Online Character Recognition

ABSTRACT

Background – Character Recognition is the process of identifying hand-drawn characters and converting these into an electronic format. While Offline Character Recognition – using scanned images – has its uses in post code recognition and bill payments, consumer applications of the technique seem limited based upon its current accuracy rates. Conversely, applications of Online Character Recognition – using the movements of a pen – have increased significantly in recent years because of the increased popularity of touch screen devices.
Aims – The main aim of this project is to measure the performance of online character recognition as a method of character input on a touch screen device. This will then be compared to traditional methods, such as the use of an on-screen QWERTY keyboard, to evaluate its suitability in different scenarios.
Method – Initially an existing online character recognition algorithm will be implemented, and potentially improved. This algorithm will then be incorporated into an Android application, allowing use of the algorithm on an Android device. User testing will be performed to compare this application to a traditional on-screen keyboard.
Proposed Solution – An Android application that uses online character recognition to identify characters that have been drawn on the device screen. This will allow users to input characters into the device by drawing them.

Keywords – online character recognition; handwriting recognition; android keyboard; keyboard application.

I. INTRODUCTION

Machines represent characters with binary sequences of fixed length, each defined in a character set. This allows the machine to uniquely distinguish different characters, with each sequence representing a single character, and each character being represented by exactly one sequence. Standard machine input methods convert physical key presses, typically on a keyboard, to characters represented in machine code, allowing them to be used in this format.

While traditional machine input methods, such as a QWERTY keyboard, are very successful with established users, they may not be the most intuitive methods for new computer users. For example, elderly people who are using a computer for the first time may have never used anything like a QWERTY keyboard before, and so will be required to learn how they work. This extra learning may actually deter them from starting to use computers in the first place. However, these elderly people will almost certainly be familiar with hand writing words, and will likely do this on a daily basis. Therefore it would be appropriate to allow them to draw characters, whether this is with a special pen or on a touchscreen, as a method of computer input.

There are two types of algorithm that will allow them to do this: Offline Character Recognition and Online Character Recognition. Offline recognition algorithms take a standard image file, usually in the form of a scanned document, and use this image to determine the text that the image contains. This requires a large amount of processing power, because of the image manipulation involved, but can be recorded with a simple imaging device such as a scanner. Online recognition gathers more data than offline does – such as how many strokes were used and the direction of each stroke – and stores the drawing as a sequence of points to remove the need for image manipulation. This allows features to be extracted quickly and easily in order to help determine the character drawn, but does require that all the relevant data is recorded by the input device.

As (Plamondon & Srihari, 2000) state, handwriting is a skill that is personal to individuals.

While a machine represents each character with a single chunk of digital code, each character can take an infinite number of handwritten forms, and no two people will draw characters in exactly the same way. For example, the character 'A' can be drawn with a curved or pointy top, with lines overlapping each other, or even with the "hole" in the middle missing, as shown in Figure 1. In addition to this, human written characters are very rarely composed of perfect lines and curves, like those that are seen in computerised fonts, and are often inconsistent in shape and size. Any process allowing hand drawn characters to be converted to computerised ones must be able to reliably recognise these variations and inconsistencies in order to determine the exact character that has been drawn.

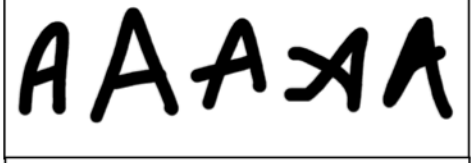

Figure 1: Demonstrating 5 of the ways that an 'A' can be hand drawn. Images taken from the Chars74K dataset (University of Surrey, 2012).

One of the main problems for online character recognition has been the lack of hardware available to record the input. Typically, computers are equipped with a mouse and keyboard for input, and while a mouse may have the capability to draw characters on a screen, it is not very practical. This meant that users would have to purchase extra hardware, in the form of special pens, to easily draw virtual images on screen. However, development of touch screens in the early 2000's led to them increasing in popularity, in particular as part of smart phones. While in Q1 2005 smartphones only accounted for 4% of mobile phone sales in the UK, this had increased to 48% by Q1 2011 (Ofcom, 2011), with over 1 billion smartphones in use around the world by Q3 2012 (Strategy Analytics, 2012). This increase means that more and more consumers are now equipped with a device capable of recording the input for online character recognition algorithms, and so are only in need to an application to perform it.

