SKETCH-2-IMAGE SEARCH FOR iPADS

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Sukhada Pendse

Xiaochen He

M.Eng Field Advisor: Prof. Tsuhan Chen

Degree Date: May 2012

Abstract

Master of Engineering, ECE Program

Cornell University

Design Project Report

Project Title: Sketch-2-Image Search using iPads

Authors: Sukhada Pendse, Xiaochen He

Abstract: High-performance tablet devices like the iPads are becoming more and more popular with rapid advancement in computer hardware development. Armed with fast CPUs, GPUs, touch screens, cameras and inertial sensors, they have become an ideal device for performing various computer vision tasks. The Apple iOS platform supports interactive tasks on its iPad and iPhone and has the ability to store and search large amounts of data from its memory and from the web. In this project, we exploit these characteristics to develop an iPad/iPhone app for image search from hand sketches on the touch screen. The app recognizes objects from their sketches drawn on the touch screen, sends a query containing object name and location to the server, and retrieves images similar to the sketch from the server. The server performs the image search on a large database of images. The app is, therefore, a sketch-recognition app that can provide feedback to the user. We use pictorial structure models for object detection [1] on the iOS platform and a Similarity Algorithm [2] implemented in MATLAB at the server end for image search. The query transmission and image retrieval is on a persistent HTTP connection.

Acknowledgement:

We thank Prof. Tsuhan Chen for providing us the opportunity to work in the Advanced Multimedia Processing Laboratory and for his support and guidance throughout the duration of the project. Our special thanks to our mentors Adarsh Kowdle and Amir Sadovnik, both members of the AMP Lab, for helping us out with the nitty-gritties of the project. We are thankful to all the members of the AMP Lab for their support. Last but not the least, we thank Apple for their fantastic iOS platform, MathWorks for their software and the ECE Department, Cornell University for exposing us to cool stuff through the Design Project.

Table of Contents

i.	Abstract	i
ii.	Acknowledgement	ii
iv.	Executive Summary of Accomplishments	iv
1.	Project Overview	1
2.	Framework and Division of Work	2
3.	Sketch-Recognition and UI	
3.1.	User Interface Design	3
3.2.	Sketch-Recognition Algorithm and implementation	9
3.3.	Query formation and query handling	15
4.	Server Process	
4.1.	Preliminaries	16
4.2.	LabelMe Database and Toolbox	17
4.3.	Similarity Algorithm	18
5.	Results and Conclusions	21
6.	Further Work	23
7.	References	24

Executive Summary of Accomplishments

The Sketch-2-Image Search project is motivated by the goal of enabling image search using hand-drawn sketches. This is a relatively new field and with the rapid progress in multimedia technology, a very useful one. Little work has been done in the area and although approaches for object-recognition from photographs and sketches have been developed, the synergy between computer vision algorithms, machine learning and sketch-recognition algorithms has not been explored. We attempted and were able to explore this synergy.

Using the iOS platform for executing the idea of search-from-sketch is a major accomplishment, since the Apple store does not, yet, have a sketch-recognition app, which has now been developed as a part of this project. The app is fairly fast and can be used on iPad and iPhones alike. The app makes image search handy and widely applicable since internet surfing today is mainly performed from touch devices, more than half of which are iPhones/iPads. The resultant app is able to communicate with the server, query it for images similar to the recognized objects in the sketch and retrieve images the server searches for.

The Similarity Algorithm implemented on the server implements image search using relative positions of objects in the sketch, their sizes and their relative scaling. The scalar and rotational invariability is an added feature to the Similarity algorithm [2].

Overall, the project has accomplished two milestones – one, it has built a fast sketch-recognition app for iOS and two, it has provided a proof of concept for image search from sketches.

The app was successfully demonstrated at the ECE Day at Cornell University and was working well all through the day. Visitors appreciated the UI and the novelty of the app and its usefulness.

Chapter 1

Project Overview

It would not be an exaggeration to say that the world today breathes internet. People are 'online' from their homes, offices, cars, from the mall, the gym, here there and everywhere. Hand-held touch devices like the Apple iPads, iPhones and iTouch and tablet devices have made internet surfing a natural and easy task. With accelerometers, fast CPUs, GPUs and large memories these devices support a multitude of interactive tasks as well.

Why not use these features for a naturally inspired way to search for images? Yes, searching for images from hand-drawn sketches is probably the next big step for the internet. Touch devices and their superior capabilities of processing and storing large amounts of data inspire the Sketch-2-Image Search project in the Advanced Multimedia Processing Lab at Cornell University. The Advanced Multimedia Processing Laboratory, headed by Prof. Tsuhan Chen, is known for its application intensive research and iOS-based projects. Our project – developing an iPad/iPhone app for Sketch-2-Image Search is a part of this.

Little work has been done on this front, and the work of Tao Chen, Ming-Ming Cheng, Ping Tan, Ariel Shamir, Shi-Min Hu in 'Sketch2Photo'[3] is probably the best so far. Sketch2Photo uses sketches for obtaining the relative positions of objects in the 'desired' image, but each object is labeled with its class name. Therefore, sketch to image conversion is supported by text input from the user. Amir Sadovnik, Tsuhan Chen [1] used the work of Pedro et al. [4] to develop a fast sketch-recognition algorithm using pictorial object models and probabilistic graphical models for objects. We, therefore, decided to put the two and two together (our Similarity algorithm is, however, different from [3]) and package the good stuff into an iPad app. The project, therefore, uses ideas from image processing, computer vision and machine learning.

The objectives of this project are:

1. To implement a sketch-recognition algorithm on a touch device
2. To send a query to the server detailing the object class and its relative location in the sketch
3. To design an iPhone/iPad app for the sketch-2-image search application

Chapter 2

Framework and Work Division

Based on the project goals and related work in the field, the system framework was clear:

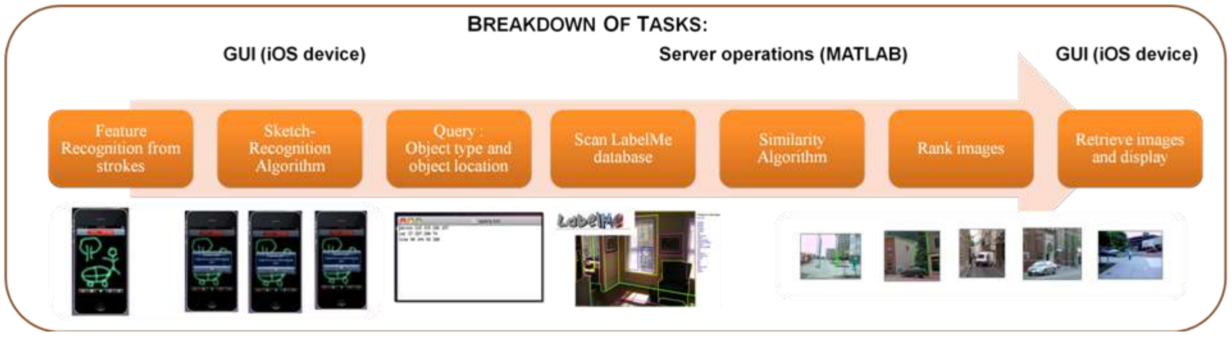

BREAKDOWN OF TASKS:
GUI (iOS device)
Server operations (MATLAB)
GUI (iOS device)
Feature Recognition from strokes
Sketch-Recognition Algorithm
Query : Object type and object location
Scan LabelMe database
Similarity Algorithm
Rank images
Retrieve images and display

Fig 1. System Framework

The project was, therefore, divided into two interdependent parts –

A. UI and sketch-recognition

B. Server processing – image search and ranking

Sukhada Pendse worked on the development of the UI and implementation of the sketch-recognition algorithm. Xiaochen He worked on the implementation of the search algorithm and query handling at the server end. Sukhada worked on the iOS platform while Xiaochen has used MATLAB for implementation. The server also has a few php scripts for query handling, which were borrowed from an earlier project.

