A PARALLELIZED ALGORITHM FOR HYPERSPECTRAL BIOMETRICS

By

Christopher Baker

Timothée Bouhour

Akshay Malik

Final Report for ECE 445, Senior Design, Fall 2012

TA: Lydia Majure

10 December 2012

Project No. 1

Abstract

The parallelized algorithm for hyperspectral biometrics uses the processing power of a GPU (Graphical Processing Unit) to compare hyperspectral images of people's faces. The feature extraction algorithm first retrieves uniquely identifiable features from raw hyperspectral data from 64 bands and creates both a database and individual target files. Using these files, the comparison algorithm written in CUDA C compares a given target against the database and returns the top five matches, their calculated distance from the target, and their security clearance level. A wireless door locking mechanism can be engaged to simulate unlocking and re-locking any of four doors based on the given security rating. The feature extraction algorithm is accurate to within 2% of actual location and the comparison algorithm returns the target in the top 5 matches 65% of the time. The wireless door locking assembly works as expected although it occasionally has packet corruption errors in its communication. Improvements can also be made in the range of data that the feature extraction algorithm accepts and in the accuracy and speed of the comparison algorithm.

Contents

1. Introduction

In the modern world unique person identification has become an increasing challenge, central to strategies in combating terrorism and crime to provide global security. Recent research has shown that hyperspectral imaging provides new and improved biometric data, which can be leveraged to meet this challenge by examining features in different spectral bands.

Hyperspectral imaging provides light intensity information on a spectral band of different wavelengths, as opposed to a regular image that only typically provides R, G and B values for each pixel. For this project, the database of hyperspectral data includes information from 65 bands spaced evenly (every 10 nm) in a spectrum of wavelengths from 450 nm to 1100 nm. It has been obtained from Carnegie Mellon University [1]. This visual data shows much promise because near infrared bands have been proven to gather data past 1cm into subcutaneous (below the skin) tissue [2]. A method that can effectively use this type of biometric data would be much harder to counteract than a standard facial recognition system.

Despite its promise, this method of identification still has some challenges, which must be addressed before it can be applied in the real world. One of those challenges is dealing with the massive amount of data that a hyperspectral sensor generates. Another is implementing a reliable algorithm to successfully run comparisons and putting together a useful package able to interpret hyperspectral sensor data.

The system described in this document successfully implements a feature extraction algorithm that is 2% accurate over our testing database, a reliable algorithm for feature extraction that returns the target in the top 5 matches 65% of the time, and a wireless door locking assembly that successfully simulates the locking and unlocking of four doors wirelessly from a web browser. The door locking assembly does suffer from packet corruption and could be made more robust. Further improvement could be made in the range of inputs the feature extraction algorithm can take as well as in the precision of the comparison algorithm.

Though some implementation decisions have changed, these three main blocks and their functionalities have remained the same throughout the development process.

1.1 Feature Extraction Algorithm

The system described in this document addresses these engineering issues by combining three major blocks shown in Figure 1. The first block is a feature extraction algorithm, which selects the most useful bands and points of a hyperspectral image. This algorithm selects an optimal group of nine by nine pixels in the regions of the forehead, lips, right cheek, left cheek, and hairline. For pixel in a batch, the intensities for the 30 bands found to be the most useful are stored. The entire collection of data thus created is then either stored in an individual file to be used as a target for comparison, or added into a large database.

Figure 1 - Design Block Diagram

The performance requirements for the feature extraction algorithm are to use hyperspectral data, identify the position of features in an image within 10% accuracy, and output a database of at least 20 real subjects and 180 generated normalized subjects with intensities in the same range as the real subject intensities for each band.

1.2 Parallel Comparison Algorithm

The second block is a fully parallelized algorithm written in CUDA C (a parallel programming language). This algorithm leverages the Kepler architecture of CUDA GPUs (Graphical Processing Unit) to optimize the comparison of all intensities and achieve higher performance than a functionally identical sequential algorithm running on a CPU (Central Processing Unit). The algorithm compares all the features included in a target subject file against the features of the subjects in the database. It outputs an ordered list of the top five matches in the database, along with the calculated distance and their security level. At this point, the security level is randomly generated for each subject and stored in a separate database.

The performance requirements for the algorithm are to be able to run compare any target with every entry in the database, output the top 5 matches in order, and have the target within these top 5 matches 50% of the time.

1.3 Wireless Door Locking Assembly

The third block is a door locking mechanism composed of a wireless router, microcontroller, and a PCB hooked up to four motors. The motors are used to simulate the operation of door locks disengaging and re-engaging. The purpose of this block is to provide a real-life example application of the algorithm. The microcontroller hosts an http web server that a user can access through any browser connected to the same wireless network. In the browser, the user can select to simulate the unlocking of any four doors by clicking links. The microcontroller will then use digital outputs to rotate one of the four motors one way for three seconds, and the other way for three seconds after a one second pause.

The performance requirements for the door locking assembly are to receive and act on signals from the computer, correctly engage the desired motor when requested, and properly rotate this motor for three seconds in one direction followed by three seconds in the other with a one second pause in between.

2 Design

2.1 Feature Extraction Algorithm

The purpose of this module is to extract the spatial locations of features from a subject's face and obtain the relevant information for each feature for every subject. This algorithm is used to build the database and also obtain the radiance information for a target subject that is compared against the database. By storing only specific information for each subject, there is minimization of the amount of memory required to store data for a large set of people. Figure 2 shows the flowchart design for this algorithm. The data obtained from a research project at Carnegie Mellon University [1] contains 46 subjects. Out of these, 10 had one session only, 10 were used for training purposes, and 26 were used for testing the final algorithm. Each session information contained image files built from radiance information for each of 65 bands of wavelengths ranging from 450 nm to 1100 nm. These were converted into a single MATLAB file for each session for every subject in a format that contained 65 radiance numbers for each pixel in the 640*480 pixel image.

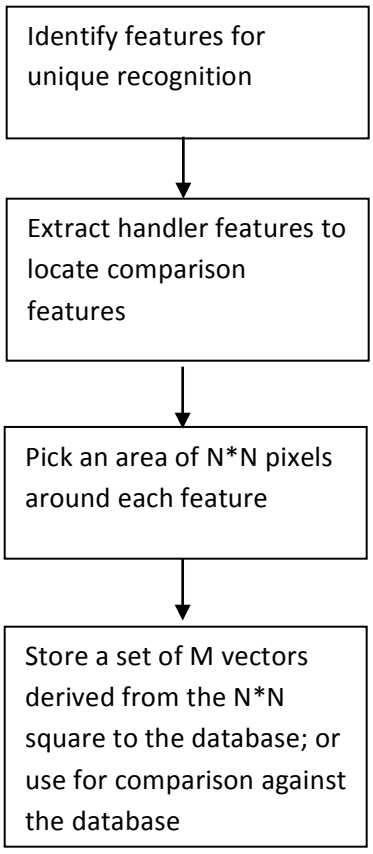


Figure 2 - Feature Extraction Algorithm Flowchart

2.1.1 Identification of features

The first step was to identify the features that would be easy to extract and a combination of which would be sufficient to uniquely identify a person. After extensive research and literature review [1][3], we decided to use the hyperspectral information from the cheek tissues, lips, forehead and hair for each subject to store and compare. It should be noted that some features may have been left unexplored, and as a design alternative different features could have been picked.

2.1.2 Feature Extraction

The next step was to extract the positions of these comparison features. To achieve this, we realized that it was more appropriate to extract the positions of the eyes, lips and hair, because of the gradient between the radiance information for these attributes and the skin tissue, as demonstrated in Figure 3. These 'handle features' are then used to geographically map out the face of the subject and extract the positions of the comparison features (cheeks, lips, forehead and hair). The highlighted band in Figure 3 was used to perform the extraction of these features.