Despite this drastic increase in sales of touch screen devices, a keyboard has remained the primary method of character input. Early smartphones had hardware keyboards attached to them, while later devices replaced this with a virtual keyboard shown on the screen. The purpose of this project is to investigate whether the keyboard is still the most appropriate form of character input for this relatively new technology, and specifically whether the touch screens ability to record finger movements can be exploited to allow character input through drawing.

Recent developments in mobile operating systems, such as Google's Android 2.3 (Google Inc., 2010) or Apple's iOS 5 (Apple Inc., 2012), have led to the introduction of voice recognition as a form of character input. This feature allows a user to speak into the device, which will then process this speech and attempt to recreate it as electronic text. This technique has to deal with it's own problems, such as differentiating between the many different accents and languages that users may speak in, but also adds a range of benefits, such as being able to perform functions on a device without looking at the screen. Whilst voice recognition should not be ignored as an alternative to the QWERTY keyboard, it's potential benefits will not be investigated as part of this project. However, a third party implementation may be used during the evaluation phase as a means of comparing the performance of character recognition and voice recognition as forms of character input.

Project Aim

The aim of this project is to evaluate the suitability and practicality of using online character recognition methods as a form of character input into a touch screen device. This will be achieved by creating a prototype application, described in the "Deliverables" section below, in order to analyse the performance of the algorithm. The performance will be measured in terms of recognition accuracy, writing speed and user enjoyment – amongst other things. This data will then be compared to equivalent measurements for other character input methods, initially an on-screen QWERTY keyboard, to assess the potential for a fully functional character recognition keyboard. This keyboard will not be developed as part of the project.

Deliverables

The project will have the following deliverables, culminating in a final prototype and a report to evaluate its suitability:

1. **Android Application for Input**: This will be a simple Android application, designed to record character input data for later use in a character recognition algorithm. The application will prompt users to draw a particular character on a canvas, allow them to do this, and then record information about this drawing for later use. It is important that this application records all information that will be needed by the character recognition algorithm, and would be useful if it recorded information to assist testing and training.
2. **Desktop Application for Recognition**: This will be a Java application that will read character drawing files, created by the Android application above, and redraw them on-screen. It will then be further developed to incorporate my implementation of an online character recognition algorithm, allowing this to be run on the drawings. The program will also maintain and display statistics about the performance of the algorithm.
3. **Android Application for Recognition**: This will be another Android application, which will allow users to draw characters on a canvas. It will then run the recognition algorithm (from Deliverable 2) on each drawing, and input the recognised character into a text box on screen. While this will not be a fully functional Android keyboard, and so will not be available for character input into any application, it will simulate this functionality in a single application.
4. **Project Report**: Finally I will deliver a project report, which will contain details of, and results from, the evaluation of the prototype. It will also provide conclusions to the research question of whether this could be suitable as a fully functional input method. The evaluation will include measuring the performance of the prototype as a method of character input, and then comparing this to other character input methods, such as a traditional keyboard.

II. DESIGN

A. Requirements

Table 1 shows the functional requirements that have been identified for the project. FR-01 and FR-02 describe the required functionality of the initial Android application (Deliverable 1), which will be used to record character data for use in the algorithm. This will then be extended with FR-09, FR-10, FR-11 and FR-12 to form the final prototype application (Deliverable 3), simulating the effects of a fully functional keyboard.

FR-03 describes the desktop application that will be created to allow users to view characters that have previously been drawn using the Android application. It will then be extended with FR-05, FR-06 and FR-07 to enable it to make use of an Online Character Recognition algorithm, and provide information to help test and train the algorithm.

Table 1: List of Functional Requirements

#	Requirement	Priority
FR-01	The Android application (application A) should allow a user to draw a character on screen, by adding lines to the drawing based upon where the user touches it.	High
FR-02	Upon completion of a drawing, application A should save information about this drawing in an online format, so that it can be recreated exactly as it was drawn.	High
FR-03	A desktop Java application (application B) should be able to read the information in this online format, and redraw it on the screen.	High

