A Gesture Recognition System using Smartphones

Luís Tarrataca
Instituto Superior Técnico
Av. Prof. Dr. Aníbal Cavaco Silva
Lisbon, Portugal
tarrataca@gmail.com

ABSTRACT

The need to improve communication between humans and computers has been instrumental in defining new modalities of communication, and accordingly new ways of interacting with machines. Humans use gestures as a means of communication. The best example of communication through gestures is given by sign languages. With the latest generation of smartphones boasting powerful processors and built-in video cameras it becomes possible to develop complex and computational expensive applications such as a gesture recognition system.

We present a gesture recognition prototype system for smartphones. We describe two segmentation processes, based on RGB and HSI color spaces, used to distinguish between foreground and background objects and conclude that RGB based rules outperform those of HSI. A model based representation alongside a template based technique were developed for hand posture classification. Comparison between methods evidences the latter superior recognition rates. We employed hidden Markov models to model the sequences of hand postures that form gestures. Hidden Markov models proved to be an efficient tool for dealing with uncertainty due to their probabilistic nature.

We trained a set of three gestures and based on user interaction obtained an average recognition rate of 83.3%. We concluded that the latest smartphone generation is capable of executing complex image processing applications, with the most penalizing factor being camera performance regarding capture rates.
Keywords: Development cycle, hidden Markov models, pattern recognition, skin detection, smartphone performance, system architecture.

1. INTRODUCTION

1.1 Motivation

The need to improve communication between humans and computers has been instrumental in defining new modalities of communication, and accordingly new ways of interacting with machines. Sign language is a frequently used tool when the use of voice is impossible, or when the action of writing and typing is difficult, but the possibility of vision exists. Moreover, sign language is the most natural and expressive way for the hearing impaired. Gestures are thus suited to convey information for which other modalities are not efficient.

When it comes to gesture recognition and human-computer interaction there is quite a lot of scientific work developed, but before proceeding it is important to distinguish between gesture and posture. This distinction, as well as the definition of both terms can be found in [Davis and Shah, 1994], namely:

- **Posture** - A posture is a specific configuration of hand flexion observed at some time instance;
- **Gesture** - A gesture is a sequence of postures connected by motions over a short time span. Usually a gesture consists of one or more postures sequentially occurring on the time axis.

With the latest generation of smartphones boasting powerful processors and built-in video cameras, the development of complex and computational expensive applications becomes achievable. Also, by developing gesture recognition software for smartphones it allows one to study the feasibility of such an interaction process.

1.2 Goals

The main goals of this paper focus on the areas of computer vision, pattern recognition and their applicability to embedded systems such as smartphones. These goals can be stated as follows:

- Develop algorithms that allow the recognition of hand postures, by utilizing a predefined set of hand postures;
- Develop algorithms that based on the hand posture recognized allow the recognition of a simple gesture-based language;
- Implement the selected algorithms on a smartphone with a built-in camera;

- Study and analyze the capabilities of smartphones to execute computationally intensive algorithms;
- Analyze the performance of the algorithms according to their efficiency and processing time;
- Study and analyze the capabilities of smartphones as image acquisition and processing units;
- Study the potential for code optimizations in order to speed-up the performance of the algorithms.

1.3 Problem Statement

The recent proliferation of latest generation smartphones featuring advanced capabilities (such as being able to run operating systems, adding new applications to better serve its users and multimedia features such as video and image capture) has allowed for the development of a compact and mobile system with many features that a few years ago belonged to the realm of a normal computer.

Yet due to their size and battery requirements even today's most evolved models have constraints not usually found on a regular desktop computer. This is especially true when considering that such devices are usually resource constrained in both memory and CPU performance. Since sign language is gesticulated fluently and interactively like other spoken languages, a gesture recognizer must be able to recognize continuous sign vocabularies and attempt do it in an efficient manner.

A vision based approach to gesture recognition can be realized by using the cameras that are included in most of today's smartphones and converting them into smart cameras. Smart cameras are cameras that include image processing algorithms. These algorithms might, for example, extract some features of images. In this case the data structures representing feature information are transmitted to the computation system's core, in order to be properly analyzed. One of the tasks of a smart camera might be the recognition of gestures identifying a command to be executed and properly notifying the system's core.

By developing an approach considering the real-time requirements associated to gesture recognition and respective constraints associated to embedded systems, such as smartphones, it becomes possible to study any potential issues that may arise.

1.4 Organization

The remaining sections of this work are organized in accordance with this gesture recognition overall system architecture.

Section 2 presents the different stages relating to the smartphone component, namely:

- Section 2.1 presents the segmentation process, denominated by "Skin Detection". In this process a thresholding operation is realized in order to allow for differentiation of background and foreground. This way it becomes possible to label sections of an image as belonging or not to a hand posture.
- Section 2.2 focuses on feature extraction algorithms that allow the determination of points of interest in a posture. Once the features describing a posture are obtained, they can be compared by a classification algorithm against a labeled database containing features of a sample posture population. The approaches taken toward feature extraction as well as posture classification and respective issues are presented in this section.
- Section 2.3 focuses on the methods developed and respective issues surrounding the use of hidden Markov models applied to gesture recognition. Once a posture has been labeled it needs to be contextualized with the remaining postures of a gesture in order to provide for semantic meaning in a probabilistic fashion, this is where the hidden Markov models come into play.

