the rating of the previous frame as a parameter, and outputs the next focus step to take to find the peak focus. To simplify this module, we imagined this as a finding the local maxima given a set of points (focus level vs frame rating). Our initial unit tests involved half a period of a sine wave to simulate a function with a peak. After this was working, we tested it with a noisy gaussian function (bell curve), and debugged it further until it worked in several cases. Other modules involve interfacing with the device and adjusting the focus level in real time, such as sending controls to the device via Serial to change focus level or handling user input.

The main issues we encountered happened when testing autofocus on the actual device. The Video Capture feedback from the device did not match the quality that was expected with the given focus level. When manually sending the optimal focus level to the device, it would not update immediately. For example, if the focus level happened to be at F=210, we would tell the device to go to 210, and then capture an image of the next frame. The image captured did not turn out to be the quality that we expected from this focus level. We first attempted to fix this by adding some delay before receiving the feedback from the video, but this did not change the quality at all. After hours of debugging, the problem turned out to be that the device was still streaming video frames while the focus level was shifting. These frames turned out to be buffered, which explains why adding a delay did not change the quality. Instead of adding a delay, we simply threw out the frames while the camera was adjusting focus, so that the intermediate, inaccurate focus levels would not be captured.

In the future, some improvements we have in mind for autofocus are optimizing the algorithm to detect the peak faster, and also improving the autofocus when the finger is pressed against the glass. Currently, the autofocus works well when the finger does not touch the glass at all. However, when the finger touches the glass, the edges get blurred out due to the pressure against the flat surface. This results in very random ratings that significantly affect the reliability. To fix this issue, we are thinking about having a selectable area to focus on, rather than trying to sharpen the edges over the whole image. Since the resolution control turned out successful, we can also try implementing the autofocus directly onto a microcontroller. This may actually work in real time since we can further reduce the resolution to where the microcontroller can rate frames fast enough.

3.2 Bluetooth Communication

Bluetooth communication was set up utilizing two Arduino chips and the Arduino IDE as well as Python paired with pyserial. The two Arduino chips were the SparkFun Pro Micro, which is the actual Arduino controller, as well as the SparkFun BlueSMiRF Silver, which is the Bluetooth module that was connected to the Pro Micro. The Pro Micro was connected to the Silver via the RX and TX serial communication ports, which were soldered onto the TX and RX communication ports of the Silver respectively. The Pro Micro was powered through its Micro USB communication port, which then provided power to the BlueSMiRF via a soldered connection. Communication on the computer side was handled first with the Arduino IDE and then with a simple Python script that sent data via pyserial. These were programmed and run off a Windows laptop.

An interesting problem that we ran into was for reasons unknown, the baud rate - or data transfer rate - of the Bluetooth chip and the serial communication had to be initialized to different values for data transfer to work. Though the Bluetooth chip was initialized to a standard baud rate of 115K, the only baud rate that worked when initializing serial communication was 9.6K. Other baud rates that were close to 9.6K resulted in garbage values being sent and received by the Bluetooth module. Furthermore, sometimes characters would be dropped and not read. This was tested by making the Pro Micro blink a certain amount of times if a specific character was received. Initializing the serial connection to something even further from 9.6K resulted in nothing being received at all. We changed the baud rate on the BlueSMiRF itself to match the baud rate we initialized our serial connection to, but that simply resulted in all communications failing with no data being sent.

Gathering timing data thus took place at a baud rate of 9.6K, with the BlueSMiRF running at 115K. For the Python part, this was simple because you could just access the global timestamp. This became a little bit more complicated for the Pro Micro controller, because it has no global timestamp and can only begins counting when it turns on, and has no way to print it. To fix this problem, we tacked on the elapsed time to the end of the input that we returned, as well as visually judged when the LEDs on the Pro Micro began blinking.