FR-04	An Online Character Recognition algorithm should be implemented to receive drawings in this online format, and then output the character that the drawing is believed to represent.	High
FR-05	The algorithm described in FR-04 should be incorporated into application B to allow character recognition to be performed on the drawing shown on screen. Application B should then display the output of the algorithm on screen.	High
FR-06	Application B should store statistics about the performance of the algorithm, including how accurate the recognition is and the average time for each recognition. This will allow us to clearly see the effects of any changes that are made to the algorithm.	Medium
FR-07	Application B should also record the identified signature of any drawing that fails the recognition (that is, is not identified as any character, or is identified as the wrong character). This will help with training the algorithm.	Medium
FR-08	The algorithm described in FR-04 should run correctly on both Windows and Android operating systems.	Medium
FR-09	The algorithm described in FR-04 should be incorporated into application A, to allow recognition to be started on a drawing as soon as it is complete. Application A should then enter the output of the algorithm into a text box.	Medium
FR-10	Application A should also include a method for adding special characters to the text box (aka those not included in the algorithm), such as space, full stop and delete.	Low
FR-11	Application A should allow text to be modified at any position in the text box, through the use of a cursor.	Low
FR-12	The algorithm could be modified to take into account the context of the text box when deciding upon a character. For example, it could use the previous letter to help determine whether the character should be uppercase or lowercase, or it could attempt to match a whole word to a dictionary in ambiguous cases.	Low

Table 2 shows the non-functional requirements that have been identified for the project. They describe performance related attributes that are required to ensure that the final application for recognition is usable and effective.

Table 2: List of Non-Functional Requirements

#	**Requirement**	**Priority**
NFR-01	Application A must record sufficient data points for the algorithm to run effectively.	High
NFR-02	The algorithm should never fail to complete, assuming it is provided with valid input data.	High
NFR-03	On average, the algorithm should provide an output within 40ms on a HTC Desire mobile device.	Medium
NFR-04	The algorithm should output a result 100% of the time, regardless of how unsure it is.	Low
NFR-05	The keyboard section of Application A should not take up more than 50% of the screen, leaving enough space to display the content that the user is entering characters into (in this case a simple text box).	Low

B. Algorithm Design

Choice of Algorithm

The online character recognition algorithm that will be initially implemented for the project will be that proposed by (Agarwal & Kumar, 2005), which expands upon the plex structure solution by (Feder, 1971).

Feder suggests that any particular character can be represented as a combination of symbols, which Feder refers to as "*n* attaching point entities" (NAPEs). Each of these symbols can have up to *n* attaching points, at which it can be connected to any of the other symbols in the structure. Similarly, a drawing of a character can be represented as a combination of strokes, with each stroke having *n* attaching points to other strokes in the drawing. Once a drawing is converted into this form, it can be compared to a model database to determine the character that has been drawn.

Agarwal & Kumar build upon this algorithm by suggesting that each symbol in a character must be one of 5 basic primitives: line, point, loop, up (clockwise curve) or down (anti-clockwise curve). Each of these primitives must also have a direction, which is discretised into the 8 cardinal and ordinal directions (see Figure 3, next page). They suggest that each primitive should only have 2 attaching points, one at the beginning and one at the end, and that a primitive can only be attached to those immediately before or after it in the structure. This simplifies that process of calculating these connections. Agarwal & Kumar go on to provide other enhancements, such as allowing primitives to be rotated if an initial match is not found, that are described later in this document.

With these modifications, Agarwal & Kumar claim that the algorithm provides recognition rates of 98.3% for digits and 99.2% for uppercase letters. The representation of characters also means that drawings are size invariant, and allowing the structures to be rotated also makes it rotation invariant. The algorithm also claims to provide fast recognition speeds under low memory requirements, something that is especially important when planning to use it on mobile devices.

Algorithm Details

Figure 2 shows the process that a drawing will go through during character recognition. This includes the 7 different recognition stages, as well as nodes for input and output. (Agarwal & Kumar, 2005) describes what each stage of the algorithm should achieve, but not how it should achieve it, and so this has been described below.