Section 3 discusses the approach taken towards system implementation namely the development cycle emplyed, as well as core components of the system architecture.

Section 4 focuses on the main experimental results affecting the gesture recognition system developed, such as overall smartphone and system performance.

2. GESTURE RECOGNITION

2.1 Skin Detection

Skin color has proven to be a useful and robust clue for gesture recognition, face detection, localization and tracking. Color allows for fast processing and experience suggests that human skin has a characteristic color, which is easily recognized by humans. In the scientific literature several authors provide an excellent introductory base for the topic of skin detection and respective issues see [Jones and Rehg, 2002], [Kovac et al., 2003] and [Vezhnevets et al., 2003].

When color is used as a feature for skin detection it is necessary to choose from a wide range of possible color spaces, including RGB, normalized RGB, HSI and YCrCb, among others.

2.1.1 RGB-based rules

Due to it is huge popularity and wide adoption in the electronics industry, namely digital cameras and computer monitors, the RGB color space presents itself as an adequate candidate to be used in color-based skin detection. A detailed analysis of the properties of the RGB color space is presented in [Yamaguchi, 1984].

Using the RGB model a simple pixel based heuristic is presented in [Kovac et al., 2003] that determines whether a certain pixel of an input image corresponds to the skin color. With the algorithmic rules of Figure 1, it becomes possible to label the pixels of an image as being skin or not. The obvious and main advantage of this method is the simplicity of the set of skin detection rules that leads to the construction of a fast classifier. On the other hand, the main difficulty achieving high recognition rates with this method is the need to find both good color space and adequate decision rules empirically. These rules are also subject to being highly influenced by the skin samples used and also by the lighting conditions of each of them.

Algorithm *SKIN-DETECTION* **(I)**
Input: I = an array containing the RGB encodings for all pixels
Output: O = an array containing the classification for each pixel of I

```
let m be the size of I
let R be the red component of a pixel
let G be the green component of a pixel
let B be the blue component of a pixel
for i ←1 to m
    do if R > 95 AND G > 40 AND B > 20 AND
                MAX(R,G,B) - MIN(R,G,B) > 15
                AND
                |R − G| > 15 AND
                R > G AND R > B
          then WHITE( O[i] )
          else BLACK( O[i])
  return O
```

Figure 1: Rules describing the skin cluster in the RGB color space at uniform daylight illumination.

There are however, according to [Vezhnevets et al., 2003], some aspects of the RGB color space that do not favor its use, namely the high-correlation between components and the mixing of chrominance and luminance data. These factors are important when considering the changing nature of color under different illumination conditions. The RGB color space has been used by innumerous authors to study pixel-level skin detection such as [Jones and Rehg, 2002] and [Brand and Mason, 2000].

2.1.2 HSI-based rules

Due to the different perception of color under uncontrolled light conditions it becomes useful to consider other color spaces. The Hue Saturation Intensity is a color model based on human color perception with explicit discrimination between luminance and chrominance. The explicit discrimination between luminance and chrominance properties of the HSI model presented a strong factor in the choice of a color space as can be seen in the work related to skin color segmentation presented in [Zarit et al., 1999] and [Mckenna et al., 1998]. However, [Poynton, 1995] points out several undesirable features of these color spaces, including hue discontinuities, and the computation of brightness which conflicts with the properties of color vision.

Several authors have presented work (see [Lin and Chen, 1991, Zhang and Wang, 2000]) regarding the use of HSI in color image segmentation. In this work the segmentation technique used, which is presented in Figure 2, only takes into account the Hue component of the HSI color space (as in [Bonato et al., 2005]), allowing for a faster computation and also limiting the amount of computation resources required.

In this case $H(i,j)$ represents the Hue values of a certain pixel and T_1 and T_2 describe inferior and superior thresholds. The thresholds values are required for determining when a pixel belongs to a region of interest and they should take into consideration the skin locus on the HSI color space. The resulting image contains a binary representation. An RGB-HSI conversion mechanism is presented in [Gonzalez and Woods, 1992], with expression 1 illustrating how to calculate the hue value.

$$H = \cos^{-1}\left[\frac{\frac{2}{3}\left(r-\frac{1}{3}\right)-\frac{1}{3}\left(b-\frac{1}{3}\right)-\frac{1}{3}\left(g-\frac{1}{3}\right)}{\sqrt{\frac{2}{3}\left[\left(r-\frac{1}{3}\right)^2+\left(b-\frac{1}{3}\right)^2+\left(g-\frac{1}{3}\right)^2\right]}}\right] \quad (1)$$

Algorithm *SKIN-DETECTION* **(I)**
Input: I = an array containing the RGB encodings for all pixels
Output: O = an array containing the classification for each pixel of I

```
let m be the size of I
let H be the hue value of a pixel
let T1 be a minimum threshold value
let T2 be a maximum threshold value
for i ←1 to m
    do H ←CONVERT_TO_HUE( I[i] )
        if T1 <= H <= T2
          then WHITE( O[i] )
          else BLACK( O[i] )
  return O
```

Figure 2: Rules describing the skin cluster in the HSI color space.