The input to the application is a sketch drawn on the iPad/iPhone screen by the user. The sketch-recognition algorithm breaks it down into 'strokes' to extract feature vectors (features are explained in the next section) corresponding to each stroke in the sketch, from which inter-stroke normalized feature vectors are calculated and fed to the graphical models (classifiers) for different objects. The objects identified in the sketch are encoded into a query, which is then sent to the server. The server operations identify the five closest matching images to the sketch via the Similarity Algorithm. The user's iPad/iPhone then pulls the images from the server and displays them as the output of the system.

In accordance with this, Sukhada handles the input interface, i.e. taking a sketch and recognizing objects from the sketch to output a query for the server. Xiaochen takes this query vector to search the LabelMe [5] database of images for the closest match image. The link between the above parts is the query vector, which is mutually agreed upon, as explained in further sections.

Chapter 3

3.1 User Interface Design

A primary objective of this project is to develop an iPad/iPhone app to enable image search using hand-drawn sketches on the iPad/iPhone screen. The UI design for the project was therefore, completely implemented in iOS (the operating system used by Apple). We used Xcode 4.2 for implementing and testing the UI design using the in-built simulator.

The key aspects of UI design the application requires are –

1. It should be easy to use and have instructions for the user at each step
2. The UI should allow the user to sketch as many objects as he/she wants to, possibly overlapping and in different colors etc.
3. The UI should warn the user if there is no recognizable input and/or if the algorithm is not able to find any objects in the sketch
4. The UI should show the user a message as to what objects were found in the sketch, and give the user an option to redraw
5. Once the user confirms the sketch, the UI should fetch the images and prompt the user to look at those, in a ranked fashion

These objectives can be divided into tasks –

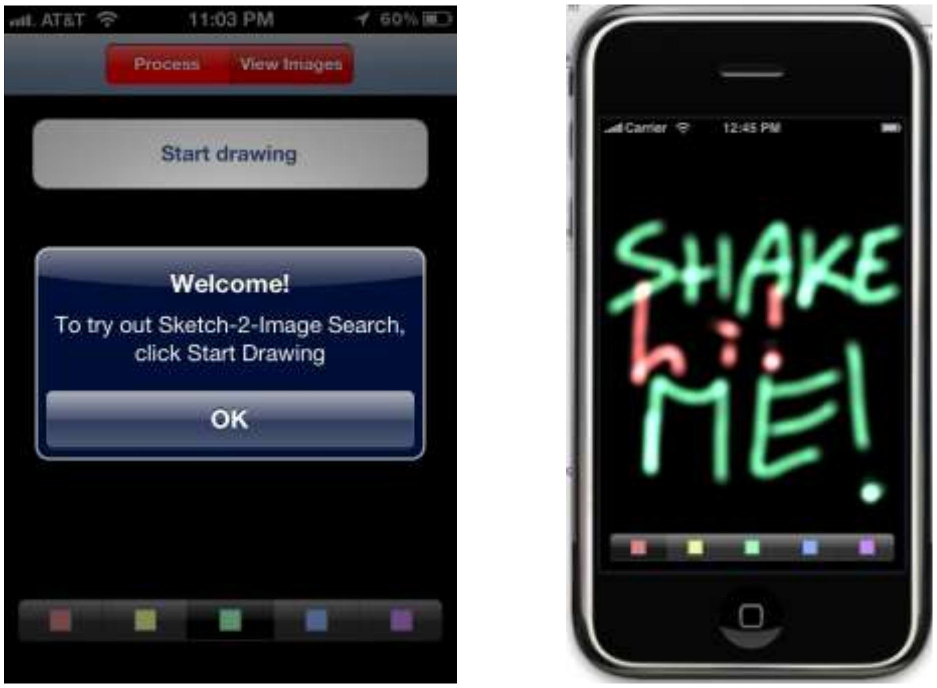
1. Sketching
2. Processing the sketch drawn
3. Viewing images found by the algorithm

We will see how each has been achieved in the Sketch-2-Image app.

3.1.1 Sketching

A search for apps that implement (2) above, led to finding GLPaint [6], a free app on the Apple Store that enables the user to sketch drawings on the screen using touch. It allows the user to change colors as well as erase the drawing. GLPaint, therefore, was used as the backbone and modified to suit the other requirements above.

A screenshot of the GLPaint app and the Sketch-2-Image app is shown in Fig.2



Welcome!

To try out Sketch-2-Image Search, click Start Drawing

Fig 2. A snapshot of the Sketch-2-Image Search app and GLPaint app

GLPaint identifies the ‘touch begin’ and ‘touch end’ positions on the screen when the user begins sketching and ends a stroke. It samples the smooth drawing of a stroke via the touches and interpolates between two consecutive samples. GLPaint, therefore, implements pixel-by-pixel sketching, which is important in the processing (sketch-recognition) step. GLPaint achieves these functions through its class ‘PaintingWindow’, which uses the objective-C View ‘PaintingView’. Addition of the PaintingWindowController was made to facilitate switching between ‘Views’, since the GLPaint app only supports drawing, but our app requires switching to ImageView and so on. ImageViewController helps view images obtained from the server.

After launching, the Sketch-2-Image app prompts the user to ‘Start Drawing’. The user can change colors while he/she is drawing, can overlap the strokes and draw as many objects as he/she wants.

The requirement of the sketch-recognition algorithm, however, is that each part of the object be drawn as one stroke, as explained in the next section. Similarly, objects cannot be drawn haphazardly, but need to be drawn according to the sample object sketches as shown in Fig 3, for best results. Each different color in the figures represents a new stroke. This makes it easier for the algorithm to recognize objects, since drawing is a skill and users might draw sketches that do not resemble the objects they want to search for. Guidelines smoothen out the effect of such variations.