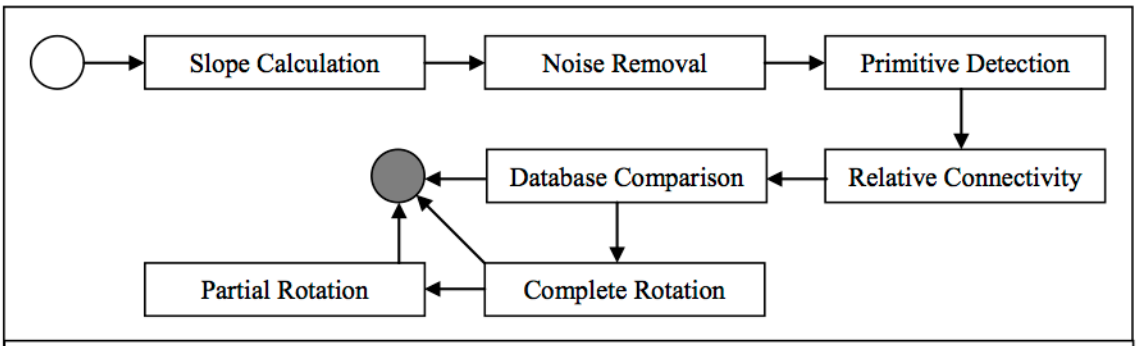

Figure 2: A process diagram to show the different stages of the recognition algorithm

Input: The input to the algorithm will be a drawing of a character. This drawing will be represented by a series of graph points, recorded in the order that they were drawn. There must also be a way for the algorithm to differentiate between the different strokes in the drawing, and this will be done by having a "special" point to represent the start of a new stroke. Other desirable information would be the size of the canvas that the character was drawn on, so that values can be standardised, and the actual character that the drawing is meant to represent, for testing and training purposes.

Slope Calculation: Each stroke in the drawing will be converted from a list of points to a list of slopes, with each slope representing the angle between consecutive points. This will be done by iterating through each pair of points, and using their positions and order to calculate the direction that the pen was travelling when they were drawn. The direction will be discretised into one of the 8 ordinal and cardinal directions, before adding it to the list of slopes and moving on to the next pair of points. The final list of slopes for each stroke will be 1 item shorter than the list of points for that stroke.

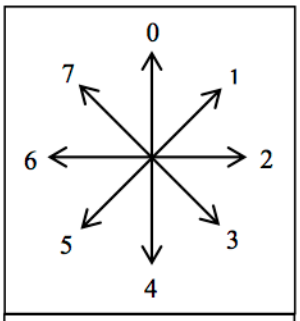

Figure 3: The 8 directions that slopes will be discretised into

Noise Removal: As discussed previously, human drawings of characters rarely result in perfect lines and curves being drawn. This causes the real slopes to be interpolated with much smaller slopes, which appear when the finger has fluctuated slightly during the drawing motion. These smaller slopes are not part of the overall structure of the character, and so need to be removed. This will be done by specifying a noise threshold. The list of slopes produced in the previous stage will be iterated through, looking for slopes with a length less than this threshold (where the length of a slope is the number of consecutive appearances of the same direction in the list). If such a slope is found then it is removed from the list, and the relevant points are also removed from the stroke.

Primitive Detection: This stage will convert a list of slopes in a stroke into a list of primitives that they represent, and then combine all strokes to provide a complete list of primitives that a drawing contains. This will be done in several steps:

1. Firstly, if the stroke is short enough, then it will be regarded as a "point".
2. If the stroke is not a "point", then it will be split into segments. A segment will be either: a large slope of a single direction, and so a "line"; several slopes where each slope is one direction clockwise from the previous one, and so an "up"; or several slopes where each slope is one direction anti-clockwise from the previous one, and so a "down".
3. For each curve (both "up" and "down"), the distance between the first and last point in the curve will be calculated. If this distance is short enough, then it will instead be regarded as a "loop".
4. Finally we will calculate the direction of each primitive. The direction of a line will be the direction of the slope that forms the line. The direction of a curve will be the slope in the middle of that curve, and so the direction of a tangent to the curve. Points and loops will have no direction.

This will result in an ordered list of primitives for the drawing.

Relative Connectivity: Here we will determine any connections at the attaching points of each pair of consecutive primitives, *s* and *t*, in the primitives list. The only attaching points relevant to this algorithm are the points at the start and end of each primitive, providing 4 possible points of connection:

- The first point of s to the first point of t
- The first point of s to the last point of t
- The last point of s to the first point of t
- The last point of s to the last point of t

The distance between the points of each combination will be calculated, and if this is lower than a threshold then it will be regarded as a connection. The connections will be recorded as 4 bits, one for each possible connection point, where a 1 represents a connection and a 0 represents no connection. These 4 bits will then be inserted in the correct position in the list of primitives, so that all primitives are separated by 4 bits that show how they are connected.

Database Comparison: By this stage we will have a complete signature (sequence of primitives and connections) to represent the drawing, and so this can be compared to a model database to see if there is a match. This will be done with a simple database query, using a standardised String representing the signature as a search term. As shown in Figure 4, this signature string will be the primary key of the database table, and so each signature can represent only one character. This means that the search will return at most one result. If a result is found, then this will be used as the output. Otherwise, Complete Rotation will be performed.