2.1.3 RGB- vs. HSI-based rules

RGB is one of the most widely used color spaces for processing and storing digital image data such as those acquired by digital cameras. The absence of direct support for the HSI color space requires a transformation process, from the hardware provided color space to HSI. This situation can potentially harm the use of HSI.

In order to evaluate the performance of the skin detection process in both HSI and RGB color spaces, a total of 850 images were served as input to both skin detection algorithms. The resulting binary images were then compared against the actual skin positions allowing for the determination of a precision value. HSI-based rules obtained an average precision of 38% against a value of 82% for the RGB-based rules.

2.2 Posture Recognition

In many practical problems, such as posture recognition, there is a need to make some decision about the content of an image or about the classification of an object that it contains. The basic approach toward object classification views it as a vector of measurements or features. A d-dimensional feature vector x typically represents the object to be classified. The classification process might actually fail, either because the posture's image contains noise, or a new hand posture that the system does not know was presented to it.

The remaining sections deal with the approaches developed toward feature extraction, respectively the Convex Hull and Simple Pattern Recognition approaches.

2.2.1 *Convex hull method*

According to [Cormen et al., 2003] the convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior. For convenience we denote the convex hull of Q by CH(Q). Intuitively, one can think of each point in Q as being a nail sticking out from a board. The convex hull is the shape formed by a tight rubber band that surrounds all the nails. For each hand posture it is possible to extract a number of useful features such as:

- The number of vertices identified;
- The x-coordinate and y-coordinate of each vertex;
- Polar angles and euclidean distance of each vertex with respect to an anchor point;

Feature comparison can be performed by calculating the euclidean distance between them. Graham's Scan, presented in Figure 3, (see [Graham, 1972] and [Cormen et al., 2003]) is an algorithm that computes the convex hull of a set of n points. It outputs the vertices of the convex hull in counterclockwise order and runs in $O(n \log n)$.

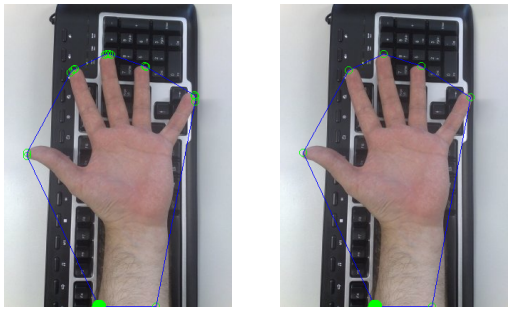
Algorithm *GRAHAM-SCAN* **(Q)**
Input: Q = input set of points
Output: S = a stack containing, from bottom to top, exactly the vertices of ConvexHull(Q) in counterclockwise order

1. let p_0 be the point in Q with the minimum y-coordinate, or the leftmost such point in case of a tie
2. let $< p_1, p_2, ..., p_m >$ be the remaining points in Q, sorted by polar angle in counterclockwise order around p_0 (if more than one point has the same angle, remove all but the one that is farthest from p_0)
3. PUSH(p_0, S)
4. PUSH(p_1, S)
5. PUSH(p_2, S)
6. **for** $i \leftarrow 3$ **to** m
7. **do while** the angle formed by points NEXT-TO-TOP(S), TOP(S), and p_i makes a non left turn
8. **do** POP(S)
9. PUSH(p_i, S)
10. **return** S

Figure 3: Graham's scan algorithm.

The result of applying Graham's scan to an input image can be seen in Figure 4. Notice that Graham's scan is only applied after a segmentation process has distinguished between foreground and background, thus enabling the creation of the set Q that is received as an argument by the algorithm.

As expected, and illustrated by Figure 4(a), several vertices are present for each finger. Finger borders are not solely constituted by a single point and thus the convex polygon has to incorporate several pixels surrounding the frontier of each finger. In order to solve this problem a simple clustering algorithm was employed which agglutinates vertices falling withing an error margin. Figure 4(b) illustrates the new vertices obtained after clustering has been performed.



(a) Before clustering.

(b) After clustering.

Figure 4: Convex hulls before and after clustering.

2.2.2 *Simple pattern recognition method*

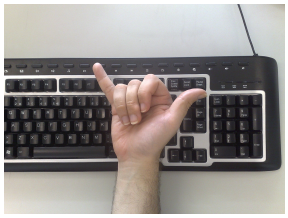
The next approach towards posture recognition although relatively simple, is substantially different from the previous method. The main idea behind the applicability of the convex hull was to obtain a worthy representation or model of a hand posture. Rather than trying to model a posture one can try to obtain some representation of a scene and match it against a set of scenes belonging to a feature labeled database.

Let $S = \{p_1, p_2, ..., p_m\}$ contain all skin-labeled pixels of a certain scene and let each skin-labeled pixel $p_i = (x - coordinate, y - coordinate)$, as illustrated by Figure 5(b). Then, if one wishes to compare the set S with any other set S' one simply has to calculate the number of positions that differ between sets in order to obtain an indication of the proximity between scenes. A brute force approach would simply check to see if every element in S and S' is present in the opposite set, for every element not found a simple counter would be incremented and keep track of the respective number of differences. This process would be rather inefficient in terms of speed. It is also necessary to consider that each pixel position has to keep x- and y-coordinates so there also occurs a significant impact in terms of memory usage.