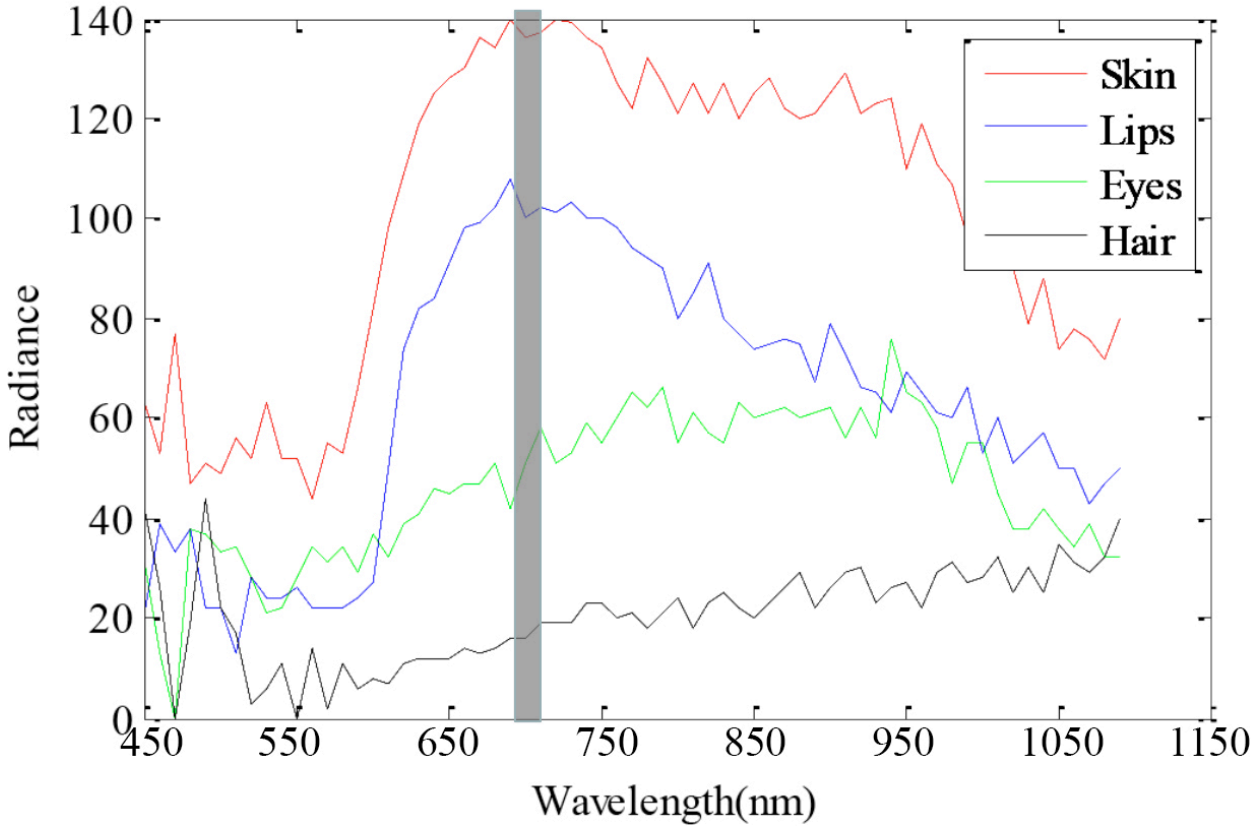


Figure 3 - Chart of the Radiance versus Wavelength for the handler features and skin tissue of subject 2

The algorithm uses the gradient between the radiance from the background and the hair to find coordinates for the left, right and upper bounds of the face. These coordinates are used to find the center of the face, which is located somewhere in the nose area. Since the orientation is fixed for all faces in the data set, the left eye is always in the upper left region (Quadrant I), the right eye is in the upper right region (Quadrant II) and the lips lie somewhere below the center of the face. To extract the pixel locations relevant to the left eye, the algorithm performs a modified version of breadth-first search in Quadrant I, considering each pixel on the upper and left edges of squares growing outwards in Quadrant I as shown in Figure 4. The information for this pixel is compared to the pixel at the center of the face and pixels sufficiently above and below (approximately 1/10th the total length of the face). If the current pixel intensity is sufficiently different (60-70% less) as compared to these three pixels, a test is performed in a small square centered at the current one to re-affirm that the current pixel is actually a part of the left eye rather than an inconsistent data point. This test includes checking all the pixels in the small square and ensuring that at least 60% of these have radiance values within 20% of the pixel under consideration. The dimensions of the face determined the size of this square (1/30th the width of the face).

The algorithm continues a search in the rest of the first quadrant and picks other pixels that satisfy the criteria. Pixels which are further than 1/10*width of the face from the first pixel picked are discarded to ensure that none of the pixels in the hair or eyebrows are picked. Since the left eye is the first feature with sufficient gradient encountered from the center of the face which performing such a search, all pixels that are considered are part of the left eye. The center coordinates of the left eye are calculated by averaging over the coordinates of all the pixels that were picked. The same algorithm is applied in Quadrant II to calculate the coordinates of the center of the right eye. For the lips, the area below the center of the face lying between the centers of the eyes is considered, and a similar search is performed.

Once these three handler feature positions are extracted the x-coordinates of the cheeks are determined as the x-coordinates of the eyes and the y-coordinates of the cheeks are found by averaging the y-coordinates of the center of the eyes and lips. The forehead x-coordinate is the x-coordinate of the lips and its y-coordinate is determined to be at a distance equal to half the x-distance between the eyes above the center of the eyes. The hair position is considered as the location for the upper bound of the face.

Thus, this algorithm extracts the locations of the features needed to uniquely identify a person from an input image of a person's face.

Our algorithm works well for data where the entire face is captured from a front view, as was the case with the data from CMU. Alternatively, more advanced image-processing techniques such as eigenfaces [4] or linear discriminant analysis [5] could be used for real world data and further improvements could be made to increase the robustness of the current algorithm.

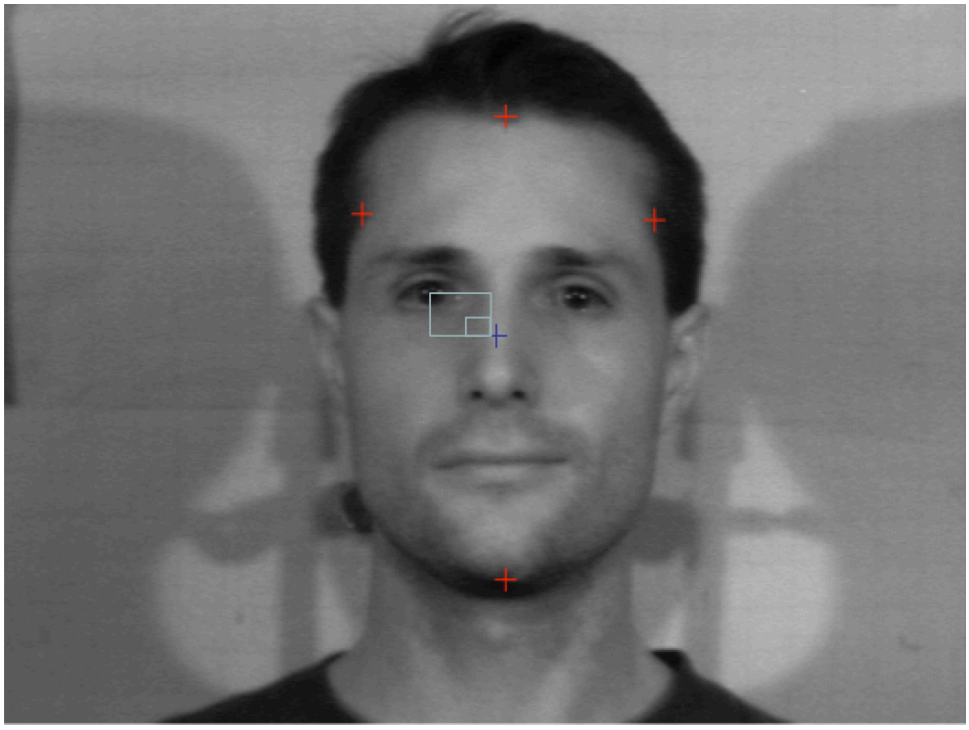


Figure 4 - demonstration of feature extraction on subject 2

2.1.3 Building the Database and Target files

The next step is to build a binary file for each subject. This is done by considering squares of size 9*9 pixels centered at the extracted coordinates for the hair, lip, cheeks and forehead, normalizing radiance across all 65 wavelength bands for each pixel in those squares and inserting this in a specific format into a binary file. Since the near IR region is the one most relevant for facial recognition, only the information for the 30 bands from 700 nm to 1000 nm was stored in the binary files for every subject. For building the database, binary files built from session 2 of the data set we had obtained from Carnegie Mellon University were used for each subject and combined into a single database file. Session 1 binary files were used as targets for comparison in the identification algorithm. The demo database had entries for 26 real subjects from the Carnegie Mellon University data and a variable number of randomly generated subjects for timing measurements for the parallel algorithm.