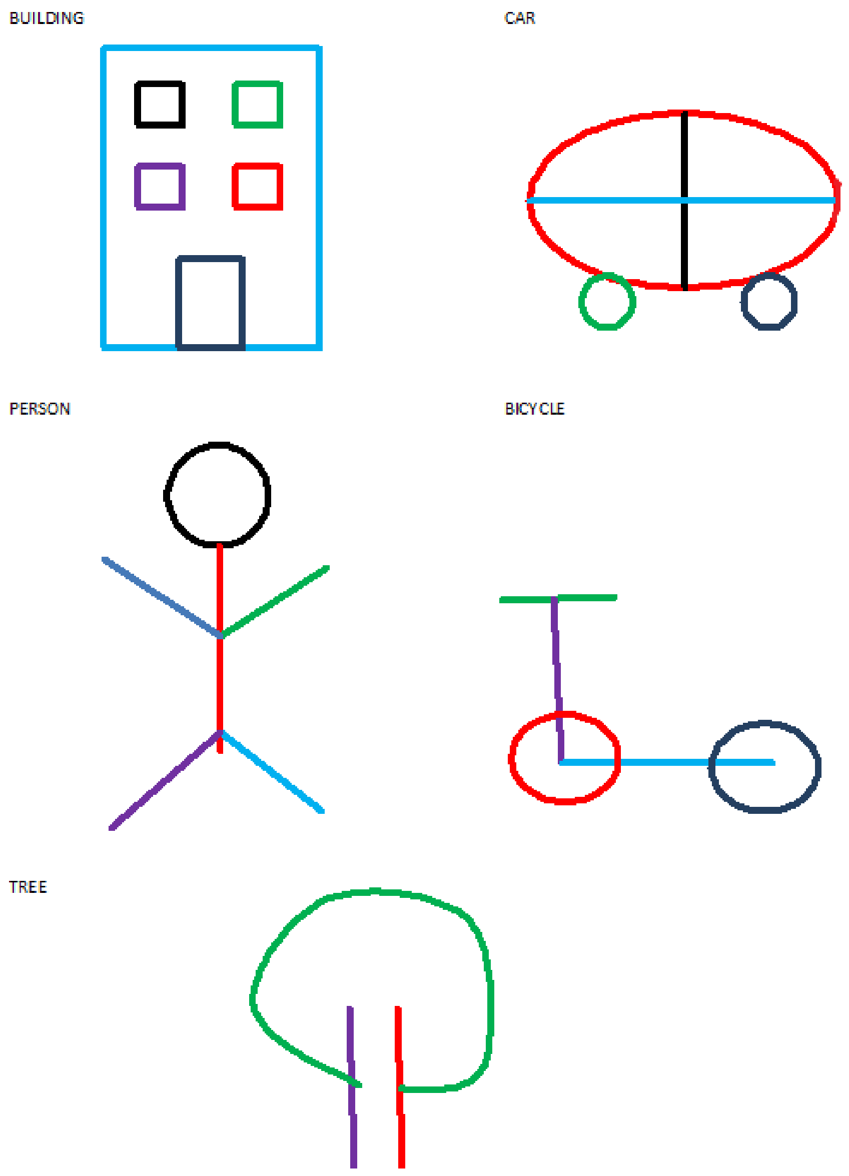
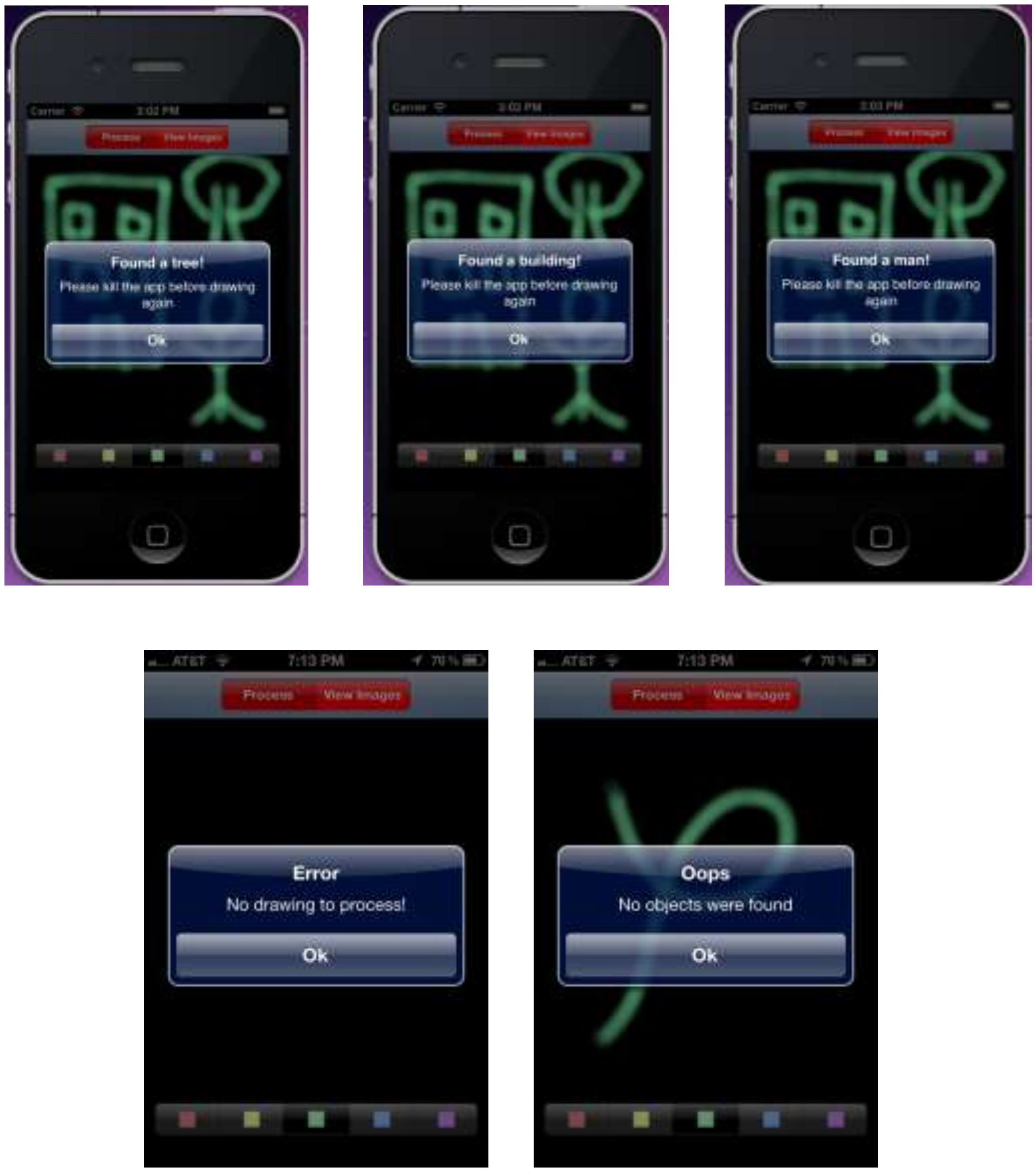


Fig. 3. Sample object models

3.1.2 Processing the Sketch

After sketching is complete, the user presses the 'Process' button on the top navigator bar which triggers the sketch-recognition algorithm. The 'Process' button calls a function that checks what the user has sketched. If it finds the drawing empty, it displays an alert message (using class UIAlertView in objective-C) to the user, saying 'No Drawing To Process!' The user can then relaunch the app and start drawing again. If it finds less than three strokes (which is the least number of strokes across our models), it displays an alert again, saying 'Oops, no objects found'. If it finds more than three strokes, however, it calls the feature extraction algorithm and the operations proceed to the actual algorithm explained in the next section. Again, after the algorithm recognizes object(s) in the sketch, it displays alerts to the user – 'Found a <object class>'. The user can verify whether those are the objects he/she wanted in the sketch and if not, redraw. The alerts are easy to handle and can be made to disappear with one click on 'OK'.

Example 'alert' messages displayed to the user are shown in the following figure: (Fig 4)



Found a tree!
Please kill the app before drawing again
Ok

Found a building!
Please kill the app before drawing again
Ok

Found a man!
Please kill the app before drawing again
Ok

Error
No drawing to process!
Ok

Oops
No objects were found
Ok

Fig 4. Alert messages displayed to the user during Processing Step

3.1.3 <u>Viewing images found by the algorithm</u>

The processing step calls the sketch-recognition algorithm, which in turn, sends a query to the server. The server searches for images similar to the query sent to it and uploads them in a folder accessible by the Device ID (every Apple device has a unique Device ID). The user then merely has to click the 'View Images' button on the top navigator bar on the screen to view the images. The first click brings him to a dummy image and displays a toolbar where the user can click on indicators 'One, Two, Three, Four, Five'. The numbers represent the ranks of each image, as found by the Similarity Algorithm. Therefore, image displayed on choosing 'One' on the toolbar is closest to the query, while that displayed on choosing 'Five' is the least close to the query from amongst the five images found.

All the images are scaled by the app to an 'almost' square size (380 x 400) for better view on iPhone/iPad screen. Internally, on choosing 'One', the device fetches the image ranked highest from the server folder, puts it in the ImageView (using the function imageview.image = rank_1_image, from class UIImageView in objective C) and then loads the ImageView on top of the current view (which is the painting window that the user sketches on). Example display images are shown in Fig 5a and 5b.