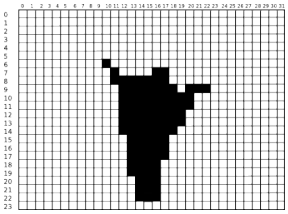
A better tactic would be to store the position of skin-pixels in a binary array, this way it becomes possible to slash memory usage. This process has clear advantages regarding the overkill method of storing coordinates in typical data types such as an integer that usually has a 32-bit precision. In fact the use of integers might suffice if their precision is sufficient to depict a scene. Consider for example that an image with resolution 32×24 is acquired, rather than having a binary array of dimension 768, one could encode each line using an unsigned 32-bit precision integer in the exact same way as a binary array. By employing binary arrays it becomes possible to use the Hamming distance, which allows for efficient computation of array differences.

The procedure presented in Figure 6 is applied to an image of resolution $N \times M$, where N represents the number of pixels per row and M the number of rows. The encoding algorithm

takes as input a binary array I representing the scene which contains the classification of each pixel and produces a list containing for each row the respective encoding obtained. The algorithm runs in $O(kn)$ where k is the number of rows and n represents the amount of binary pixels per row.



(a) Posture Y.



(b) 32×24 version of Posture Y.

Figure 5: Two images depicting the transformation process from an image containing a posture (Figure 5(a)) to a segmented image with a reduced resolution (Figure 5(b)).

Algorithm *ENCODING-ALGORITHM* **(I)**
Input: $I = N \times M$ binary array containing the classification of each pixel
Output: S = a list containing, from bottom to top, exactly the encoding for each line

```
let encoding denote an auxiliary variable that represents a row encoding
let offset denote an auxiliary variable that keeps track of the offset of a pixel in the binary array
for each row r ∈ I
    do offset ←0
       encoding ←0
       for each pixel p ∈ r
           do if PAINTED(p)
                 then encoding ←encoding + SHIFT-LEFT(1, offset)
              offset ←offset + 1
       ADD(S, encoding)
    return S
```

Figure 6: Encoding Algorithm.

2.2.3 *Classification performance*

In order to evaluate the performance of the methods developed for hand posture recognition, namely the Convex Hull and Simple Pattern Recognition a total of 2023 images for five postures were evaluated. Of these, a smaller set of 120 images was randomly chosen to represent the training set and the remaining images were used to build the test set.

Regarding the choice of a classifier algorithm, one possible method consists of labeling an unknown feature vector x into the class of the individual sample closest to it. This is the *nearest-neighbor* rule (see [Shapiro and Stockman, 2001]). A better classification decision can be made by examining the nearest k feature vectors in the database. Our final choice, respectively the K-Nearest-neighbor classifier can be effective even when classes have complex structure in d-space feature vectors and when classes overlap. The algorithm uses only the existing training samples so no assumptions need to be made about models for the distribution of feature vectors in space.

In the end, the simple pattern recognition method successfully classified 84% of the test images whilst the convex hull only correctly identified 47% of the same test set.

2.3 Hidden Markov models applied to gesture recognition

At its core a gesture is nothing more than a set of postures, which can be characterized as a signal. A problem of fundamental interest in real-world processes producing observable outputs consists in characterizing signals in terms of signal models [Rabiner, 1989]. The general principles of hidden Markov models were published by Baum and his colleagues (presented in [Baum and T., 1966], [Baum and Egon, 1967], [Baum and Sell, 1968], [Baum et al., 1970] and also [Baum, 1972]). Rabiner's and Dugad's work on hidden Markov models (HMMs), respectively presented in [Rabiner, 1989] and [Dugad and Desai, 1996], provide a detailed analysis, focusing on a methodically review of the theoretical aspects of this type of statistical modeling.

HMMs represent stochastic processes that are adequate for dealing with the stochastic properties of gesture recognition [Yang and Xu, 1994]. For this gesture recognition system, HMMs representing gestures were employed. The models and respective parameters were built from training data, with each model featuring a number of states equal to the number of postures present in each gesture. Each HMM constructed can be evaluated against a given input set of postures, in order to assess the likelihood of the model generating that sequence. In order to determine which gesture was presented to the system all HMMs are evaluated, with the model presenting the highest probability being ultimately used for classification.

3. SYSTEM IMPLEMENTATION

Many embedded systems have substantially different design constraints than desktop computing applications. The combination of factors such as cost sensitivity, long life cycle, real-time and reliability requirements impose a different mindset that makes it difficult to adopt traditional computer design methodologies and tools to embedded systems. Thus, no single characterization applies to the diverse spectrum of embedded systems [Koopman, 1996].

In the case of this gesture recognition system, the development cycle used is illustrated in Figure 7 and described next. Typical design methodologies start with a requirement capture phase characterizing the functional and non-functional requirements of an application. This phase is usually followed by an application architecture defining the execution flow and requirements compliance responsibility associated to each functional block. The combination of requirements capture associated to the definition of an application architecture, expressed as Stage 1 in Figure 7, resulted in the definition of the following functional cores for this gesture recognition system:

1. **Skin Detection Phase**

   In charge of image acquisition and respective data processing in order to distinguish between foreground and background and thus enabling the determination of image pixels that could be skin-labeled. This initial stage also provides the desired input data to the Posture Classification Phase;

2. **Posture Classification Phase**

   In charge of posture database creation, processing and manipulation and whose main responsibility would be to classify a given input image provided by the Skin Detection Phase against a trained database. This intermediate stage also provides the desired input data to the Gesture Classification Phase;

3. **Gesture Classification Phase**

   In charge of gesture database creation, processing and manipulation, namely the acquisition and training process of the hidden Markov models representing gestures. Besides these responsibilities, the Gesture Classification Phase is also liable for determining to which gesture a given set of postures belongs to.