A more extensive analysis might reveal the dominance of other bands and features, which can be used in the database. There could also be different formats for developing the database that might lead to further compression of information.

2.2 Parallel Comparison Algorithm

The Parallel Comparison Algorithm, as described previously in section 1.2, handles the comparison of target data to a database in order to identify the target.

This section will first discuss the chosen method of comparison (2.2.1) and the serial algorithm developed for benchmarking (2.2.2). It will then move on to discuss the development of the parallel algorithm from the serial algorithm (2.2.3 – 2.2.5) and finally conclude with an overview of the final implementation (2.2.6).

2.2.1 Method of Comparison

The direct method of comparison between one subject and another rests at the core of the comparison algorithm. This piece of the design was created first, since its validity and accuracy are what make the algorithm useful. Inspiration for the design presented here came heavily from the successful algorithm presented in "Face Recognition in Hyperspectral Images" [3].

In choosing how to compare subjects were compared, the following aspects were considered:

1. Use of hyperspectral data
2. Accuracy
3. Data storage

The use of hyperspectral data and accuracy are intuitive, since the system is designed to demonstrate the potential to use hyperspectral data as a biometric. The data storage aspect is less intuitive but carries high importance. All data used for comparison must be stored, and storing more data leads to a greater requirement for memory space. This data must also be copied many times in the process of comparison and larger copy operations would significantly increase the runtime of the algorithm.

To address our considerations, the method of comparing the hyperspectral content of a set of features at fixed positions was adopted from "Face Recognition in Hyperspectral Images" [3]. This allowed for the algorithm to use a greatly compressed database, since it only needed to store the data for the relevant features. The method of taking the distance summed over multiple features had also been shown to be accurate previously and entirely reliant on the hyperspectral content of the feature rather than any gradients or spatial data [3].

The features (storage previously defined in 2.1.3) each consisted of nine by nine pixel boxes defined around five locations (see 2.1.2) where each pixel was represented as a vector of intensities in the various hyperspectral bands.

To compare the features, the algorithm would first average the vectors for each three by three square of pixels (resulting in nine vectors total) to attempt to reduce the affect of outliers. After obtaining these nine vectors, a distance would be calculated by taking the Mahalanobis distance between each of the nine vectors from each of the two features being compared (resulting in 81 distance calculations). The result of each Mahalanobis distance would then be summed to form the total distance for the given feature as another attempt to reduce the affect of outliers. Once a distance was calculated between each of the features in the two subjects with the same spatial location, the five feature distances were summed to obtain the final distance between two subjects.

In the vector distance calculations, the algorithm uses Mahalanobis distance calculation (see below) as opposed to Euclidian distance because the data varies differently in different bands as observed by Zhihong, Healey, Prasad, and Tromberg [3]. The Mahalanobis distances takes into account the expected variance of intensity in each band individually by using a precision matrix in the calculation. The precision matrix forms by using the data from the database as a sample of variation in different bands across the facial tissue.

$$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^T S^{-1}(\vec{x}-\vec{y})}$$

Equation 1 – Mahalanobis Distance calculation where S is the covariance matrix and x and y are the column vector results of averaging boxes of 3 by 3 pixels

2.2.2 Sequential Algorithm Implementation

In developing the Parallel Comparison Algorithm, it was first necessary to create a simple sequential algorithm for benchmarking and to use as a starting point for parallelization. Figure 5 depicts how the algorithm progresses through the comparisons, which define its runtime.

The sequential algorithm progresses in a logically straightforward manner, comparing the target passed to it against each subject in the database one at a time and storing the distance if it is one of the shortest five distances seen so far along with the name and access level of the potential match. Within the comparison of each database entry to the target, all features are compared in a set order with the distance between features being calculated by the two innermost loops.

For large datasets, the runtime of this algorithm, defined by the outermost loop, is on the order of O(n) where n is the number of entries in the database (linear time increase for additional subjects added to the database). This runtime could of course greatly increase or decrease if the algorithm was modified in further work to compare more features or do a more detailed comparison of features since this would result in many more iterations of the inner loops. The algorithm is thus relatively slow for extremely large databases and poorly scalable. The parallel algorithm will focus on improving these flaws.

2.2.3 Optimization of Loop Order and Memory

The first consideration in converting the sequential algorithm to a parallel algorithm focused on removing the O(n) runtime caused by the outer loop iterating over each entry in the database. It would be ideal to achieve O(c) runtime (no time increase for additional subjects added to the database except from hardware constraints), but even O(log(n)) runtime would be a vast improvement (logarithmic time increase for additional subjects added to the database).

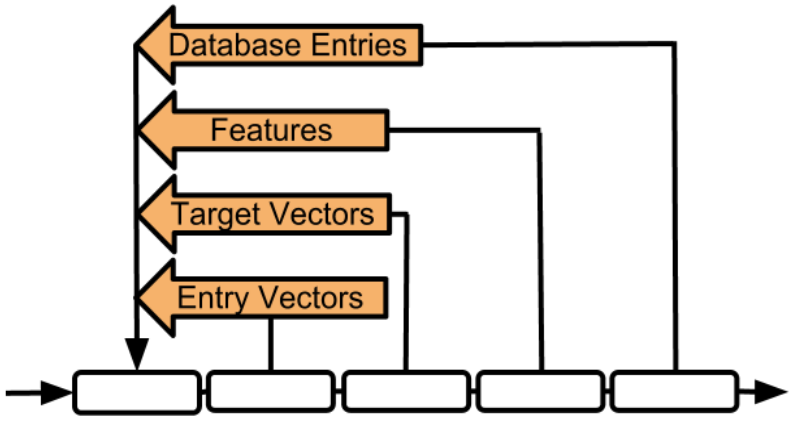


Figure 5 - Sequential algorithm loops

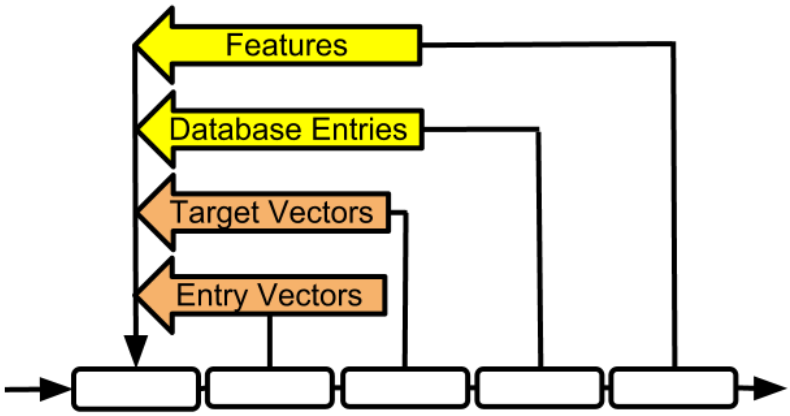


Figure 6 - Changing of loop order from the sequential algorithm to the parallel algorithm

The seemingly logical way to overcome this problem is to parallelize all the loops. This would in theory solve the problem and yield O(c) runtime, but in practice it does not allow for enough data granularity to optimize performance on the hardware. The GPU hardware used by the parallel algorithm has special memory caches (discussed later in this section) that are of limited size. In order to optimize use of these caches, we would like to break the data into its smallest comparable segments, which are features. This desire results in the inversion of the "Feature" and "Database Entries" loops from the serial algorithm

The loop implementation in Figure 6 now calculates the distances from all features of one type to that feature of the target for each iteration of the outer "Features" loop. The distances are stored into arrays and summed later in parallel to calculate the distance between each subject and the target at time cost O(c) (not displayed in figure), which is insignificant in dealing with large data sets.