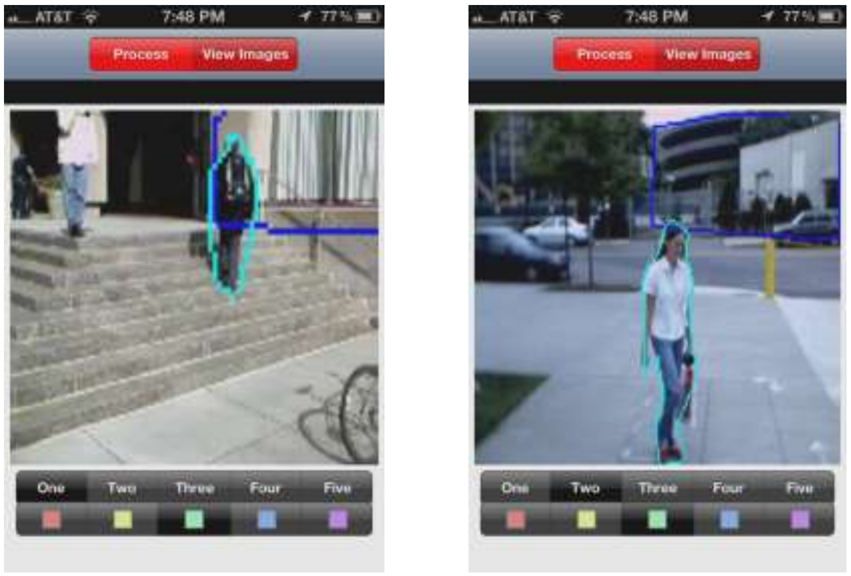


Fig 5a. Example images displayed on the app after user clicks on 'View Images'

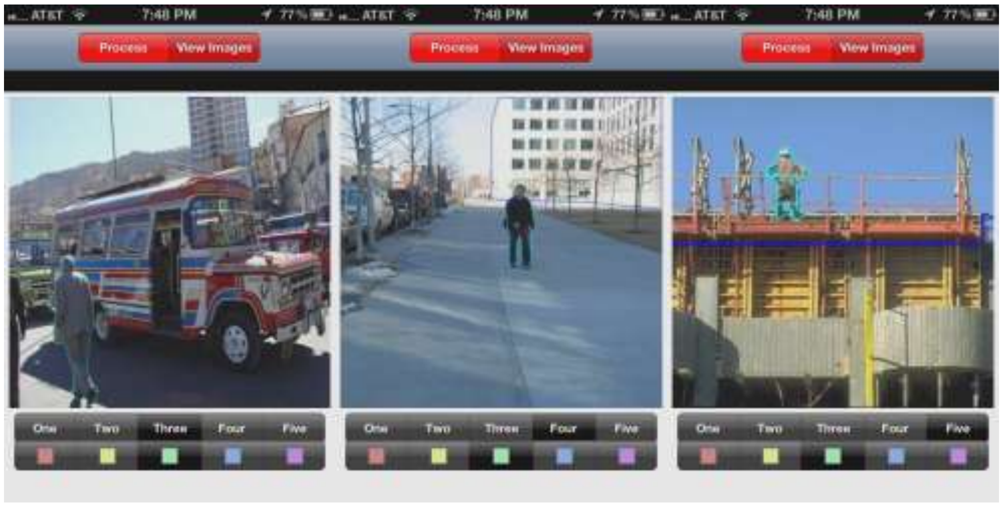


Fig 5b. Example images displayed on the app after user clicks on ‘View Images’

This completes the UI Design part of the application, and we next look at the Sketch-Recognition Algorithm and its implementation in detail.

3.2 Sketch-recognition Algorithm and Implementation

Sketch-recognition algorithm and its implementation in the iPad/iPhone itself is the most important part of the project and a key feature of the app. It is the first sketch-recognition app on the Apple Store, and probably the first app which embeds a machine learning algorithm. The sketch-recognition algorithm is built on the principles of machine learning – generative learning and probabilistic graphical models. The work of Amir Sadovnik, Tsuhan Chen [1] explains the basis of the algorithm and its key ideas. We will focus on the implementation and execution parts of the algorithm.

For our convenience, we can divide the algorithm into sub-tasks:

1. Feature extraction
2. Likelihood calculations
3. Viterbi implementation
4. Thresholding and verification
5. Exhausting all models

All these subtasks are performed by the device itself, within an acceptable time frame.

3.2.1 Feature extraction

As proposed in [1], we break down a sketch into a collection of strokes. Each stroke begins when the user touches the screen and ends when the user brings up his/her finger. These events are recognized by inbuilt functions ‘touchesbegan’ and ‘touchend’ in the GLPaint app.

We modified these functions to call another function which stores the coordinates where touch began and where touch ended, as well as all other pixel locations that the user moved his fingers on, on the screen, capturing the stroke location, pixel by pixel. The coordinates are stored in a global array called ‘storedvals’. The newly defined Class globalarray has declarations and implementations for the array. In iOS, global variables are difficult to handle and hence, seldom used. Our project, however, needs the array to be accessible to both the drawing function and the calculations, hence we used globalarray.

Globalarray storedvals is initiated by storing a (-1) as its first element and then stores the x-coordinate followed by y-coordinate of each pixel of each stroke. The pixel coordinates of two different strokes are separated by ‘-1’. In the end, an extra (-1)s is added to maintain symmetry.

The structure of the global array storedvals is therefore as follows:

-1	x_{11}	y_{11}	x_{12}	y_{12}	x_{13}	y_{13}	-1	x_{21}	y_{21}	x_{22}	y_{22}	x_{23}	y_{23}	x_{24}	y_{24}

Here, (x_{11}, y_{11}), (x_{12}, y_{12}), (x_{13}, y_{13}) are the (x,y) coordinates of the three pixels which form stroke 1, while (x_{21}, y_{21}), (x_{22}, y_{22}), (x_{23}, y_{23}) and (x_{24},y_{24}) are the (x,y) coordinates of the pixels which form stroke 2. Note how y13 and x21 have a (-1) between them to indicate that one stroke has ended and the other has begun. This is important because feature extraction is based on stroke information.

Each stroke has a bounding box, which indicates the Xmax, Ymax, Xmin and Ymin coordinates. Once the 'Process' button is pressed, the boundingboxCalc function is called, which calculates the bounding box for each stroke and stores it in an array named 'bb', as [Xmin Ymin Xmax Ymax], again two strokes separated by (-1). Based on the bounding box coordinates, we extract the features for each stroke as mentioned in [1]:

1. X-center of the bounding box
2. Y-center of the bounding box
3. Diagonal length of the bounding box
4. Rectangularity of the stroke
5. Angle of the diagonal from horizontal (theta)

Rectangularity of a stroke is the fraction of the number of pixels in the stroke with respect to the total area of the stroke's bounding box. It is important to differentiate between a filled circle and an empty circle, for instance. The features for each stroke are shown in Fig. 6

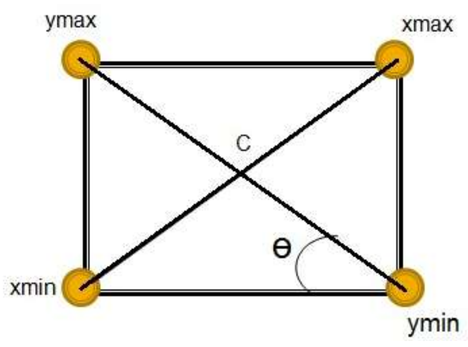
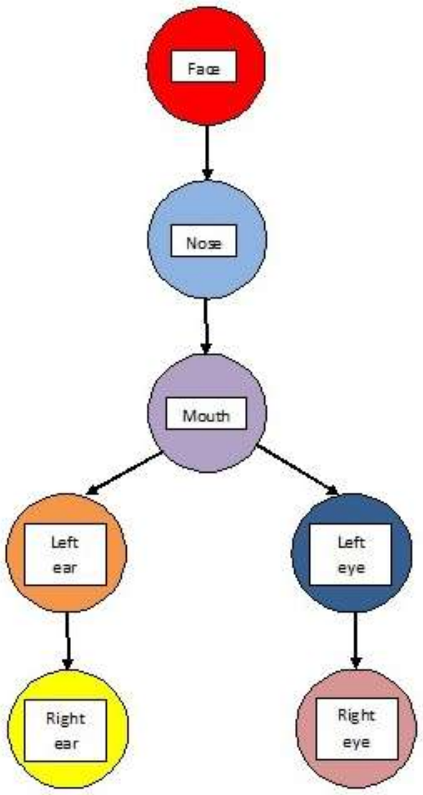


Fig 6. Feature calculations for one stroke

What the algorithm needs, however, are inter-stroke features. Hence, we calculate inter-stroke features, by calculating the difference between the corresponding features of the two strokes. The features 1, 2 and 3 are normalized by the diagonal length of the bounding box, to account for size differences in the sketch. These are stored in a [Nscribble x Nscribble x No. of features] array, named ‘featmatrix’. Nscribble is the total number of strokes and number of features is 5.