The approximate time it took for each step of communication is as follows. Sending the data and receiving it from the Windows device took approximately 40ms. Reading all the data and sending it back took approximately 1100ms. Receiving the returned data on the Windows device took approximately 500ms. This meant that two-way communication turned out to be too slow to use effectively. At its fastest, it took approximately 1.6 seconds for a message to be sent to the Pro Micro, read, and returned. This is unreasonable for any means of real time communication, and made it impossible to implement the autofocus algorithm on it. The autofocus algorithm would have no way of knowing whether or not the lens value had been adjusted by the Pro Micro, because it was so slow, and would likely send data multiple times for the same focus level, throwing off the algorithm completely. This issue was already run into when implementing autofocus, over a direct wired connection. Having it run wirelessly with a delay of 1.6 seconds would cause the autofocus to not work at all. One-way communication, however, seems to be viable, as the data sent over Bluetooth was received almost instantly. This means that for tasks that do not require feedback from the Bluetooth controller, Bluetooth control could be implemented.

This could be for a variety of reasons. Firstly, it's possible that the Bluetooth chip itself was the problem. Without another Bluetooth module to test against, we simply could have received a defective one that happened to be very slow for whatever reason. Secondly, it's possible that the issue was with the Windows computer that we used to communicate with the Pro Micro. We did not have a Linux system available to test Bluetooth communication on, and the Macintosh system we had could not establish a connection with the Bluetooth chip for unknown reasons -

suggesting that perhaps the BlueSMiRF itself was the problem. Thirdly, it is possible that due to issues with the baud rate the data is being transmitted far too slowly. Given that the baud rate must be initialized to something different than what the BlueSMiRF is running, perhaps that is causing issues. It is unlikely that the problem lies within pyserial or the Arduino IDE. Timing data suggests that pyserial and the IDE both receive and send messages at the same time, so the problem likely lies within the devices themselves.

Communication was established successfully in that the Windows device and Pro Micro could send data back and forth. However, the timing of the devices means that many applications using Bluetooth are not viable. Until the issues with Bluetooth are worked out, using the SparkFun chips should be avoided. More time as well as alternate devices are needed to fully understand the problem.

3.3 User Interface

There are two features of the User Interface that we implemented - Registration and Identification. This interface is intended to be used by the doctor with the fingerprint device. We decided to implement the User Interface in Python for consistency and simplicity. As of right now, the interface does not require a nice GUI, but does require the basic functionality to work.

- **Registration**
  The first time an infant uses the device, he or she needs to be registered into the database. The parents will need to fill out some basic information, such as name, date of birth, city. The user would then need to take several images of each finger in order to create a sample set to match the fingerprints. Our work in this feature involves creating a brief questionnaire to request all the necessary information to register a new infant and adding the new record to a database.

- **Identification**
  These are the steps for the identification process:

  1. Use a camera to take an image of the fingerprint
  2. If the image is clear, process the image using the software pipeline
  3. Identify the matching child and pull up medical record information

  Our work in this process involves being able to look up the record of an infant. We did not work with the image processing pipeline, so our temporary identification process looks up an infant by name.

3.4 Development Environments

- **Python**
  We used Python for the majority of development in this project. The Image Capturing interface, Autofocus, and the User Interface were all implemented in Python.

- **Arduino**
  We used Arduino for development on the micro controllers. The Arduino controlling the Varioptic camera accepts keyboard input from Python to perform various actions over serial. The bluetooth communication through the Pro Micro controller also compiles using Arduino.

4 Milestones

4.1 Completed

- **Autofocus Implementation**
  The purpose of this milestone is to significantly reduce the time required to capture fingerprints from infants. The current prototype model uses a varioptic without autofocus, so there are two potentiometers serving as tuning knobs for adjusting focus. Before capturing each image, the team must determine whether the image is in focus, which takes a significant amount of time. With the autofocus implementation, the team would be able to hit a button, which will tell the device to find the optimal focus level in under one second.

- **SparkFun Pro Micro Interface**
  The purpose of this milestone was to implement Bluetooth communication with a microcontroller. As part of the prototyping phase, it was not necessarily intended to be part of the final device, but merely experimental. The current prototype connects to the laptop, and we wanted to test different ways of processing the fingerprints. The Bluetooth device would allow the device to capture fingerprints and immediately send them over to the computer for processing.

  We accomplished this milestone by setting up two way communication with the Bluetooth. One issue we ran into was that the two way communication was too slow to be reliable. We did not end up implementing the controller onto the chip, as it was not a priority and not worth the effort given the long communication delays. Though we did not implement the camera controller onto the SparkFun chip, we managed to successfully setup the communication and development environment that we needed in order to do so, which was the goal of this milestone.