The above implementation also yields another novel result by putting the features loop on the outside. Since there are a fixed number of features to be compared, the runtime of the "Features" loop is O(c) and therefore does not need to be parallelized as it is only repeated once. We now only need to parallelize the three inner loops to achieve a great speed up.

Returning to the topic of data granularity and hardware memory cache constraints, the data, now in more granular form, needs to be fitted to memory caches in the hardware to optimize read and write times. There are two kinds of caches that we will use. Constant memory can be read rapidly from anything executing on the GPU and is read only. It will remain having whatever value is loaded into it until the host (computer controlling the GPU) loads something different into it. Shared memory is specific to a group of threads executing on the GPU referred to as a block (512 threads in this implementation). The shared memory can be read and written to rapidly.

For the implementation preformed here, the feature of the target and the precision matrix used will be constant for all Mahalanobis distance calculations preformed, so they will be loaded into constant memory. Since the feature data for each database entry is only used once, this will be stored into the shared memory specific to each block.

The implementation done in this project used a GeForce GT 650M GPU made by NVIDIA, which has 65536 bytes of constant memory and 49152 bytes of shared memory per block.

Performing calculations to validate implementation:

$$\left(4\frac{bytes}{float} * 30\frac{floats}{vector} * 9\frac{vectors}{feature}\right) + \left(4\frac{bytes}{float} * 900\frac{floats}{precision\ matrix}\right) = 4680\ bytes$$

Equation 2 - calculation of the bytes required for constant memory

4670 is inferior to 65536 so constant memory will fit the precision matrix and target feature data easily.

$$\frac{49152\ bytes}{\left(4\frac{bytes}{float} * 30\frac{floats}{vector} * 9\frac{vectors}{feature}\right) + \left(4\frac{bytes}{float} * 81\frac{floats}{feature}\right)} = 35\ features$$

Equation 3 - calculation of the number of features per block based on memory limitations

Therefore each block of threads has enough memory to handle up to 35 features.

The constant memory has ample memory space for the feature from the target we want to store along with the precision matrix for that feature type. We also see that each block we schedule can have a maximum of 35 features from database entries based on the size of shared memory.

2.2.4 Thread Parallelism

Having addressed the memory concerns and loop ordering, this section moves on to discuss how groups of parallel threads are scheduled to preform operations.

When using CUDA to program a GPU, processes are written as kernels since they interact directly with the hardware rather than interfacing through an operating system. Each kernel in the case of our code schedules one block of 512 threads. This thread number was chosen since either 256 of 512 threads per block are considered to be optimal for NVIDIA's Kepler architecture (the most recent GPU hardware architecture supporting CUDA).

Thread limit on features that can be processed by one kernel/block:

$$\frac{512\ \frac{threads}{block}}{81\ \frac{threads}{feature}} = 6\frac{features}{block}$$

Equation 4 - calculation of the number of features per block based on thread limitations

After preforming this calculation, it is clear that the number of threads and not the memory size will be the limiting factor for this dataset in calculating the number of features to be processed per block. It is important to note that the additional granularity of data allowed six features to be processed per block

still as opposed to the five that belong to a single entry, resulting in more optimal hardware resource usage.

Before deciding on the one block per kernel implementation, there was a decision to make between scheduling multiple blocks in one kernel and scheduling multiple kernels to deal with each feature. The choice to schedule only one block per kernel was decided upon to achieve better granularity for hardware concurrency that will be discussed in the next section.

2.2.5 Concurrency

Concurrency exploits the GPU hardware's ability to preform multiple operations at the same time [6]. In the case of the Kepler architecture, data can be simultaneously copied to and from the GPU while many separate kernels run at the same time.

Referring to the previous section, we chose the approach of multiple kernels as opposed to multiple blocks so that we could optimize hardware memory bandwidth and processing power. The smaller kernel size allows the hardware to begin processing the first kernel more quickly, and the scheduling of multiple simultaneous kernels allows us to overlap memory copy and processing to achieve a greater speedup and flatten the loop over the database. Without concurrency, each row in the Figure 7 would have to be placed end to end in one row, resulting in a much longer processing runtime. The current implementation only uses five instances of the same kernel, since this should be more than enough to saturate the memory copy bandwidth while maximizing the use of GPU processing power although Kepler can handle up sixteen concurrent kernels (same as older Fermi Architecture) [6].

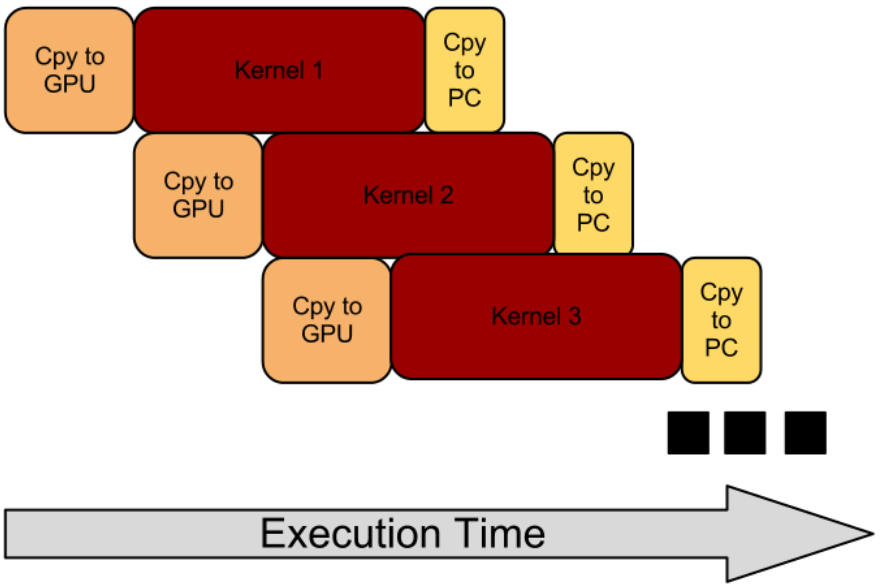


Figure 7 - Diagram of the concurrent kernel and memory copy pattern

In order to implement the concurrency used in this algorithm, it was necessary to map five buffers from the host memory into the memory space of the GPU as input buffers and another 5 buffers for output, which are pinned memory. The input buffers are one per kernel and allow the implementation of asynchronous memory copies of the data for each kernel run onto the GPU. The output buffers are implemented to hold the outputs of each kernel until the parallel sum is done to calculate the total distance for each subject.

2.2.6 Final Implementation

With concurrency included in the final implementation shown in Figure 8, the only loop still remaining is the constant size loop across the five features. Examining the expected runtimes, we now have the "Features" loop which O(c), the parallel processing of feature comparisons O(c), and the summing to come to a final subject distance O(c).

This gives us a final runtimes of O(c) for filling an array with the distances of all subjects. From here, a variation of a CUDA parallel reduction algorithm is used to select the closest five matches and they are ordered and output. This algorithm has a runtime of O(log(n)) and is well documented on the internet in many places [7].

This gives our algorithm a final runtime complexity of O(log(n)). Much of the runtime will however be determined based on the hardware used. Memory transfer bandwidth is generally the major issue and in large data systems, the theoretical speed of the algorithm will be bounded by the data transfer rate of the hard disk holding the database. Smaller databases may be preloaded in host RAM in which case the memory transfer rater between the GPU and host is likely to become the limiting action in processing data. This makes calculations other than the big O calculations useless, as hardware will vary widely by system.