Once the core functional blocks of the application architecture were defined work proceeded as follows:

- For each functional core a proof of concept was performed using Java Standard Edition (J2SE) represented as Stage 2 of Figure 7. A standard computer application was developed featuring all functional cores. The application served as a simulator allowing for parameters specification and result observation. If the proof of concept did not meet the requirements defined for that functional block, then further refinement would be performed on it.

  Due to the different operational audience of J2ME, which targets resource constrained devices, not every class of the plentiful J2SE API exists in J2ME, this is particularly true regarding data structure availability and also features such as built-in iteration mechanisms. With these limitations in mind and in order to ensure successful migration between platforms, a set of strategic rules was devised, namely:

  - **Rule 1** - Only use API methods available in both J2SE and J2ME;
  - **Rule 2** - Substitute J2SE native data structures by a combination of purpose-built data structures, *Vector* and *Hashtable* classes;
  - **Rule 3** - Guarantee that all purpose-built data structures override the methods of the *Enumeration* interface;
  - **Rule 4** - Substitute all J2SE standard iteration mechanisms with calls to the *Enumeration* interface;
  - **Rule 5** - Exercise great care when allocating objects and take all possible steps to avoid holding references to objects longer than necessary, in order to allow the garbage collector to reclaim heap space as quickly as possible.

- Each time a functional core was deemed as having an adequate behavior and performance it was migrated to a J2ME Mobile Information Device Profile (MIDP) application, also known as MIDlet, represented as Stage 3 of Figure 7. The migration process required a careful application adaptation effort as there would be portions of code, such as database creation and experimental tests, that were not necessary for MIDlet execution. Once the migration process was concluded the MIDlet was tested on Sun's Java Wireless Toolkit (WTK) 2.5.2 emulator in order to check for proper compliance within emulator specifications. Besides enabling specification of memory restrictions, Sun's WTK also provides a series of tools in order to profile memory usage and CPU performance, thus enabling bottleneck detection. If the tests performed on the MIDlet revealed the existence of performance hampering bottlenecks, optimization techniques would be applied to those portions of the code and further MIDlet reassessment would be made;

- Once the MIDlet achieved the desired behavior, the application would be loaded into smartphones and tested, expressed as Stage 4 of Figure 7. At this stage of development, the main test concerns regarded performance issues such as algorithm execution, memory issues such as ensuring that the application did not run out of memory and also fine tuning parameters such as camera resolution in order to obtain the best balance between speed and quality;

- If at any given time it was concluded that an implementation strategy was not suited due to performance issues, further refinement would be made in the previous stages.

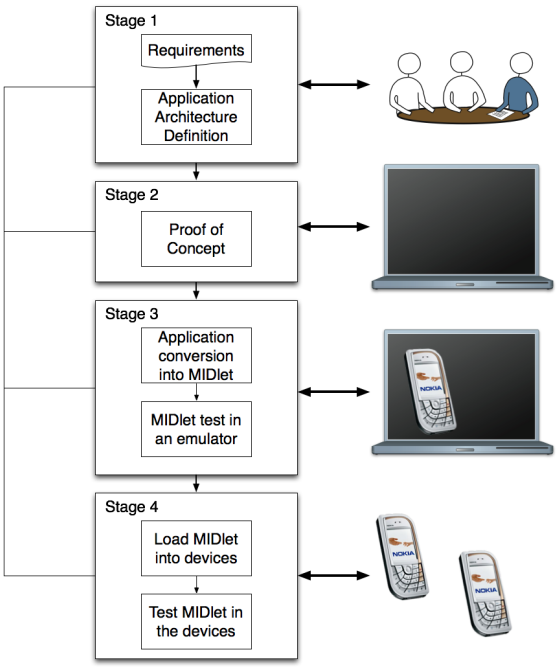


Figure 7: Development cycle characterization.

4. EXPERIMENTAL RESULTS

In the case of this work, besides the evident performance issues regarding how many correct results were achieved by this gesture recognition system it is also crucial to consider its core execution platform, i.e. smartphones. Keep in mind that one of the main objectives of this work is to assess current smartphone performance, considering not only computational power but also camera wise.

4.1 Smartphone Performance

J2ME grants access to a phone's camera through the optional library Mobile Media API (MMAPI) (see [Sun, 2008]) which provides audio, video and other multimedia support to resource constrained devices. MMAPI enables Java developers to gain access to native multimedia services available on a given device thus enabling the development of multimedia applications from different phone manufacturers.

Applications such as this gesture recognition system whose main focus is on pattern recognition rely heavily on image acquisition and processing. If an image processing application is to be successful, our device must be able to execute heavy algorithms as well as acquire images at a significant acquisition rate. Factors like image processing and camera performance in J2ME should be properly analyzed has they carry possible implications for the application [Tierno and Campo, 2005].