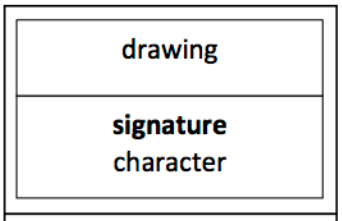

Figure 4: An Entity Relationship Diagram showing the structure of the database.

Complete Rotation: The algorithm will only proceed to this stage if the Database Comparison failed to return a result. My implementation will allow each drawing to be rotated one position clockwise and one position anticlockwise, and so will only assist in detecting characters drawn at a slight angle, but not those that have been drawn upside down or on their side. I believe this is appropriate, as in the Android application the canvas should automatically rotate based on the phones orientation, meaning the drawings should always be the right way up. The rotation will be performed on the signature of the drawing, by simply altering the direction of every primitive it contains. A new database comparison can then be made for each new signature. Like with the original database comparison, if a result is found here, then it will be returned as the output.

Partial Rotation: Once again, the algorithm will only perform this stage if the previous stage failed to find a result. Partial Rotation will attempt to detect characters where only some, but not all, of the primitives have been drawn at slightly the wrong angle. It will do this by recursively checking all possible combinations of primitives being un-rotated, rotated once clockwise, and rotated once anticlockwise. Each combination will be compared to the database, producing a list of possible characters for the drawing. The algorithm will then attempt to find the most common character in the list, or the character that required the least primitives to be rotated, to be returned as the output.

Output: An output can be produced when any of the previous 3 stages find a character that the drawing being processed represents. The output will take the form of a single character, and so these stages must provide a way to select a value if multiple candidates are available.

Proposed Extensions

In addition to the stages described above, I have proposed a number of additional stages that will hopefully improve the performance of the algorithm. They are not only designed to increase recognition rates, but also to remove redundant operations that may occur and reduce the time taken to produce a result.

Each of these modifications will be implemented and tested, with results collected on the impact they have on the algorithms performance. These results will then be used to help decide whether they should be included in the final application.

Merge: This function is designed to fix errors that occur when a single primitive is recognised as a combination of several smaller primitives. These errors are likely to occur during Primitive Detection, but will not be recognisable until after the connections between primitives have been identified. Therefore, Merge would take place after the Relative Connectivity stage of the algorithm.

Merge will look for specific combinations of primitives, and "merge" them into a single primitive. These combinations will have been previously identified as situations where the problem above occurs. It will add all the points from the merged primitives into a newly formed primitive, and then replace the old primitives with the new one. The function will then correct the connections at either side of the primitive by partially rerunning the relative connectivity stage.

For example, a single line may be identified as two separate lines if there is a large piece of noise in the middle of it. This function will fix this by searching for consecutive primitives, where both primitives are lines; both have the same direction; and where the end of the first line is connected to the start of the second. These lines will be merged into the single line, which will contain all the points from the two original lines, and replace them in the drawing signature.

Other combinations that the function will attempt to fix include a loop being identified as two curves, and a single line being drawn several times by the user.

Trim: The trim function is designed to improve the recognition speed of the algorithm, by removing primitives from signatures that are too large.

For the English alphabet, the largest signature of a valid character contains 4 primitives (for the characters 'E', 'M' and 'W'). This means that if the signature of a drawing has more than 4 primitives a result will never be found, regardless how much rotation takes place. This makes every operation in the Database Comparison, Complete Rotation and Partial Rotation stages completely redundant. Therefore it would be appropriate to reduce the size of these signatures to at least give them a chance of finding a result.

Trim will achieve this by repeatedly removing the smallest primitive (that is, the primitive with the least number of points) from a signature until its size is at most 4. This means that Trim will perform no operations on signatures that are initially a valid length, leaving them unaffected.

Whilst Trim would have all the information needed to run after the Primitive Detection stage, I believe it is important that the Merge function runs before it in order to prevent primitives being removed unnecessarily. Therefore the Trim function will run immediately before the Database Comparison stage.

Interpolate: In previous stages of the algorithm, the length of primitives has been used to assist in making decisions, and this length has always been determined by the number of points a primitive contains. However, there is no guarantee that these values are proportionate, as the number of points will actually be determined by the speed at which the pen (or finger) moves, and not the distance that the pen has travelled.