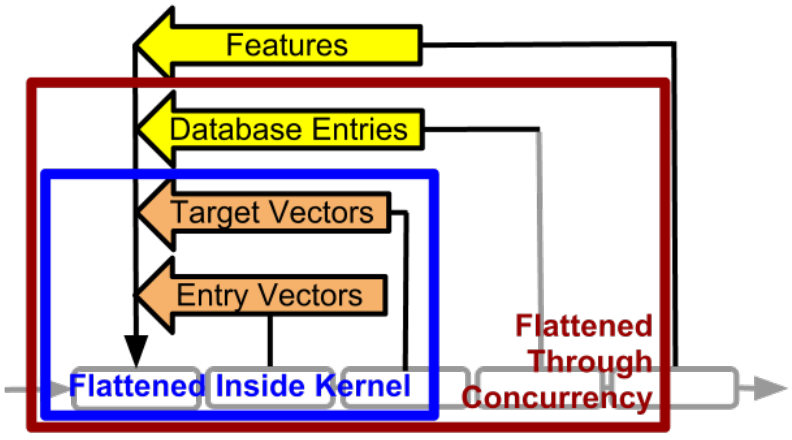


Figure 8 - Final parallelization of the algorithm

It is important to note that the algorithm presented here was designed to be scalable. More kernels, features, feature vectors, etc. could easily be added easily in any dimension since these numbers were all purposely defined as preprocessor macros in the implementation. The hardware specifications were accounted for in macros similarly meaning that by simply putting in the memory constraints and target threads per block of the hardware, the algorithm will optimize the number of features being processed per block. All numbers given in the previous descriptions were specific to the hardware and data used in the work done for testing later presented in this paper.

2.3 Wireless Door Locking Mechanism

The door locking mechanism operates with five major blocks as can be seen on Figure 9: an Arduino Uno R3, a wireless interface (Arduino WiFi Shield), a wireless router, a computer wireless module, and a PCB (Printed Circuit Board) containing 4 H-bridges, each connecter to one of four motors. The original design of the door locking mechanism included a single electric strike, but was replaced by motors because the scope of the project only required the simulation of door locking and unlocking and the motors were much cheaper than an electric strike. Four motors give a more accurate representation of the application of the design to a security system with more than one level of security.

The Arduino can take power inputs from 5 to 15V, with an ideal range of 7-12V. The original design had a 9V power input, but this lead to inconsistencies in the amperes released to the digital output pins. This resulted in inconsistent operation of the motors, so the design now uses a 12V power input to the Arduino from the lab bench. The WiFi shield is powered through the Arduino and plugs straight on top of for communication efficiency.

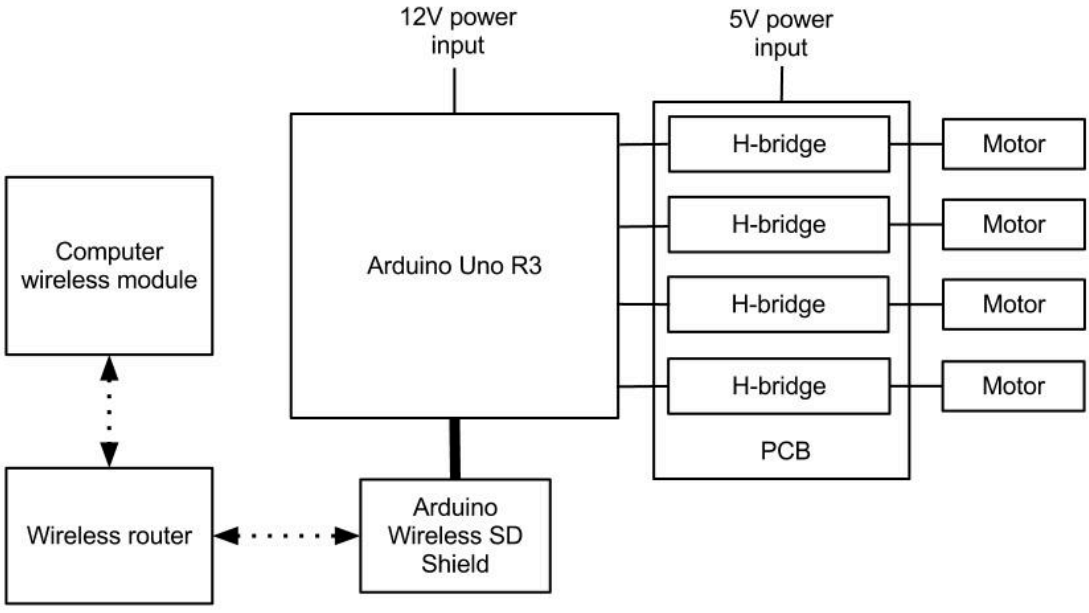


Figure 9 - Block diagram of the Wireless Door Locking Mechanism

The motors and PCB are both powered by a 5V input. The original PCB design required a 12V input, however this was not needed for the revised design explained in section 2.2.2. Finally, the PCB's H-bridges power the motors.

2.3.1 Overall functionality

The underlying element in the wireless door locking mechanism is the wireless network, which enables communication between the Arduino and Computer. In this design, a Netgear Wireless-G Router (model WGR614) sets up the wireless network. However, the design will function with any type of wireless network with minor modification of the Arduino program.

The Arduino acts as a basic HTML (HyperText Markup Language) web server using the WiFi Arduino library. Through this server, it outputs one web page called the Motor Control Page. This page includes four links, one for each motor. When a link is clicked, it sends a GET request (HTTP method to retrieve information from a server) to the server running on the Arduino through a packet. The Arduino then reads four characters of the packet indicating which motor should be operated and acts accordingly. This design choice was made over other protocols because other attempts (using an HTML web form and the POST HTTP protocol) proved to create excessive packet corruption that made the design nearly inoperable.

To run the motors, the Arduino switches one of its eight digital output pins to high (5V) from low (0V) as can be seen on Figure 10. The motors are then run using electronic circuits called H-bridges. H-Bridges are circuits that allow a voltage to be applied across the motors in either direction. For each motor, the Arduino can enable forward and backward operation. To meet requirements, the Arduino outputs a high voltage to the forward direction pin of the motor for three seconds, waits for one second, and then outputs a high voltage to the reverse direction pin of the same motor. The delay() function of the basic Arduino programming library is used to accurately follow this timing.