3.2.2 Likelihood Calculations

The featmatrix obtained is then used in the object models to calculate log likelihoods. An object model is a tree structure, with each node being a part of the object. The link between parent node and child node represents the dependency of two parts of the object on each other. Each link has μ and σ values (for a Gaussian distribution) for the five inter-stroke features, wherein the two strokes represent the two nodes linked in the sketch. This is illustrated in the following Fig. 7



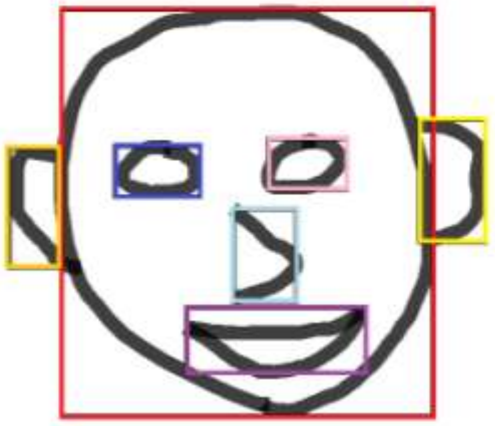


Fig 7. Object model for face and an example sketch with bounding boxes.

Here, the tree model for face has a root which is the stroke representing the ‘face’. The nose depends directly on the face (in terms of its location and other features) while the mouth depends on where and how the nose stroke is. Similarly, the locations of left ear and left eye depend on where the mouth is, but once the left eye and ear are determined, right eye and right ear can be determined by comparing other strokes with the strokes representing the left eye and left ear. E.g. their bounding boxes will have diagonals of nearly equal length, their rectangularities will be almost equal, and so will be their y-coordinates. The inter-stroke feature 3 (diagonal length) for the link between left eye and right eye will therefore have $\mu \approx 0$ and a small value for σ. If σ is large, more variation would be allowed in the value for diagonal length which is not desirable. The μ, σ values are used to calculate the likelihood of two strokes representing the parent and child nodes for a single link. Creating such models invokes the generative learning algorithm in machine learning. For creating a model, we consider a number of training sketches, and calculate the μ, σ values for the inter-stroke features for all links between all the nodes. However, only those links which 'seem' important i.e. have low values of σ or are logically 'close' to each other are kept in the tree model. The models are hard-coded in the code.

The likelihood of two strokes representing a link is calculated as:

$$L_{link} = \Pi \{\Phi \text{ (inter-stroke feature value} \mid \mu_{link}, \sigma_{link})\} + \Sigma L_{link,\ leaves}$$

where, the product is taken over all the five features, Φ is the Gaussian pdf, and μ, σ are the values for the features for that link. The additive term $L_{link,\ leaves}$ is the sum over all the leaves of the child node of the link and the importance of this term is in the Viterbi nature of the algorithm [1]. These likelihood values are stored as their negative logarithm values (log likelihood) in the [Nscribble x Nscribble x Number of nodes] matrix 'prob'. A term prob[i][j][k] represents the log likelihood of the strokes i, j representing the link between node k and its parent, when i represents the node k.

Note that we start calculating from the leaves upwards to the root, so that the summation term is added from the second stage. The final stage is calculating the likelihood of each stroke forming the link from root to the second-highest level of the tree. These likelihood values are stored in a separate array 'temp' for finding which stroke(s) represent(s) the root best. Note also that at each stage the likelihood calculations have been cumulative because of the additive term $\Sigma L_{link,\ leaves}$.

After calculating the likelihood values, we create an array of 'best matches' for each node. This [Nscribble x 2] array, named 'scores', represents the likelihood of the 'best match' and the stroke number of the 'best match'. For each node, we store the value Y where Y is given by – 'If my parent was stroke X, I can be best represented by stroke Y', where strokes X and Y have the lowest log likelihood of forming the link between this node and its parent.

3.2.3 Viterbi implementation

Now that we have travelled upwards from the leaves to the root, it is time to come down in an order that only finds best matches. Note that we had stored the best matches in the 'scores' array.

We first find the root, which is the stroke that has the lowest log likehood of being node 0 (root). Let us say stroke 2 is the best match for root. Now we traverse downwards, moving to the child of node 0 – let's say node 1 – and checking its 'scores' array. Since stroke 2 is the parent stroke, we check which stroke can best represent node 1, if the parent is stroke 2. This is found in the corresponding column of the scores array. If this is found to be stroke 4, we move down again, and check which stroke can best represent the child node of node 1, given stroke 4 is the parent and so on, till we reach the leaves. In this manner, every node has been assigned a unique stroke which represents that part of the sketch. The stroke assignment to nodes is stored in the array named 'labels'.

Note that to recognize the object, we do not really need the strokes, we can simply check whether the (cumulative) log likelihood of the root is less than a model_thresh, which is the threshold likelihood for the object model. However, the stroke assignments are useful while forming the query and hence we execute this step.

3.2.4 Thresholding and Verification

Since our likelihoods are assigned using the Gaussian function, we might end up assigning log likelihoods, albeit very large, to strokes which do not actually represent any object. As mentioned in the previous section, therefore, we need a verification step. This is done by checking whether the cumulative log likelihood of the root is less than a threshold value (called model_thresh, obtained by experimentation). If it is, we know that the object is present in the sketch and can proceed with the stroke assignment.

Although we have mentioned this as the fourth task in the algorithm, the thresholding is checked before performing stroke assignment.

Since the sketch can contain more than one objects of the same kind, we need to run the Viterbi step again, this time setting the already assigned strokes' log likelihood to infinity, so that there is no double detection. When there is more than one objects of the same kind present in the sketch, the assignment of the 'root' stroke gives more than one possible values (i.e. more than one strokes have cumulative log likelihood values of being the 'root' less than model_thresh), which are stored separately in an array called 'roots'. We run the Viterbi step till we exhaust this array and assign the strokes for each possible root.

3.2.5 Exhausting all models

Given that there can be more than one objects in the sketch, we need to execute the sketch-recognition algorithm for all the object models we have. For that, we need to clean up the 'prob' array – which stores log likelihoods, the 'scores' array – which stores the best matches and the 'labels' array – which stores the stroke assignment, since these are specific to a particular model.

The following table summarizes the sketch-recognition algorithm and its implementation:

Given: Array containing pixel locations.

1. Calculate bounding box for each strokes and store in array 'bb'
2. Use the information from bb to calculate the features for each stroke. The features are center coordinates, diagonal length, rectangularity and angle of the diagonal
3. Calculate inter-stroke features and store them in the 'featmatrix'
4. Calculate the log likelihood of two strokes forming each link in the model using the hard-coded values of μ and σ for each link. Start from the leaves and keep accumulating the likelihood till you reach the root node.
5. Find the best match stroke and best match log likelihood for each node, given its parent stroke assignment and store in the 'scores' array
6. Compare the log likelihood of the root node to the model_threshold. If it is less than model_thresh, proceed with the Viterbi algorithm for stroke assignment to the nodes. If not, go to 8
7. If the number of roots is more than 1, run steps 5 and 6 again, this time nullifying all the 'already assigned strokes'
8. Proceed to the next model and repeat steps 1-7.

Table 1. Implementing the Sketch-Recognition Algorithm

3.3 Query formation and query handling

The basic requirement of search is a well-defined query. With Sketch-2-Image, albeit the query is a sketch input by the user, it has to be translated to a query suitable for the search algorithm for use. Clearly, the query structure will also be determined by the search algorithm used at the server. Currently, the Similarity Algorithm used for search requires information about the object class and object top-left and bottom-right locations (in terms of the corresponding x,y coordinates) in the query. Therefore, after execution of the sketch-recognition algorithm, the application forms a query.txt file and uploads it to the server using a persistent HTTP connection.

The structure of the query.txt file is as follows:

<object class><space><object-top-left-x-coordinate><space><object-top-left-y-coordinate>

<space><object-bottom-right-x-coordinate><space><object-bottom-right-y-coordinate>

<newline>

After writing the x,y coordinates of an object, the next object data is printed on a new line.