Two smartphones were employed for system testing, namely Nokia's N80 and N95 models, which feature respectively 3 and 5 megapixels cameras. Nokia's N95 model boasts a Texas Instrument's OMAP2420 processor whilst the N80 model features an OMAP1710. The OMAP2420 features an ARM1136 processor core clocked at 330 Mhz whilst the N80's OMAP1710 features an ARM926TEJ clocked at 220 Mhz. For further information please refer to [TI, 2008b] and [TI, 2008a].

4.1.1 Camera performance

MMAPI enables video snapshot acquisition, outputing immutable instances of the *javax.microedition.lcdui.Image* class. MMAPI also allows one to obtain the RGB pixel values from an image in order to proceed with data processing. The combination of these three procedures corresponds to the steps required in order to execute some useful computation over a given image and as such the combined total time can be interpreted as representing the acquisition time [Tierno and Campo, 2005]. Figure 8 illustrates the average acquisition times obtained for Nokia's N80 and N95 models using MMAPI. For each resolution mode ten image processing iterations were performed (contemplating the three procedures mentioned above). A total of 120 tests were performed.

As it is possible to see both models present relatively high acquisition times even for such low resolutions as 160×120 making it impossible to meet real-time requirements. Also noteworthy is the fact that both models present an acquisition time drop for the 640×480 resolution which seems to provide consistency to the fact that this resolution represents the standard operating mode for both cameras, with the remaining resolutions being obtained as rescales [Tierno and Campo, 2005].

It is also important to draw attention to the fact that from a smoothness operational perspective the N95 MMAPI implementation, regarding camera functionality, operated in a consistent manner throughout system development. The same could not be said for the N80 MMAPI implementation which systematically and randomly raised exceptions, grinding system development to a halt.

Figure 8: Image acquisition times regarding image resolution.

4.1.2 Low-level operations performance

Regarding J2ME processing performance in the N80 and N95 we measured the processor's execution time for basic low-level operations to determine the overall speed and to identify the fastest and slowest operations. Each operation was executed in a loop of 10.000.000 iterations and the total execution time of loop was measured. In order to counter the effects of loop specific instructions, such as increments and comparisons, the total execution time of an empty loop with the same number of iterations was measured. The duration of each operation corresponds to an average between the time of first loop minus the empty loop, over the number of iterations. An average measure was chosen because the application was executed concurrently with other processes running on the smartphone's operating system. This procedure was then applied to measure how long it took to:

- access an array
- increment an integer variable;
- add two integer variables;
- use bit shift operators to multiply and divide integers by a power of 2;
- compare two integer variables (equal, less or equal, less).

The results presented in Figure 9 reveal the times achieved for each operation. In both smartphones models, division is the slowest operation followed respectively by array extractions, compare operators and finally arithmetic operations. Although the N80 and N95 feature different base ARM processors and one expected to see better performance from the latter it was still surprising to check that regarding the less or equal operation the N95 was more than three times slower than the N80. Regarding the remaining operations it is possible to check a clear performance superiority from the N95.

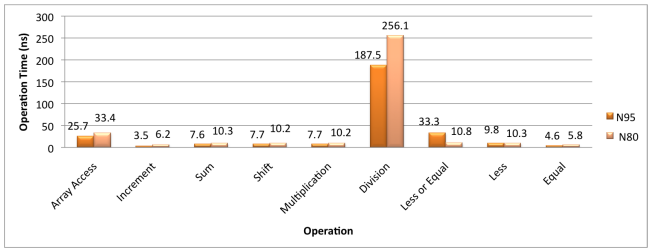


Figure 9: Operation times regarding low-level arithmetic operations.

4.1.3 System execution time performance

In order to assess the impact of each smartphone computing power in the final gesture recognition system a total of 50 hand posture classifications per smartphone were conducted. The hand posture classification was chosen as it conveys information regarding the execution times associated with the posture labeling of each input image in an acquisition time-wise independent fashion. This situations contrasts with the system's attempts at determining to which gesture a given set of postures belongs. In order to do so it is necessary to take into account the total time between acquiring an initial image and calculating the most probable gesture. By analyzing the time spent processing each image it becomes possible to exclude the acquisition time delay associated to each camera and thus provide for a comparable measure.

The final system was deployed onto both smartphone models. Table 1 illustrates the average execution times per hand posture, as well as respective frequencies allowed for those values, obtained for Nokia's N80 and N95 models.

	Nokia N80	Nokia N95
Average Execution Time (ms)	4935	1036
Frequency (Hz)	0.202	0.965

Table 1: Average execution times per hand posture and frequencies.

The results obtained demonstrate a clear superiority, regarding execution times, for the N95 smartphone. In average, Nokia's N80 model was over four times slower when comparing with the N95 results. The N95 smartphone results allow for an input frequency of approximately a hand posture per second.

If one considers a real-world application of a gesture recognition system for smartphones with interacting users this value, of a hand posture per second, seems like a reasonable one to expect. On the other hand the N80 frequency of 0.202 Hz is considerably lower and would not allow for a convincing user experience. The combination of MMAPI stability alongside better performance results were deciding factors that contributed to our final choice of using Nokia's N95 model as the smartphone component for final system testing.

4.2 Profiles

Sun's WTK, which was used during system development, incorporates a profiler, enabling behavior measurement during program execution, focusing on the frequency and duration of function calls. It outputs a statistical summary of the events observed. It should be noted that different measurements along a timeline, will result in different profile observations, as the contribution of different functions outweigh those of others.