Interpolate aims to fix this, by adding new points to the drawing in between existing ones, with the number added dependant on the length between the original points. Once this has been performed, the distance between consecutive points in the drawing will always be roughly 1 unit, and so the number of points will be a more accurate representation of length.

Interpolate will form part of the Slope Calculation stage of the algorithm. Each new point will be added on the line between existing ones, and so slopes can simply be duplicated rather than having to be recalculated. This means that there should be little time spent calculating the points, but also means that they give no extra information on shape, as would be the case with a true interpolation function.

Type Deformation: This technique is suggested by (Chan & Yeung, 1998). It allows primitive types to be changed to another similar type, in an attempt to find an approximate match. This is a similar idea to allowing the directions of primitives to be changed slighty, as is done in the Complete and Partial Rotation stages.

For example, this process would allow a primitive that has been detected as a "line" to be transformed into either an "up" or a "down", creating 2 new signatures to compare the database. However, an "up" would only be able to transform into a "line", as a "down" is too different to it.

As with Partial Rotation, this could be implemented recursively to create around 3^n new combinations to compare to the database. This would result in many more database comparisons, and so the time taken to complete the algorithm would also increase.

This stage will take place at the end of the algorithm, and only execute if an output has not yet been found.

Dictionary: I will also be looking into the possibility of using a dictionary based system to assist the algorithm in deciding which character has been drawn. It will do this using the context of the text box, initially using this to determine whether the character should be upper case or lower case, based on the character preceding it. This will later be adapted to look at the word the character is going to be added to, and comparing this to a dictionary. The algorithm would then only allow characters to be recognised if they are going to form part of a valid word, reducing the size of the character set and so increasing accuracy rates.

C. Architecture

The programming language that the system will be developed in will be Java. The main reason behind this is to maintain compatibility with the Android operating system, as all applications that run on Android must be developed in Java. The rest of the system will also be developed in Java to maintain consistency, and also allow the component structure described below. As a result of this, the Desktop Java Application will not be constrained to running on a single operating system, and will instead be runnable on any machine with the Java Virtual Machine installed.

Android was chosen as the mobile development platform for this project because of the open nature of the system. It allows any application to be installed on a device, without it having to be put through a review process by a third party. In addition to this, Android does not limit the capabilities of any application, allowing system functionalities to be replaced and performed by our application. This is particularly important, as deliverable 3 will replace the default Android keyboard with a custom Character Recognition keyboard. Other systems, specifically Apple iOS, require all apps to be reviewed before they can be installed, and do not allow system features to be replaced by a third party application.

The system will be created as a set of 4 components, separating the algorithm from each of the deliverable applications. Components will then be able to use others in order to maintain consistency and reduce code duplication. The components, and the relationships between them, are shown in Figure 5.

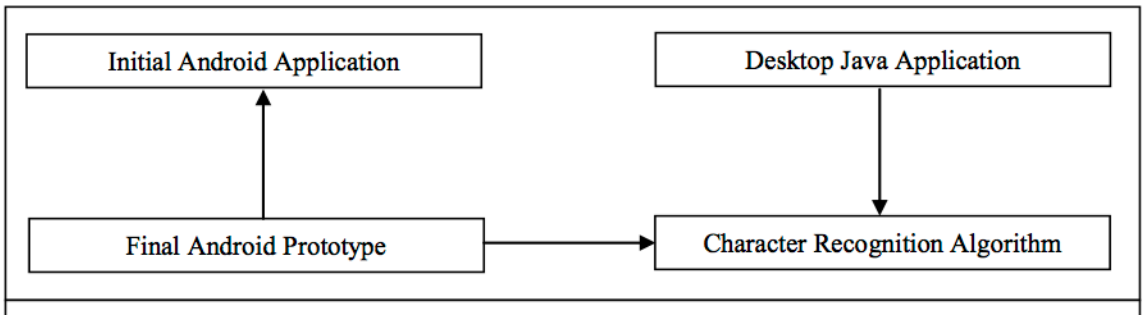

Figure 5: Diagram showing the different components of the system. An arrow shows that one component uses another.

The Character Recognition Algorithm component is designed to allow any application to perform the algorithm on drawings that it provides. This component will not only contain the stages of the algorithm itself, but also any objects that are used during the algorithm. It will not be possible for separate applications to make use of each individual stage of the character recognition algorithm, but they will be able to run the full process.