After forming the query.txt file, the algorithm calls a function that creates the HTTP POST to be transmitted to the server. The post contains the boundary content necessary for HTTP and the query.txt file. The data encoding used is NSUTF8StringEncoding.

The same function establishes a connection with the server (chenlab@musicserver.ece.cornell.edu) and calls the php script sketch_upload.php in the folder ~/Sites/sketch. The php script reads the HTTP Post and creates a folder named <Device ID> in the ~/Sites/sketch folder. The query.txt file is uploaded to this folder. Next, the algorithm calls the MATLAB script that implements the Similarity Algorithm. The script on completion uploads the ranked images in the same folder ~/Sites/sketch/<Device ID> from where they are fetched by the UI of the application.

More sophisticated query structure can be explored, e.g. structure that provides information about the relative locations of the objects, their other attributes – e.g. color, sub-type and so on. However, for the Sketch-2-Image app, the aforementioned query structure works just fine.

We now move on to the server side for the search algorithm implementation.

Chapter 4

4.1 Preliminaries

The server essentially performs the search for images according to the query it receives. The server output is a ranked set of five images which it claims are similar to the query. In addition to it, the server needs to crop the images it finds so that excess data is not sent and possibly resize them for better display.

Based on the data in the query, the following processes are performed at the server:

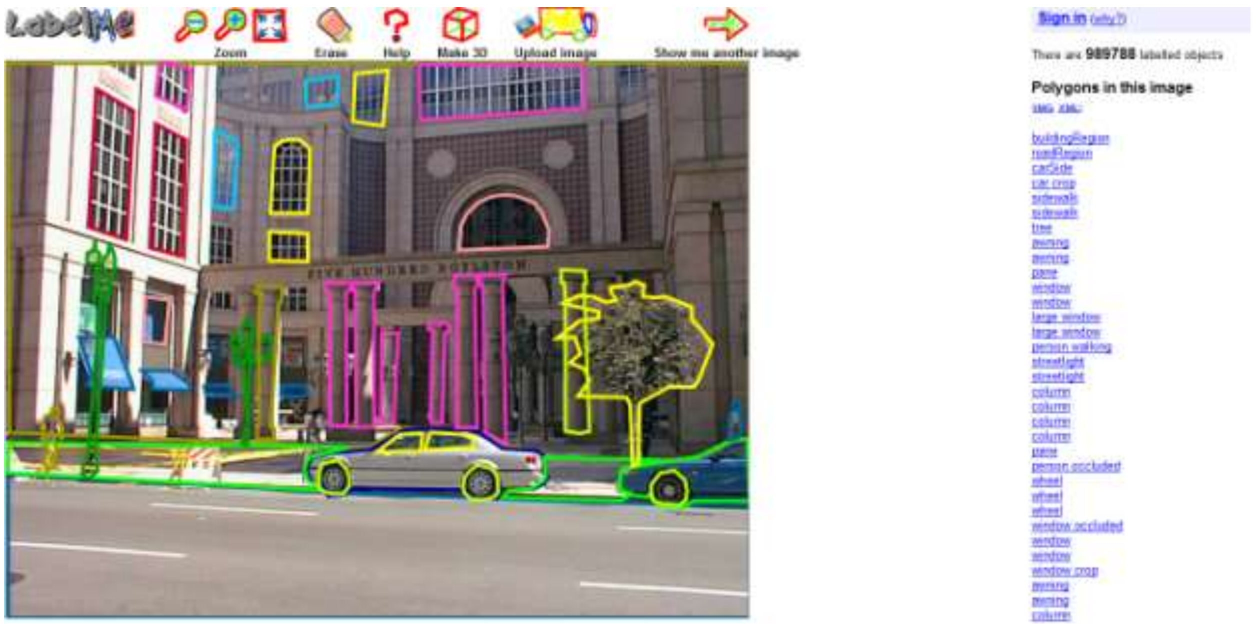
1. Create a subset of images from database such that all the images in this set contain at least the same objects as in query
2. Classify the objects in each subset image according to the type (name) of each object in the query. We don't care about those objects that are not in the query.
3. Each image in the subset has a set of sub-images, such that each sub-image has a permutation of all the objects in this image
4. Create new_query and new_sub-image and call the Similarity Algorithm [2] on each pair to get a similarity score for each database image
5. Rank the similarity score in ascending order and return the images with top five scores as the best match images.
6. Add a field 'deleted' in each object in image such that the matching object in Similarity algorithm has a value of 0 while others have value 1. Then outline these matching objects with different color
7. Set a bound to the matching part, and save the images in the same folder as the query

The Similarity Algorithm is described in [2] and in the next sections, we will have a look at the image database used and the improvements and shortcomings of using the Similarity Algorithm and possible modifications.

4.2 LabelMe Database

We use the LabelMe image database and MATLAB toolbox at the server. Created by MIT Computer Science and Artificial Intelligence Laboratory in 2008, LabelMe is a web-based image annotation tool that allows researchers to label images and share the annotations [5]. We have 30,000 database images in the server currently.

The LabelMe toolbox, which is a part of the MATLAB toolbox is useful to crop images from the database, search for images with specific annotations in the database and so on. Following figure shows an example of an image in the LabelMe database.



Sign in (why?)

There are **989788** labelled objects

Polygons in this image

IMG XML

buildingRegion
roadRegion
carSide
car crop
sidewalk
sidewalk
tree
awning
awning
pane
window
window
large window
large window
person walking
streetlight
streetlight
column
column
column
column
pane
person occluded
wheel
wheel
wheel
window occluded
window
window
window crop
awning
awning
column

Fig 8. Example image from the LabelMe database

The highlighted areas have been annotated by users using tags shown on the right hand side.

4.3 Similarity Algorithm

We use the Similarity algorithm [2] to compare the similarity between the query and each database image. Basically, this algorithm first makes a loop for pivot vector in the query, which is formed by the first two objects. Then it computes every angle between pivot vector and each other vector. The same thing is done in the database image so that we can get the difference between each pair of corresponding angles. After the pivot vector loop finishes, the biggest difference is chosen as the similarity score between the query and this image. Note that we calculate angles based on the locations we obtain from the query.

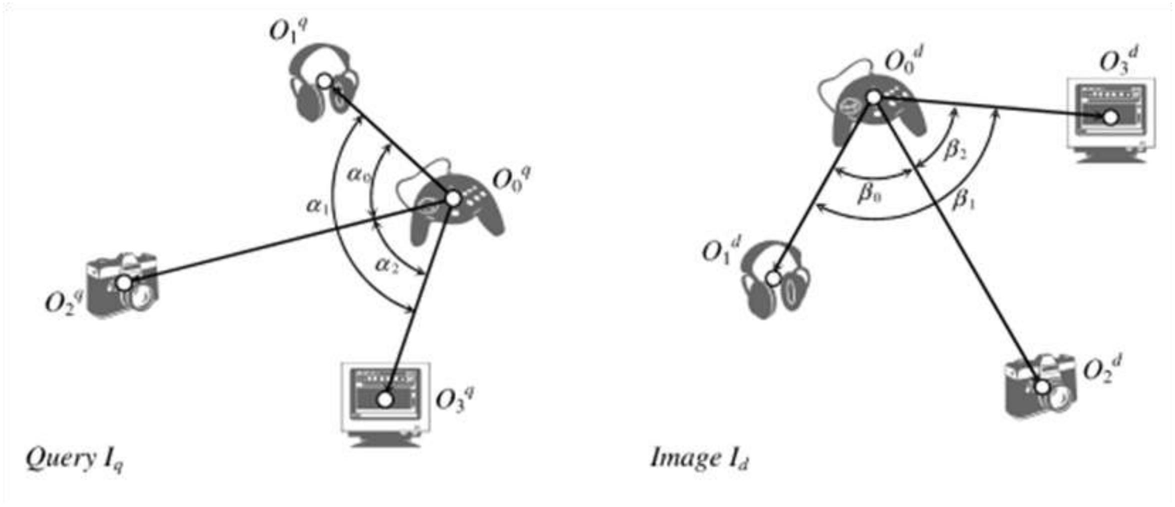


Fig 9. Similarity Algorithm making use of angles

However, there are a few constraints of the Similarity algorithm we deal with:

1. It does not deal with the translation problem. If the objects in an image rotate together by the same angle, the algorithm still thinks that it is the same image similar to the query as before.