- **Timing Information**
  The purpose of this milestone was to obtain the timing delay for different stages of the image capturing, namely for the autofocus algorithm and the Bluetooth communication. We analyzed the timing delay of rating frames at various resolutions and this information helped us determine that reducing resolution down to 1280x720 is optimal during autofocus.

  We accomplished this milestone by recording delay times between streaming and frame rating. After measuring the time it takes for each frame, we decided that in order to autofocus in real time, we had to reduce the resolution.

  Timing data was also successfully obtained for the Bluetooth via the means explained in the Technical section of this writeup.

- **User Interface(Python)**
  The purpose of this milestone was to create a very basic User Interface for registering and identifying infants. This is to make a holistic interface rather than just one to capture images or process images, etc. Also, the UI is supposed to be more user friendly than the ones that we currently have.

  We accomplished this milestone by creating a simple, modular user interface that has the basic functionality and will be easy to add additional features. Though it currently does not interface with the other code because their implementations are constantly changing, it is prepared to be able to interface with image capture, fingerprint processing, and database interaction once those have been set up.

- **Resolution Control**
  The purpose of this milestone was to implement dynamic resolution control for the camera. This milestone assists the team by allowing them to easily capture images at different resolutions, so they can find the lowest resolution required to still take high quality images for matching. We noticed that the resolution does not matter for determining the optimal focus point. Completing this milestone helped out with autofocus because it allowed us to reduce the resolution down while the camera focuses, and change it back once the focal point has been found. This allows the system to rate frames and change focus without any lag.

  We accomplished this milestone by parameterizing the resolution in the Image Capture interface, and allowing the user to make adjustments to the resolution during the capture.

- **Naming System**
  The purpose of this milestone was to make the file naming convention more robust. We coordinated with other team members to identify which parts are necessary in the filename and how they should be formatted. This also helps with sending the images through the pipeline for matching and adds flexibility to the filename, if it needs to be modified in the future.

  We accomplished this milestone by discussing with our team to work out a robust naming scheme. This allowed us to use regular expressions to easily check the working directory to calculate the filename of the next image to capture.

4.2 Unfinished

- **User Interface (C++)**
  The purpose of this milestone was to implement a basic user interface coded in C++. This UI would essentially copy the functionality of the existing Python user interface. During the time the milestone was created, there were considerations being made to rewrite the software in C++, as we wanted it to be platform independent and C++ is well supported across multiple devices. One of the possibilities was having an FPGA or other custom-programmed embedded controller on the final device, and having the code in C++ would result in easiry compilation to Verilog if necessary. Additionally, because some elements of the code were running rather slowly, porting them over to C++ could possibly result in performance gains that would allow them to be run in real time.

  This milestone was not completed because the team decided not to port the code to C++. There were several reasons for this. One of the primary reasons is that the image processing pipeline was being programmed in Python, and it would be simpler if all the code were in Python and there was not multi-language communication happening. The second reason is that we decided, tentatively, that the final product would be run on a Linux system, either on a PC like a laptop or perhaps on a mobile smartphone. Because Python can run on Linux just fine, we decided to stay with Python. The third reason is that we determined since we were still in the prototype phase, it was a better idea to conduct research and experiment with what we knew worked and what was fastest to implement. This was the Python code that we had been working with already. This does not necessarily mean that the platform will not be converted later, but for now Python is the language of choice.

- **Y16 Monochrome Display (C++)**
  The purpose of this milestone was to implement interfacing with the See3Cam Y16 Monochrome Camera that we had, including displaying a live stream of what the camera was capturing as

well as implementing frame saving capabilities. The idea was that having the monochrome pixels might speed up the rest of the pipeline, as both the algorithm that determined the focus as well as the image processing algorithm would not have to manually convert the standard RGB pixel format into greyscale for determining focus and emphasizing detail.

This milestone was not completed for several reasons. Firstly, as explained in the previous milestone, the decision to move to C++ was rescinded so it was unnecessary to display the monochrome camera in C++. Secondly, though we implemented some algorithms that we found online, we could not get any of them to work properly.