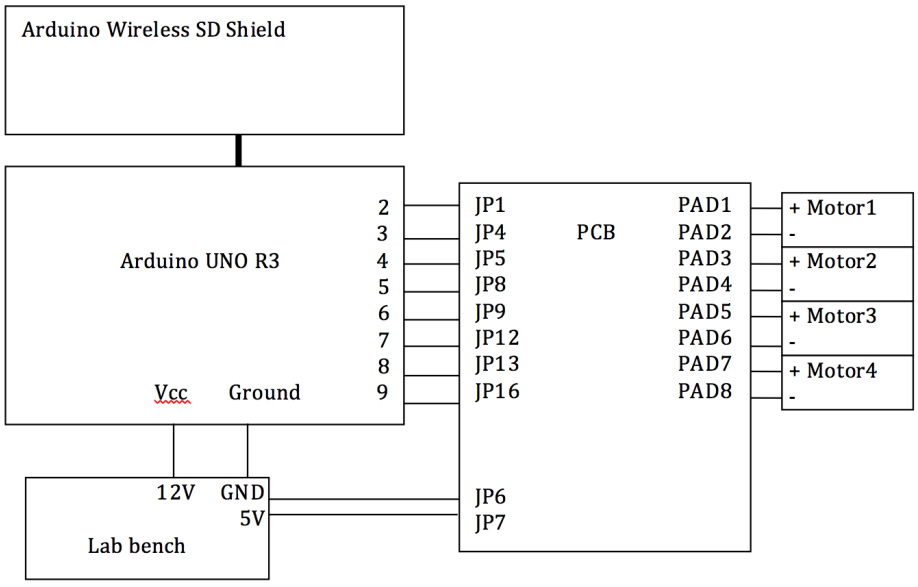


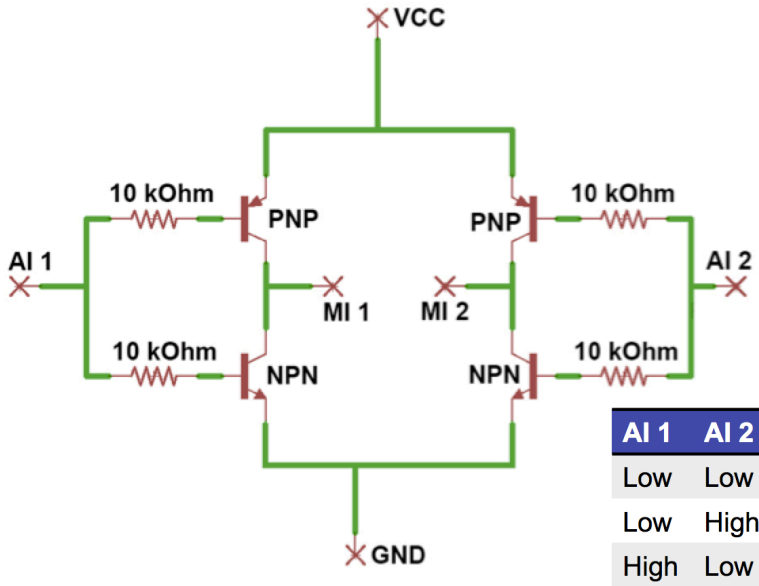
Figure 10 – Pin out of the Arduino and PCB

2.3.2 PCB Design and H-bridge functionality

The H-Bridge PCB plays the essential role of controlling the simulated locks (motors) of the Wireless Door Locking Mechanism. Since the Arduino can only 30-50mA of current, it cannot properly turn the motors using its own power. This results in a need for at minimum an amplifier to turn the motors.

An H-bridge was chosen over a simple transconductance amplifier since it allows the Arduino to turn the motor in both directions (open and close the lock). An H-bridge circuit can take a nearly infinite number of forms choosing between different types of transistors, layouts, and other functionality and parts. In the implementation used with the Wireless Locking Mechanism, the simplest, most robust, and cheapest possible H-bridge was constructed. Any safety features, additional functionality, etc. could more easily and cheaply be implemented in the Arduino program than in the PCB hardware.

The Figure 11 design implements an H-bridge using four general purpose BJTs (two NPN, 2PNP), and four 10kΩ resistors. AI 1 and AI 2 represent inputs from the Arduino logic pins; MI 1 and MI 2 are the two leads of the motor. These parts were selected for our prototype model since they had low cost and were easily accessible. The BJTs were used over CMOS transistors since they are more tolerant of heating and static, making them easier to prototype with. The resistors were already a sunk cost from a past design so it was cheaper to use four rather than finding two resistors with half the resistance value. The resistors functioned to ensure the base current and voltage of the BJTs could not easily exceed the rated values for the BJTs although this was already unlikely given the Arduino's output specification [8][9][10]. The details of the general functioning of a H-bridge are well described in many Internet sources, which may be consulted for further detail [11]. After designing the H-bridge that would be used in the Wireless Door Locking Mechanism, four H-bridge circuits were placed on a PCB to make the design compacted. Please refer to Appendix B for the PCB layout.



AI 1	AI 2	Motor control
Low	Low	None
Low	High	Reverse
High	Low	Forward
High	High	None

Figure 11 - Single H-bridge circuit and truth table

3. Design Verification

The detailed requirements and verifications table can be found in Appendix A.

3.1 Feature Extraction Algorithm

The data obtained was hyperspectral with radiance information for each pixel for 65 wavelength bands ranging from 450 nm-1100 nm. A sample of the locations of the extracted features for one particular subject can be seen in Figure 12. As seen from the Figure, the locations of the extracted features (marked with +), are quite accurate and well within a 10% error range. This algorithm was run on 36 subjects, and was successful 34/36 times with an error of less than 2% from the actual position obtained by observation. The unsuccessful runs were due to some inconsistencies in the data and improvements can be made to make the algorithm more robust and improve its capability to resolve such conflicts. The database was built for demo with 26 subjects, using the information from the pixels in the yellow squares in Figure 12, and a variable number of randomly generated entries (208-1000) for parallel algorithm benchmarking.

3.2 Parallel Comparison Algorithm

The requirements placed on the Parallel Comparison Algorithm during the design phase dealt only with specifying accuracy and output.

3.2.1 Accuracy and Output Verifications

The high level requirements for the Parallel Comparison simply required that the algorithm output the top five closest matched for a target in order and that the target appear within the top five at least fifty percent of the time to prove the algorithm was functioning better than random chance.

Left cheek	x	285
	y	248
Right Cheek	x	382
	y	249
Lips	x	331
	y	300
Forehead	x	332
	y	248

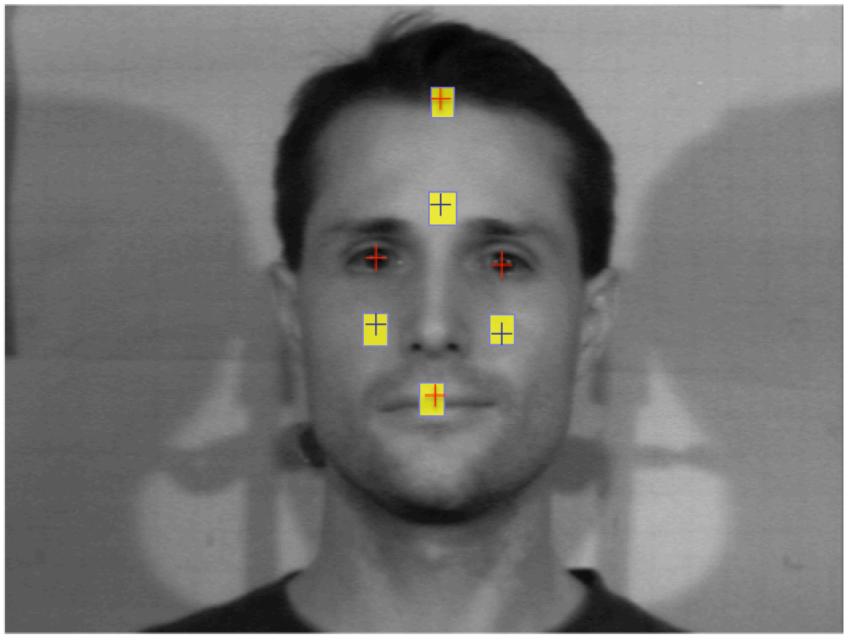


Figure 12 - Sample results obtained from the feature extraction algorithm

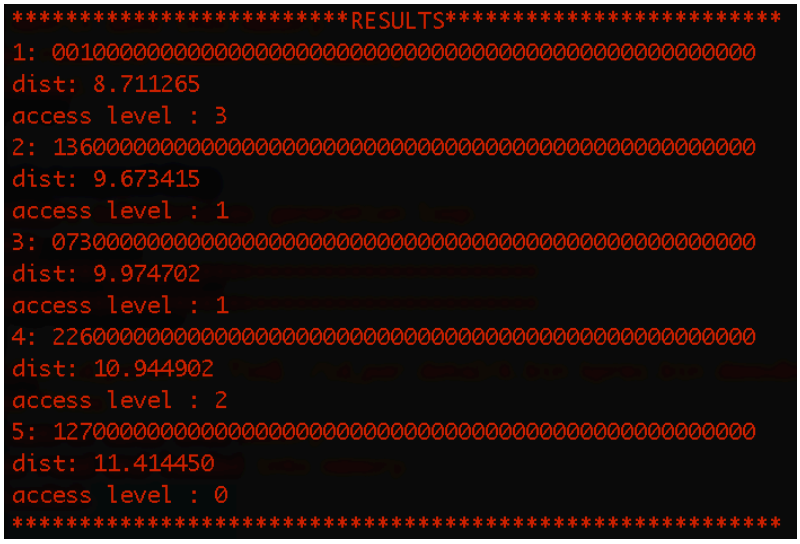


Figure 13 - Program output screenshot for subject 001

As can be seen from Figure 13, the algorithm output is in an acceptable format. First, the name field of the match is printed (50 characters that were numbers 000-234 followed by forty-seven zeros for the test preformed). The next line contains the distance of the subject followed by the access level associated with that subject.