Figures 10 and 11 illustrate the results obtained during the execution of this gesture recognition system, after one gesture was introduced and then again after ten gestures were presented to the system. As was to be expected intense computer vision algorithms claim the greatest share of the execution time distribution, namely the Skin Detection and filter components. Both these components start by representing over 75.34% and end up with 91.34%. This situation was to be expected as both components represent pixel intensive operations that besides analyzing every pixel of an input image also conduct a neighborhood survey over filtered images. Clearly, in order to tackle the system's execution time it is necessary to address the performance issues surrounding these two components and to consider some optimization techniques.

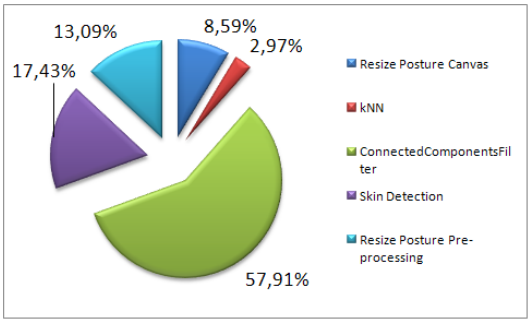


Figure 10: Execution (1 Gesture).

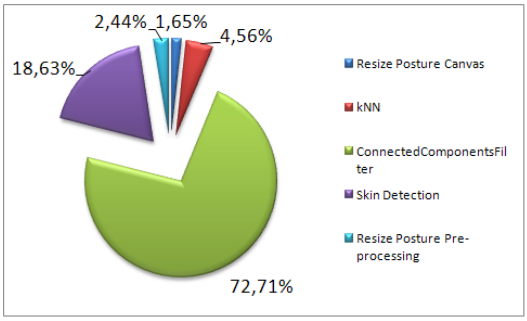


Figure 11: Execution (10 Gestures).

4.3 System performance

4.3.1 Experimental setup

In order to test this gesture recognition system we proceeded with the experimental setup demonstrated by Figure 12, which illustrates the following factors:

- Figure 12(a) - Due to a lack of scaling mechanisms within this gesture recognition system, special care was taken with the distance between smartphone and hand

gestures. A simple measuring tape was employed in order to ensure proper distance compliance between experimental setup and the images employed for training set creation regarding hand postures;

- Figure 12(c) - The system was deployed onto Nokia's N95 smartphone and gestures were presented to the system according to a previous established distance. During the duration of the tests the smartphone remained at a fixed position. Each time a posture was processed an audio signal notified the user that a new posture could be introduced to the system;
- Figure 12(b) - Illustrates the capture view associated with the N95's built-in camera. The image depicts an office room under natural lighting conditions and with an abundance of background objects. These factors, alongside not being a simple black background, allow the image to be classified as a complex background.

(a) Experimental Setup. (b) Test Example. (c) Capture View.

Figure 12: Experimental Setup.

4.3.2 *Results*

Three gestures classes were used for system testing and labeled respectively as Gesture 1, Gesture 2 and Gesture 3. For each gesture class defined, 30 gestures instances per class were presented to the system. Given the tree core gesture classes defined and the 30 instances per class it becomes possible to deduce that a total amount of 90 gestures were employed during system testing. Figure 13 illustrates the precisions obtained for the three gestures defined, the following items should be pointed out:

- Gesture class three has a slight advantage over the remaining classes. This was to be expected as the HMM model depicting this gesture presents a higher observation probability, respectively 25%, over the remaining HMMs, respectively 6.25% and 12.5% for Gesture 1 and Gesture 2. In this case, higher probability translates directly into a more likely class three classification, despite of possible posture misclassifications. Accordingly, most gesture misclassifications verified were due to this higher probability, with the system erroneously classifying gestures as belonging to class three.
- The misclassified gestures were due to errors introduced by significant posture variations that had a direct impact on hand posture classification. Every time the system correctly classified individual hand postures the correct gesture would be returned. Otherwise the system would attempt to retrieve the most likely gesture.
- Correct gesture classifications strongly depended on correct individual posture classification. In this case, the number of correct posture hits is in accordance with the results presented regarding posture classification.

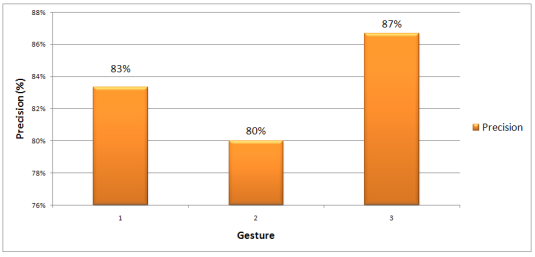


Figure 13: Final system precision results.

Considering the precision obtained for the three gestures it becomes possible to obtain an average precision value of 83.3%.

5. RELATED WORK

This average precision value behaves moderately when compared with other gesture recognition systems, specially if one consider the far simpler techniques employed in this work, namely:

- [Jota et al., 2006] - Obtains average recognitions rates around 93% using two separate techniques. The first technique recognizes bare-hands using its outer contours. The second approach is based on color marks present on each fingertip that allow for hand tracking and posture recognition;
- [Liang and Ouhyoung, 1998] - A large vocabulary sign language interpreter using HMMs was developed using a DataGlove to extract a set of features such as posture, position, orientation and motion. The average recognition rate sustained by the experimental results is 80.4%;
- [Triesch and von der Malsburg, 2002] - A system for classification of hand postures against complex backgrounds was developed employing elastic graph matching. The system obtains an average 86.2% correct classification.