- **Y16 Monochrome Display (Python)**
  Like the previous milestone, the purpose of this milestone was to implement interfacing with the See3Cam Y16 Monochrome Camera, only this time with a Python interface rather than one in C++.

  This milestone was not completed because quite frankly we could not get it to work. Much like the previous milestone, the closest we could get is for it to display a staggered image - that is multiple instances of each frame overlapped with itself. Despite implementing different algorithms that we found online to convert the pixel format, none of them worked - a few only displayed noise or just wouldn't even compile. We even tried recompiling OpenCV and that still did not fix the problem.

  If we were to change what we had done, we would not spend so much time on this milestone. We wasted days trying to get the camera to display properly, when that time could have been spent implementing the milestones that were never completed, or improving upon other elements of the project.

- **Image Burst**
  The purpose of this milestone was to implement burst capture on the camera. Essentially what it would do is upon a keystroke, it would grab multiple images sequentially, possibly each with a different focus level. This would be portable to changing light levels too, if that were necessary and if Arduino controls for the on-device LEDs were implemented. The idea was that because taking the current frame at the current focus level might result in a poor fingerprint image, leading to a poor reading, taking multiple photos very quickly would result in a higher chance of one of those being of high quality.

  This milestone was abandoned for several reasons. Firstly, it was not too high of a priority to begin with, and other milestones were proving a bit tricky and required more attention than was initially expected. Secondly, because the autofocus implementation was proving promising, this milestone was deemed unnecessary. Having a working autofocus algorithm would result in the camera automatically detecting the best focus level - or close to it - and adjusting the lens to that point. This meant that taking multiple images across multiple focus levels would be unnecessary, as the autofocus algorithm would effectively find the best image anyways.

- **Introduction Text**
  The purpose of this milestone was to come up with introduction text that would be displayed to the user when the program started. This was an attempt to make the UI more user friendly by printing a usage statement that would show the user which commands they could use. Up to this point, one of the graduate students had been handling obtaining the image from the camera, and as a result had written most of the code himself. As a result, he was the only one who knew how to use the interface he had programmed.

This milestone was abandoned for several reasons. Firstly, it was not a priority to begin with, and the autofocus algorithms and monochrome camera were proving a bit tricky and required more attention than was initially expected. Secondly, general improvements that were made to the UI made it more user-friendly, and printing out the available commands to the terminal proved sufficient for other team members to be able to use the image capturing program. Lastly, the completed Python UI was self-explanatory enough to not require a usage statement, and though it may not be used, a usage statement for that specific UI would not be portable to a new one.

5 Conclusion

The main contributions we made to this project were in autofocus and Bluetooth, along with general user interface improvements.

The first task in autofocus was to design the system for finding the best focus and determine all the separate modules needed to do so. Implementing autofocus was a good learning experience, as it gave us the opportunity to break a problem down into more manageable subproblems. It later required debugging of individual modules, as well as integration with the system.

Implementing the system to transmit and receive Bluetooth messages was problematic, and probably representative of the kind of issues that wireless serial communication has. Merely using the Arduino IDE caused issues, as using the latest build resulted in the code being unable to compile, for unknown reasons. The baud rate between the SparkFun BlueSMiRF chip and the Windows device had to be initialized to a different baud rate than the one that the . Even when wireless communications were successfully set up, the delay for two-way communication was far too great to be of any reasonable use, though one-way communication remains viable.

Given the unreliable nature of wireless communication, it seems best to avoid it altogether. At least with Bluetooth, it is impossible for the device to be fully wireless anyways, as the streaming rate of Bluetooth is far too low to transmit real-time video. Though other wireless options that are fast enough to stream video are available, namely WiFi, the other issues with wireless communication persist. Furthermore, the extra battery drain on the device is yet another cost that is unjustified. It seems that wired serial communication is the best bet for the final device, especially if the product needs reliability to be of primary importance.

Overall, Project Kidprint is moving along nicely. With the recent work that has been done implementing autofocus and Bluetooth capabilities, we feel that we have made successful contributions to the ongoing team effort and are looking to continue working on this project during Fall quarter.

References

[1] http://www.who.int/gho/immunization/en/