Both the Android Application for Recognition and the Desktop Java Application make use of the Character Recognition Algorithm component, allowing both applications to perform character recognition on saved drawings. Therefore it is a requirement that the algorithm run correctly on both the Java Virtual Machine and the Dalvik (Android) Virtual Machine. For most stages, this is not a problem because of the similarities between the Java and Android APIs. However, each API handles database connections differently, and so the Database Comparison stage of the algorithm needs to be implemented separately for both platforms. This will be done by only including an interface for the stage in the Character Recognition Algorithm component. Any application that uses the algorithm must then implement this interface with platform specific database functions, and provide an instance of this implementation when performing character recognition.

D. User Interface

Both applications will be designed with a relatively minimalistic interface, to keep the focus of the project on the character recognition algorithm itself.

The Desktop Java Application window (see Figure 6, left) will be dominated by a large drawing of the currently selected character. It will also provide a location bar to allow the user to navigate to a different folder, a file list to allow the user to change between character drawings in the current folder, and a box detailing information about the current drawing and statistics about the overall performance of character recognition. Finally there will be buttons to run recognition on the current drawing, and to run recognition on all drawings in the current folder, with the results of these operations provided on screen upon completion.

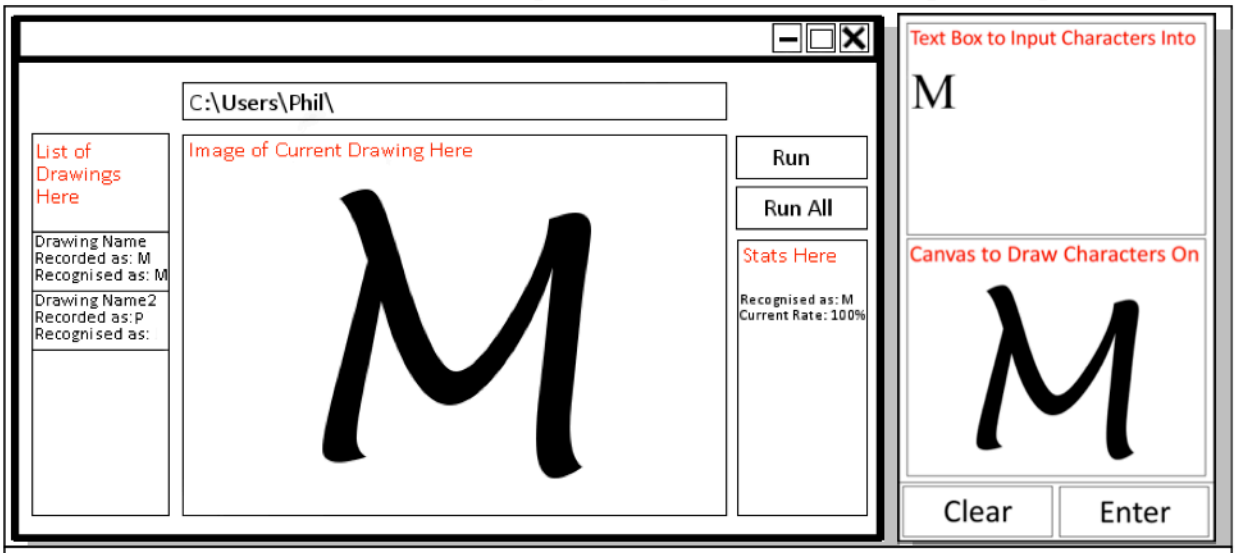

Figure 6. Left: A mock-up of the design for the Desktop Application for Recognition. Right: A mock-up of the design for the Android Application for Recognition.

Similarly, the Android applications (see Figure 6, right) will be dominated by a large canvas, allowing the user to draw characters on screen. This will be complimented by a header bar and a button bar. For the initial application for input, the header bar will inform the user of the character they should draw, and the button bar will contain buttons to clear the canvas or save the current drawing. For the application for recognition, the header bar will be larger, and contain a text box to store the characters that have been recognised from the users drawing. The button bar will then allow the user to enter the character they have drawn.

E. Training

Database Comparison is an important stage of the recognition algorithm, as it is the point at which the algorithm searches for a character signature that matches the signature of the drawing. The database will therefore need to be populated with valid character signatures in order for the algorithm to provide a result.