2. It treats each object as a point, without taking into consideration the size of any object.

3. If we have multiple object of the same class in database images, the algorithm fails.

To deal with the above problems, we have modified the algorithm to suit our needs:

1. Instead of using arccos to get the angle, we use arctan. It takes direction into consideration.

2. We treat the object top and object bottom as two separate objects and new_query and new_sub-image are created by treating the top and bottom coordinates as separate objects with the same name. These modified query and sub-image are used in the Similarity Algorithm. This solves the scale problem to an extent.
3. For each database image, we create a set of sub-images which only contain the same objects as in the query

These modifications have shown good results as compared to using the original algorithm. A few results are shown below.

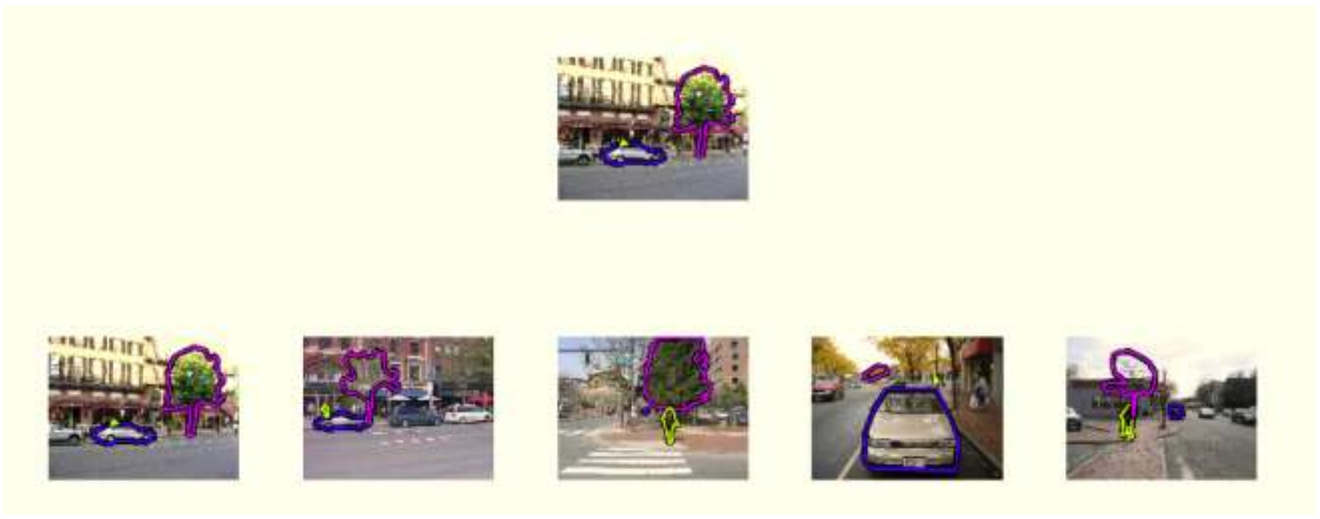


Fig 10a. Images found with only center coordinates

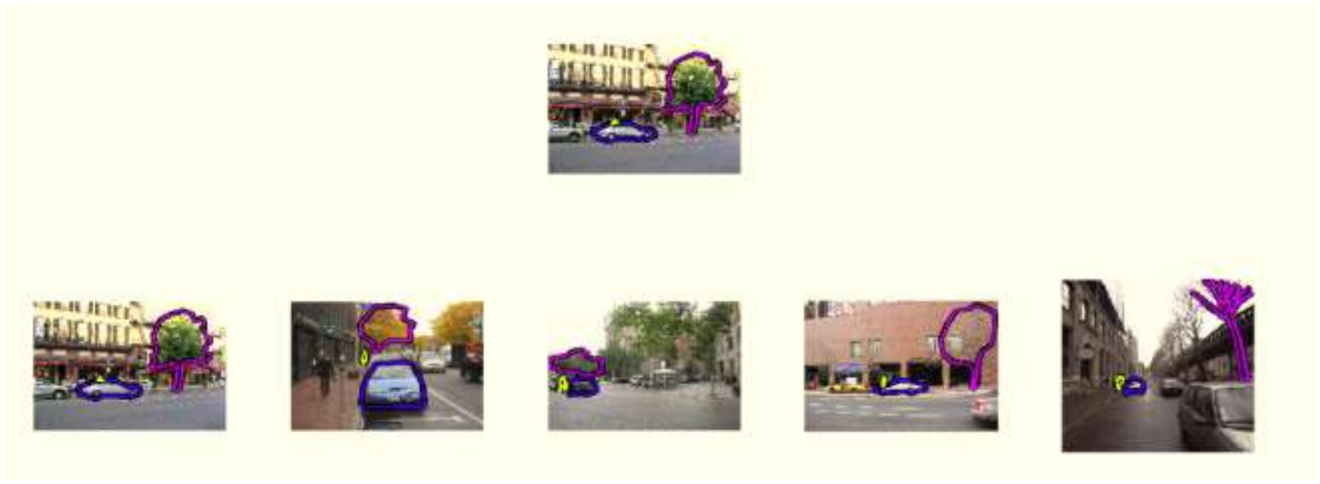


Fig 10b. Images found with the top and bottom coordinates

We can see that with only center coordinate of each object (Fig 10a), there is problem of rotation (as in the 4th returned image), and the relative size of objects are sometimes not in conformity with that in query (as in the 5th returned image, the size of the person is much larger than that of the car, which is not the case in query). After modifying the algorithm, which adds the top and bottom coordinates in each object (Fig 10b), not only the rotation problem is well solved (the relative position of each object matches well) but also the size of objects keep in conformity with query.

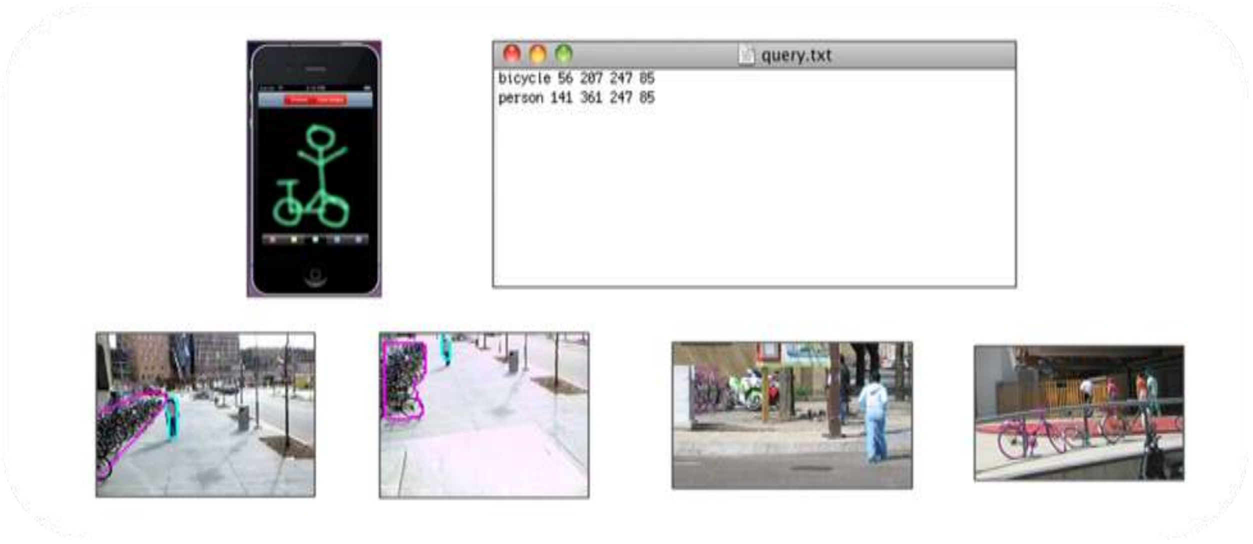
Chapter 5

Results and Conclusion

The Sketch-2-Image application was tested throughout the process of its development on the Xcode version 4.2, using the inbuilt simulator. It was seen to provide very good results, despite not using the appearance features. The object classes used were – person, building, car, bicycle, tree and these were chosen according to the objects available in the LabelMe database. Their models are as shown in Fig 3 on Pg 5.