6. CONCLUSIONS

The tests performed to assess smartphone performance focused on three dimensions, namely: camera, low-level arithmetic operations and system execution performance. Of the tests performed the most penalizing factor was camera performance with J2ME which exhibited extremely low capture times. The remaining tests, which basically attempted to measure each smartphone processing power, demonstrated that the latest smartphone generation is able to execute computationally expensive applications. Keep in mind that this gesture recognition system makes intensive use of image processing, posture and gesture databases, pattern recognition, graph algorithms and also probabilistic models.

7. REFERENCES

Baum, L. (1972). An inequality and associated maximization technique in stastical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8.

Baum, L. and Egon, J. (1967). An inequality with applications to statistical estimation for probabilistic functions of a markov process and to a model for ecology. *Bull. Amer. Meteorol*, 73:360–363.

Baum, L., Petrie, T., G., S., and Weiss, N. (1970). A maximization technique ocurring in the stastical analysis of probabilistic functions of markov chains. *Ann. Math. Stat*, 41:164–171.

Baum, L. and Sell, G. (1968). Growth functions for transformations on manifolds. *Pac. J.Math.*, 27:211–227.

Baum, L. and T., P. (1966). Statistical inference for probabilistic functions of finite state markov chains. *Ann. Math. Stat*, 37:1554–1563.

Bonato, V., Sanches, A. K., M.M., F., Cardoso, J. M., Simoes, E., and Marques, E. (2005). A real time gesture recognition system for mobile robots. In review.

Brand, J. and Mason, J. (2000). A comparative assessment of three approaches to pixel-level human skin-detection. *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, 1:1056–1059 vol.1.

Cormen, T., Leiserson, C., Rivest, R., and C., S. (2003). *Introduction to Algorithms - Second Edition*. The MIT Press.

Davis, J. and Shah, M. (Apr 1994). Visual gesture recognition. *Vision, Image and Signal Processing, IEE Proceedings* -, 141(2):101–106.

Dugad, R. and Desai, U. (1996). A tutorial on hidden markov models. Technical report, Indian Institute of Technology.

Gonzalez, R. C. and Woods, R. (1992). *Digital Image Processing*. Addison-Wesley Company.

Graham, R. (1972). An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133.

Jones, M. J. and Rehg, J. M. (2002). Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46:81–96.

Jota, R., Ferreira, A., Cerejo, M., Santos, J., Fonseca, M. J., and Jorge, J. A. (2006). Recognizing hand gestures with cali. *EUROGRAPHICS*, 1981:1–7.

Koopman, P. (1996). Embedded system design issues (the rest of the story). *Computer Design: VLSI in Computers and Processors, 1996. ICCD '96. Proceedings., 1996 IEEE International Conference on*, pages 310–317.

Kovac, J., Peer, P., and Solina, F. (2003). Human skin color clustering for face detection. *EUROCON 2003. Computer as a Tool. The IEEE Region 8*, 2:144–148 vol.2.

Liang, R.-H. and Ouhyoung, M. (1998). A real-time continuous gesture recognition system for sign language. *Proceedings. Third IEEE International Conference on*, 1:558–567.

Lin, X. and Chen, S. (9-11 Apr 1991). Color image segmentation using modified hsi system for road following. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1998–2003 vol.3.

Mckenna, S., Gong, S., and Raja (1998). Modelling facial colour and identity with gaussian mixtures. *Pattern Recognition 31*, 12:1883–1892.

Poynton, C. A. (1995). Frequently asked questions about colour.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286.

Shapiro, L. G. and Stockman, G. C. (2001). *Computer Vision*. Prentice Hall.

Sun (2008). Mobile media api (mmapi); jsr 135. http://java.sun.com/products/mmapi/, Last visited in July 2008.

TI (2008a). High-performance: Omap1710. http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11991&contentId=4670, Last visited in July 2008 ,.

TI (2008b). High-performance: Omap2420. http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11990&contentId=4671, Last visited in July 2008.

Tierno, J. and Campo, C. (2005). Smart camera phones: Limits and applications. *Pervasive Computing, IEEE*, 4:84–87.

Triesch, J. and von der Malsburg, C. (2002). Robust classification of hand postures against complex backgrounds. *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 170–175.

Vezhnevets, V., Sazonov, V., and Andreeva, A. (2003). A survey on pixel-based skin color detection techniques.

Yamaguchi, H. (Nov 1984). Efficient encoding of colored pictures in r, g, b components. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 32(11):1201–1209.

Yang, J. and Xu, Y. (1994). Hidden markov model for gesture recognition. Technical report, Carnegie Mellon, Robotics Institute.

Zarit, B., Super, B., and Quek, F. (1999). Comparison of five color models in skin pixel classification. *Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 1999. Proceedings. International Workshop on*, pages 58–63.

Zhang, C. and Wang, P. (2000). A new method of color image segmentation based on intensity and hue clustering. *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, 3:613–616.