Initially, these signatures will be created by looking at computerised fonts and determining the primitives that each character is composed of. There is, however, an infinite number of ways to draw each character, and so there are likely to be many different signatures detected by the algorithm for different drawings of the same character. Adding all these into the database would be impossible, as we have no way of predicting what the algorithm will see before it actually sees it. Therefore the database will have to be continually populated throughout the testing phase of development.

To assist with this, the algorithm will log the signatures of all drawings that it fails to find a match for. With the desktop application, it will also log the signatures for all the drawings that are recognised as the wrong character. This log will then periodically be reviewed. Each entry in the log will be redrawn, and the identified signature examined, to see if the signature is an accurate representation of the drawing. If this is the case, then the signature will be added to the database as a match for the corresponding character.

Agarwal and Kumar do not mention how to populate the database for the algorithm, but do state that the complete and partial rotation stages result in there being “no need to spend [a] large amount of time in training as in the case of neural network based approaches". This is because these stages allow for characters to be recognised even if their exact signature is not in the database.

F. Evaluation

To determine the success of the Android Application for Recognition, and therefore its potential to be developed into a fully functional Android keyboard, we must evaluate its performance and compare this to other character input methods for the same device. This will be achieved by asking participants to partake in an experiment, described below. Because of the difficulty in finding participants they will be selected through convenience sampling, anyone who can attend the experimental sessions will be allowed to take part, rather than through random or cluster sampling.

The experiment will take the form of 3 small exercises performed on the Android device. Each will ask the user to enter a small piece of text into the device, using:

1. An on-screen QWERTY keyboard
2. Voice Recognition
3. Android Application for Recognition (Deliverable 3)

The piece of text entered will be the same for each exercise, and the participant will be given this beforehand so that they are not learning it during any of the exercises. The order in which each participant completes these exercises will be chosen at random, as to reduce the effect that this may have on the results.

During each exercise the device will record statistics about the performance of each character input method. This will include the time taken to enter the text, the accuracy of the final text and the number of corrections that the user has to make. These statistics will allow the performance of the prototype to be measured and compared with the other methods.

After each exercise, the participant will complete a short survey. This survey will include ordinal scales, which ask the participant to rank aspects of their experiences of a scale from 1 to 5. These experiences will include how much they enjoyed using the input method, how easy it was to use the input method, how reliable they found the input method and how likely they would be to use the input method in future. The surveys will provide a different perspective of the prototypes performance, based on opinions rather than statistics.

III. REFERENCES

Google Inc., 2010. *Gingerbread | Android Developers*. [Online]
Available at: http://developer.android.com/about/versions/android-2.3-highlights.html
[Accessed 17 January 2013].

Agarwal, S. & Kumar, V., 2005. Online Character Recognition. *Third International Conference on Information Technology and Applications,* Volume 1, pp. 398-703.

Apple Inc., 2012. *What's New in iOS: iOS 5.1*. [Online]
Available at:
http://developer.apple.com/library/ios/#releasenotes/General/WhatsNewIniPhoneOS/Articles/iOS5_1.html#//apple_ref/doc/uid/TP40012663-SW1
[Accessed 17 January 2013].

Chan, K.-F. & Yeung, D.-Y., 1998. Elastic Structural Matching for On-line Handwritten Alphanumeric Character Recognition. *Fourteenth International Conference on Pattern Recognition, 1998. Proceedings,* Volume 2, pp. 1508-1511.

Feder, J., 1971. Plex Languages. *Information Sciences,* 3(3), pp. 225-241.

Ofcom, 2011. *Ofcom | Growth of smartphones and mobile internet use*. [Online]
Available at: http://stakeholders.ofcom.org.uk/market-data-research/market-data/communications-market-reports/cmr11/telecoms-networks/5.20
[Accessed 2 January 2013].

Plamondon, R. & Srihari, S. N., 2000. On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 22(1), pp. 63-84.

Strategy Analytics, 2012. *Worldwide Smartphone Population Tops 1 Billion in Q3 2012*. [Online]
Available at: http://www.businesswire.com/news/home/20121017005479/en/Strategy-Analytics-Worldwide-Smartphone-Population-Tops-1
[Accessed 2 January 2013].

University of Surrey, 2012. *The Chars74K dataset*. [Online]
Available at: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/
[Accessed 27 October 2012].

Yoshida, K. & Sakoe, H., 1982. Online Handwritten Character Recognition for a Personal Computer System. *IEE Transactions on Consumer Electronics,* CE-28(3), pp. 202-209.