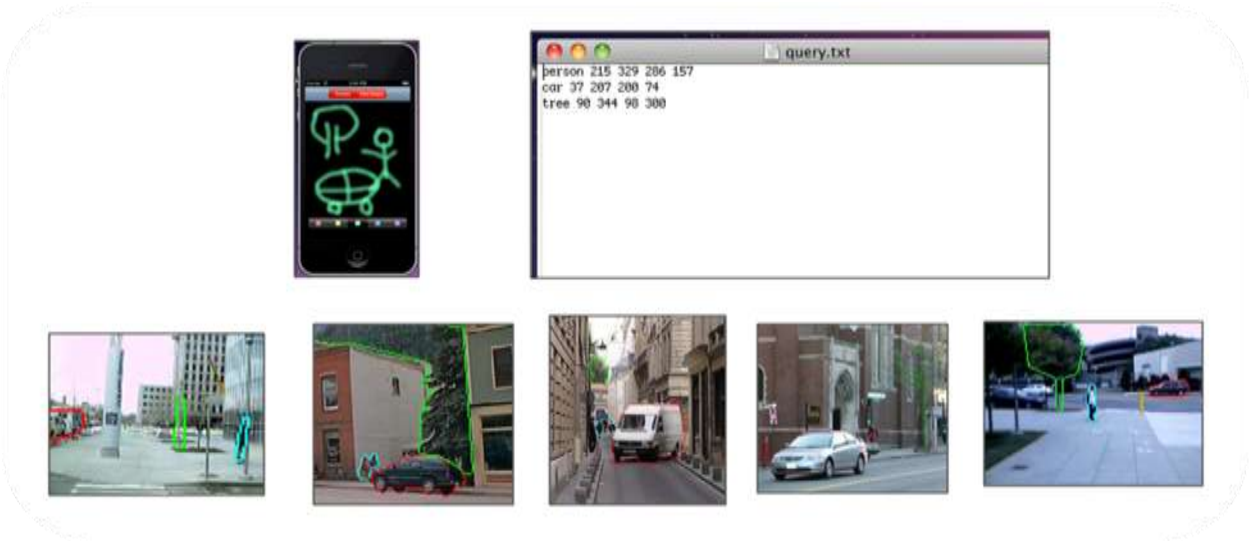
It was seen that the highest false negatives were for the bicycle, owing to the varying distances between its front and back wheels in various drawings. The highest false positives were for tree, since the model is very sparse, and therefore bound to false-positives. Building, car and person are relatively easy to recognize, with the models we used.

When the app was demonstrated on the ECE Day, majority of the failures in recognition resulted from the unfamiliarity of the users to the object models and the consequent inability to draw 'good' sketches.



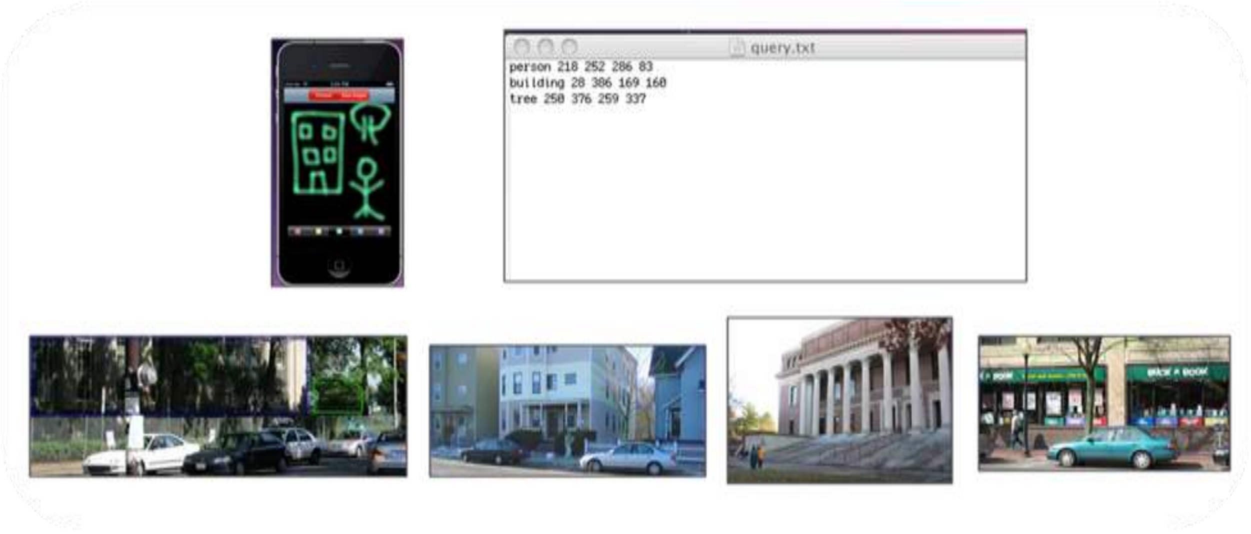
```
bicycle 56 207 247 85
person 141 361 247 85
```

Fig 7a. Example images for bicycle and man



```
person 215 329 286 157
car 37 207 200 74
tree 90 344 98 300
```

Fig 7b. Example images for tree, car, person sketch



```
person 218 252 286 83
building 28 386 169 160
tree 250 376 259 337
```

Fig 7c. Example images for building, tree, person sketch

Chapter 6

Further Work

The Sketch-2-Image app is a good start to the 'searching images using sketches' problem. The logical next steps for this app would be – expanding the set of object models so that the application becomes more widely applicable, and making the search algorithm more efficient so that the user can view images quicker. It currently takes about 37 seconds for three objects in the sketch to upto 4 minutes for five objects, which does not go well with the end-users. Increasing the number of object models might also make the app slower, since comparison of every model with the sketch takes time. A faster sketch-recognition algorithm will therefore be useful to keep the app fast enough. Intelligent implementation of the algorithms is also essential to manage the speed of the app.

Another limitation of the current algorithm is that the scale of objects in the searched images is not maintained properly. e.g. very small objects which are annotated as the <object class> are identified as similar images, which is not desirable. Making the algorithm scalar invariable is an important next step. Similarly, even though there are multiple objects of the same kind in the query, the server algorithm cannot handle them in the similar images. It can only handle one object per object class.

A paradigm shift that would make the app more realistic is moving from the LabelMe database of images to a wider database, like Google images, which are not annotated. In that case, use of object detectors for detecting objects in the images and then using the objects for the algorithm is a possible approach.

References

[1] Amir Sadovnik, Tsuhan Chen, ‘Pictorial Structures for Object-recognition and part-labeling in Drawings’, ICIP 2011

[2] E. Di Sciascio, M. Mongiello, F. M. Donini, L. Allegretti, ‘Retrieval by Spatial Similarity: An Algorithm and a Comparative Evaluation’, Pattern Recognition Letters, vol 25, Issue 14, Pgs 1637-1639, 2004

[3] Tao Chen, Ming-Ming Cheng, Ping Tan, Ariel Shamir, Shi-Min Hu, ‘Sketch2Photo’, ACM SIGGRAPH ASIA 2009, ACM Transactions on Graphics

[4] Pedro F. Felzenszwalb and Daniel P. Huttenlocher, ‘Pictorial structures for object recognition,’ International Journal of Computer Vision, vol. 61, pp. 55–79, 2005

[5] LabelMe : http://labelme.csail.mit.edu/

[6] GL Paint - http://developer.apple.com/library/ios/#samplecode/GLPaint/Introduction/Intro.html