The algorithm's accuracy identifying a subject was tested against a database of 234 subjects. It identified the target in the top five results 65.38% of the time, exceeding the minimum required performance by fifteen percent.

3.2.2 Algorithm Speed Analysis

In addition to our requirements, we benchmarked the algorithm speed up for the parallel algorithm by running it multiple times against a database and measuring the real time using the "time" command from the command line [12]. The same was done with our originally develop sequential algorithm and then the two were compared to compute the speed up as can be seen in Table 1.

Table 1 - Algorithm runtimes benchmarks

Database Size	Avg. Serial Runtime (ms)	Avg. CUDA Runtime (ms)	Avg. Speedup
26	325.2	124.34	2.62
234	1066.2	321.80	3.31
1000	5588	1197.46	4.67

The results show a significant and increasing speedup with datasets of increasing size. This speed up should peak eventually based on the data transfer rate of the hard disk where the database is stored. This peak will be variable based on the computer hardware used and the speed up achieved on smaller data sets will also vary with GPU hardware. The testing computer was a laptop so it is likely speed ups seen may be significantly better than those seen in testing.

3.3 Wireless Door Locking Mechanism

The door locking mechanism has to properly receive positive signals from the computer block. To test this, a simple program was loaded onto the Arduino that would output 5V to a pin whenever a certain signal was received through a packet. This was the only requirement that was not completely met. There are packet lot issues with this method of communication. Over multiple tests, we have found that the Arduino will receive the correct packet 70-80% of the time. This depends on the location of the wireless router and how much interference is present from other 802.11 devices.

The second requirement is that if a positive signal is received, the door locking mechanism must engage the motors. The Arduino program detects correct and incorrect packets and writes this out to the serial line connected to the computer. We were therefore able to identify when a motor should be rotating and when it should not. Every motor rotates successfully whenever a correct packet is sent, over a series of 10 tests per motor.

The final requirement is that the motors rotate the correct amount. Over all performed tests, the motors all had a proper rotation sequence of three seconds in one direction, a one-second pause, and three seconds in the other direction.

4. Costs

4.1 Parts

Table 2 includes the costs of the necessary parts to build our project. Quantities do not include any additional parts we had to purchase for testing purposes. Additionally, the actual and retail costs do not include shipping prices that were paid on any of the items.

Table 2 - Parts costs

Part	Manufacturer	Amount	Retail Cost per part	Actual Cost per part	Retail Cost	Actual Cost
Arduino Uno R3 Microcontroller	Arduino	1	$29.95	$23.76	$29.95	$23.76
Arduino WiFi Shield	Arduino	1	$84.95	$84.95	$84.95	$84.95
PNP BJTs (8) (2N4403)	On Semiconductor	8	$0.46	$0.34	$3.69	$2.72
NPN BJTs (8) (2N4401G)	On Semiconductor	8	$0.043	$0.15	$0.34	$1.20
10 kΩ resistors (16)	TE Connectivity	16	$0.41	$0.41	$6.56	$6.56
Motors (4)	NMB Technologies Corp.	4	$6.44	$6.44	$25.76	$25.76
Total					**$151.25**	**$144.95**

4.2 Labor

Table 3 shows the labor costs associated with each member of the team and the total labor cost resulting from this. The overhead cost was used to estimate the real value of work.

Table 3 - Labor Costs

Name	Rate/Hour	Overhead(x2.5)	Hours	Total
Chris	$45	$112.5	240	$27,000
Timothee	$45	$112.5	240	$27,000
Akshay	$45	$112.5	240	$27,000
			Total Labor Cost:	$81,000

5. Conclusion

We have demonstrated the potential of using hyperspectral information for facial recognition using GPUs to perform comparisons with large databases. Most of the requirements were met. The Feature extraction module extracted the correct locations of the features for 34/36 subjects over multiple sessions (within the error range) and constructed a database well suited for comparison. The comparison algorithm was quite successful and exceeded requirements. The target was in the top five for 65% of the subjects and the top match for 20% of the subjects. There were significant speed-ups obtained for larger databases proving the potential of GPU processing to perform the computations required for hyperspectral facial recognition. The wireless door locking mechanism was fairly successful except for the packet loss issue. The H-bridge circuits on the PCB and the Arduino code performed up to specification.

Various steps were taken to maintain the integrity of our project and stay consistent with the IEEE Code of Ethics. Our project followed the first pledge of IEEE ethics [13] by creating a technological solution that can increase public safety in a variety of applications (crime fighting, counter-terrorism, etc.). Our project embodies the third pledge of IEEE ethics by clearly stating the limits and range of our end result, as well as clearly defining the scope of available data and making claims accordingly. To maintain academic honesty and integrity of our results, the data set was divided into two- demo and training. The demo database was never tested on during the development of the algorithm, and the training database was not used for our final presentation of results.

Our algorithm will scale well and show even more significant speed-ups with larger data sets. For very large data sets, the only limiting factor will be the time involved in hard disk reads instead of the processing time related to the actual algorithm. The algorithm is thus very architecture dependent and as computer architectures continue to improve, the algorithm will become even more viable.

Some future work that can be done with the project includes developing a more robust feature extraction algorithm using more advanced image processing techniques, for application on more complex real world data. Using either a feedback feature that will make the computer re-send the packet until it is received correctly or a more robust communication system can alleviate the packet loss issue with the door locking mechanism. We would also like to develop cheap hyperspectral cameras as has been shown by past papers [14]. As hyperspectral sensors become cheaper, the technology has the potential to be widely applied in commercial security settings that require facial recognition. Future applications could also look into using data produced by our algorithm as part of a multifaceted approach based on algorithm fusion. This has been shown to be more effective than examining a single feature type for unique person identification [15].

References

[1] Denes, Louis J., Peter Metes, and Yanxi. Liu, “Hyperspectral Face Database,” *Tech. Report CMU-RI-TR-02-25*, Robotics Institute, Carnegie Mellon University (October 2002)

[2] Tinsy John Perumanoor, “Visible Versus Neat-Infrared Light Penetration Depth Analysis In An Intralipid Suspension As It Relates To Clinical Images”, The University of Texas at Arlington (August 2008)

[3] Zhihong Pan; Healey, G.; Prasad, M.; Tromberg, B.; , "Face recognition in hyperspectral images," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.25, no.12, pp. 1552- 1560, Dec. 2003

[4] D. Pissarenko, "Eigenface-based Facial Recognition." Eigenface-based Facial Recognition., Feb. 13, 2003 Available: http://openbio.sourceforge.net/resources/eigenfaces/eigenfaces-html/facesOptions.html [Dec. 1, 2012].

[5] W. Zhao, R. Chellappa, and P. J. Phillips. "Subspace Linear Discriminant Analysis for Face Recognition." Technical Report CAT-TR-914, Center for Automation Research, University of Maryland, 1999

[6] Rennich, Seteve. “CUDA C/C++ Streams and Concurrency,” NVIDIA, [online] 2012, http://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf (Accessed: 12 December 2012)

[7] Harris, Mark. “Optimizing Parallel Reduction in CUDA,” University of Oxford, [online] 2012, http://people.maths.ox.ac.uk/gilesm/cuda/prac4/reduction.pdf (Accessed: 10 December 2012)

[8] *PNP General Purpose Amplifier*, datasheet, Fairchild Semiconductor Corporation, 2001. Available at: http://inst.eecs.berkeley.edu/~ee105/fa07/labs/2N4403.pdf

[9] *General Purpose Transistors NPN Silicon*, datasheet, Semiconductor Components Industries, LLC, 2010. Available at: http://www.onsemi.com/pub_link/Collateral/2N4401-D.PDF

[10] *Arduino ™ Uno Rev 3*, schematic, Arduino 2012. Available at: http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

[11] H-Bridge Motor Driver Using Bipolar Transistors., *Robot Room*, [online] 2012, http://www.robotroom.com/BipolarHBridge.html (Accessed: 5 December 2012).

[12] time(1) - Linux man page., *die.net*, [online] 2012, http://linux.die.net/man/1/time (Accessed: 11 December 2012).

[13] "IEEE Code of Ethics." IEEE Available: http://www.ieee.org/about/corporate/governance/p7-8.html [Dec. 12, 2012].

[14] R. Habel, M. Kudenov and M. Wimmer, "Practical Spectral Photography", *Computer Graphics Forum (Proceedings EUROGRAPHICS 2012),* vol. 31(2), pp 449-458, May 2012

[15] David M. Ryer, Trevor J. Bihl, Kenneth W. Bauer, and Steven K. Rogers. “Quest Hierarchy for Hyperspectral Face Recognition”, Hindawi Publishing Corporation, Advances in Artificial Intelligence, Volume 2012, Article ID 203670, 7 February 2012.

Appendix A Requirement and Verification Table

Table 4 - Requirements and Verifications

Block	Requirement	Verification	Status
Feature Extraction Algorithm	Data must be hyperspectral - Wavelength of data between 400 and 1200 nanometers. - Intensity information is present in different bands.	- **Testing Procedure: Plot a sample of the data with reflectance as a function of wavelength. Ensure that the data has intensity information present across a range of at least 300nm of wavelengths between 400 and 1200nm.**	**Y**
Feature Extraction Algorithm	Data can be recognized by the computer - Data is in a file format readable by the Linux operating system - Data can be read and used by a CUDA program	- **Testing procedure: Write a simple CUDA program that opens a file from the data's format, reads the content of that file and prints these contents in any form to the screen.**	**Y**
Feature Extraction Algorithm	Databases are sufficiently and correctly populated. - Each database (testing and demo) has at least 20 unique entries. - Each entry has hyperspectral intensity information for each feature. - Each Database includes at least 50 total entries including automated generated data.	- **Testing Procedure: Open each database on the computer. For each database:** - **Use the computer's database software to output the total number of elements and ensure it is greater than 50.** - **Also output the number of unique elements and ensure it is greater than 20.** - **Finally, run a database query to output the number of elements with incomplete fields. Ensure that this returns 0.**	**Y**
Feature Extraction Algorithm	Features are extracted correctly	- **Testing procedure: run the following test 5 times: (need 4 positive results)** - **Input a hyperpectral image to the algorithm. Pre-identify locations of comparison and handler features to be extracted by the algorithm. Use print statements in the code to output the algorithm's identified location of these features. Verify that handler and comparison features identified by the algorithm are within a 10% of image dimension range from the pre-identified**	**Y**

		locations.	
Parallel Comparison Algorithm	Comparison of a new picture of an individual with an old, existing picture in the database of the same individual under similar lighting/orientation conditions should ideally yield that individual as best match	- **Testing Procedure: Run the following test 10 times:** - **Obtain two hyperspectral images of an individual with the same facial orientation and lighting. Place one in the database, with at least 19 other unique photos also in the database. Run the second test image with the program. Make sure the right photo is selected in the top 5 at least 50% of the times.**	**Y**
Parallel Comparison Algorithm	The algorithm should output its top 5 matches in order and the subject numbers of the people associated with them. The target should be in those top 5 matches 50% of the time.	- **Testing Procedure: Output the subject number of the target. Output the subject numbers of the people associated with the top 5. Use an indicator on the screen to indicate if the target is in the top 5 or not.**	**Y**
Door Locking Mechanism	The door locking mechanism gets the required information from the wireless packet for the access level of the top detected match.	- **Testing Procedure: Pass through 20 photos to test.** - **Run the algorithm with an image that exists in the database. Use a print statement in the code to output the user's access level to the screen. Make sure the correct motors have rotated based on the access level.**	**Y**
Door Locking Mechanism	The door locking mechanism must properly receive positive signals from the computer block.	- **Testing Procedure: Write the Arduino with a simple program that outputs a 5V voltage to pin 1 when a signal is received. Send the signal through the computer, use a voltmeter at pin 1 to verify the voltage switches to 5V.**	**Y/N**
Door Locking Mechanism	If a positive signal is properly received from the Computer Block, the door locking mechanism must engage the motors	- **Testing procedure: Pass a signal corresponding to each access level to the door locking mechanism; ensure that the proper motors rotates any amount.**	**Y**

Door Locking Mechanism	The motors must rotate the correct amount	**- Testing Procedure: Pass an image that exists in the database with security level 4. Ensure that all four motors rotate for 3 seconds in one direction, stop, wait for 15 seconds, and rotate for 3 seconds in the other direction.**	**Y**

Appendix B PCB layout

In Figure 14, the green circular pads represent where external inputs and outputs are connected. The pads on the left represent all the Arduino inputs, the pads on the right represent the motor control outputs, and the two pads at the top are power and ground. They are grouped in such a way to simplify the connecting of external outputs and reduce needed wire length. It is also important to note the increased trace thickness on board design in areas that have potential for greater current flow. Everything on the PCB has at least a safety factor of two for handling the voltages and currents it is expected to see.

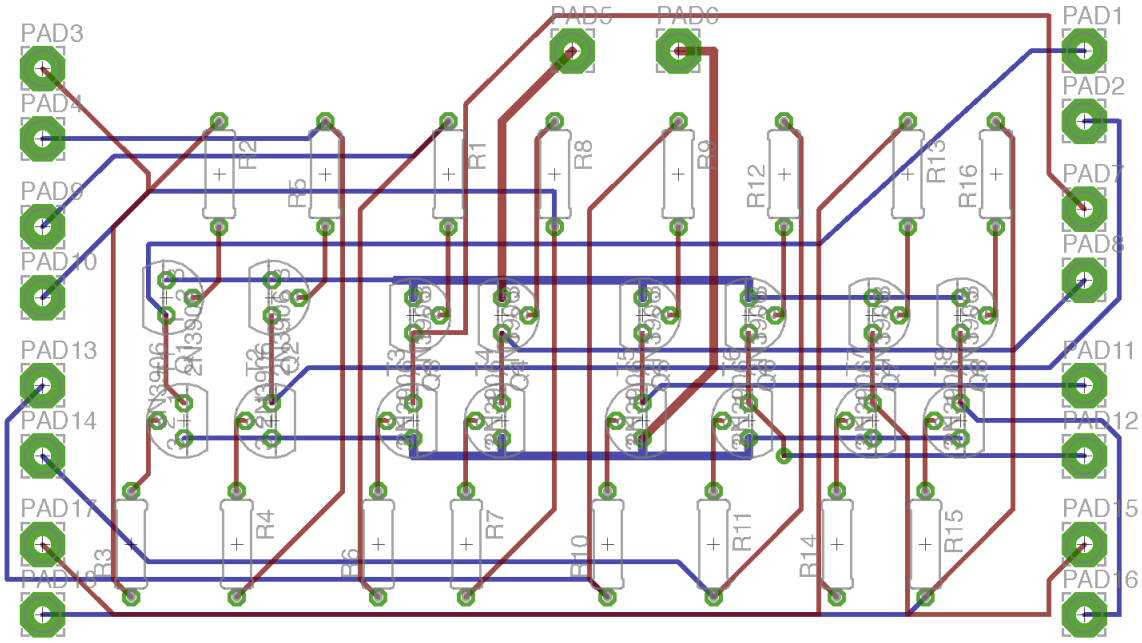


Figure 14 - PCB layout