Arts Quad Tour Guide

Team: Eric Sample & Dan Gallagher

Abstract:

As mobile devices become more and more powerful and more sophisticated, there is a big push to create algorithms and programs that can take advantage of them. One of the largest areas of research for utilizing mobile devices is “augmented reality,” the use of mobile technology to quickly provide information about a user’s surroundings. Currently, many of these mobile devices have a host of useful sensors on them that aid in implementing these augmented reality applications. This project seeks to explore the use of those sensors in creating a hand held tour guide for the Cornell Arts Quad, a collection of buildings in the center of campus. We use computer vision algorithms to extract identifying features from images taken by a smart phone’s camera, labeling the dominant building.

Related Work:

Augmented reality is a very hot research area because we now have hardware that can achieve useful results. The main idea is that we can use technology to make our experience of the world easier or more enjoyable. Some current implementations include heads up displays in cockpits for pilots, image overlay to help surgeons, and they've even been used to help blind people navigate. Some of the research is in the area of testing out what user interface designs will work best for a mobile user. Most of the useful aspects of augmented reality are dependent on determining where the user is located. This has become much easier in the since GPS devices have become so inexpensive they are put into most mobile phones. Having the position data from GPS allows the augmented reality systems to easily find what objects are near the user, and can easily prompt the user with that information. Unfortunately there are many times when GPS is not available or is very inaccurate. In these situations, we need to rely on other forms of sensors such as accelerometers, magnetometers (compasses) and cameras. This is the major purpose of this project: to test augmented reality systems in locations where precise GPS is not possible.

Implementation:

The major design goal of this project was to provide an application that could be used as a stand in for a personal tour guide. It would provide a path for the user to follow, and whenever the user came upon a building they were interested in finding out more about, they could point the camera at the building to get information about it. To provide this information, we needed to build a database that could be used to identify the buildings first, then provide contextual information about the identified building. To build this database we chose to use the Maximally Stable Extremal Regions features described in the previous section. These features are very similar to SIFT features in that the descriptor for them is computed using the SIFT descriptor pipeline, but they are identified through a different method. We tried to use the OpenCV implementation of the MSER extractor, with implementing the rest of the SIFT pipeline to handle turning them into descriptors, but for some reason the MSER we had a lot of trouble getting them the work well, so instead we moved to using the SURF feature implementation in OpenCV.

Once the features were extracted, we also had to go through each database image to label the buildings in the images. This was accomplished by having a user specify a bounding box for whatever building was in the image and identify the name of the building that bounding box corresponded to. This information was stored in a separate label file that the server would read in at start up as part of building the index.

For the implementation of the actual application for the user, our first design decision was to use Android based phones because we both have one. This provided us with the ability to test our algorithms and implementation throughout the design of the program. The second major design decision was informed by the restricted computing power mobile devices have. Instead, we opted to use a client/server model where the phone sends the current image and sensor data to the server for processing. As a result, we were able to us to use more powerful algorithms while still maintaining a responsive UI for the user. When the server had finished its processing, it sent a reply containing the augmented reality information for the building it identified inside of the image. The content of that reply was as follows: a short description of the building; a URL to the homepage of the building; a URL to Cornell's search page of that building; a URL to the Cornell facilities webpage for that building; the URL to the Wikipedia page for the building; a URL to the Google search for that building's name; and finally a character indicating which direction the user should continue if they wish to follow the tour. The rest of this section is split up into a description of the server and a description of the phone client.

Server

When it starts, the server builds the vocabulary tree of descriptors using the descriptor and label files. The label define labeled building rectangles and are labeled such that they can be matched with descriptor files. The server receives requests containing images, extracts the descriptors for the query image, and finds matching labelled descriptors in the vocabulary tree. Images to build the database were those provided by Professor Snavely. In each of these images, a human created a rectangle roughly containing each building in the image and labeled each rectangle. Any descriptor found within a rectangle is labeled according to the rectangle's label by the server.

The vocabulary tree is built by recursively dividing the descriptors into clusters according to their feature vectors. For each node, the descriptors in the node are clustered using k-means. Each cluster is than assigned to a child of the node. This process continues until the maximum specified depth is reached. Once the tree is completed, all descriptors are stored in the child nodes.

Descriptors are matched by traversing the tree and finding the closest labeled descriptors. At each node (starting from the root), the closest cluster of the clusters defined by the child nodes is determined. Distance is determined by the Euclidean distance from the descriptor to the cluster center. This process is repeated recursively on the closest child node until a leaf node is reached. The descriptors in the leaf node are then used for classification.

Originally, feature classification was done according to a simple plurality vote function. A descriptor was classified according to the plurality label in the leaf node to which it was

matched. The image was then classified as the plurality of the label of the matched descriptors. This process, however, did not yield good results for two reasons: 1) the votes were not normalized according to the number of descriptors of each label in the tree and 2) ambiguous leaf nodes (nodes with many different descriptors) were given equal weight as leaf nodes with a strong majority.

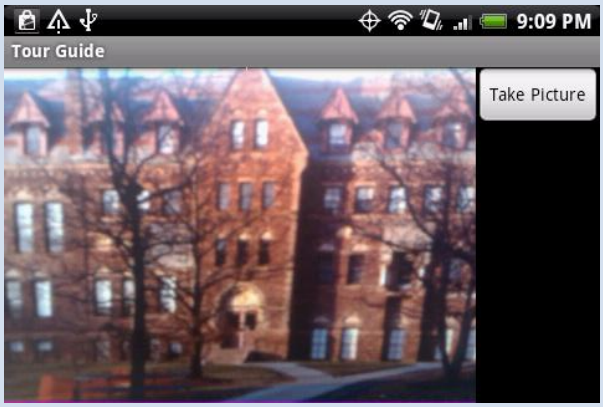
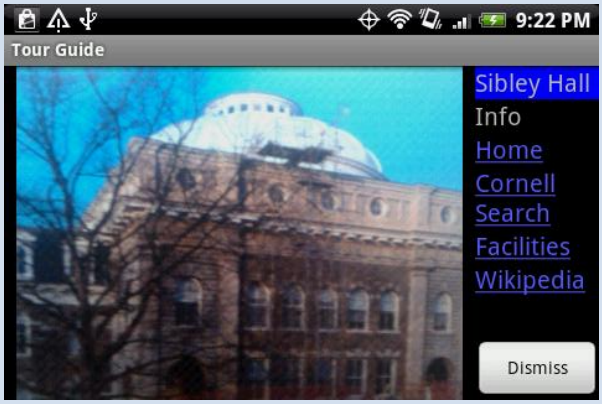
To remedy these issues, proportional, weighted scores were introduced. Scores are weighted by their entropy, so weights are defined as:

$$w_i = ln\frac{N}{N_i}$$

where N is the number of labels in the database and N_i is the number of distinct labels in the leaf node. For each label in the leaf node, the vote of that label is $n*w_i$ where n is the number of occurances of that label in the leaf node. To label an image, votes are aggregated for each descriptor. Then the total votes for a label is divided by the total votes for all labels to get the proportional vote for that label. The proportion of the weighted votes for that each label in the tree is subtracted fromt the proportional vote for each label in the query image. This difference is called the score, and the image is labelled with the label of the highest score.

Client

The client interface has two basic states. The first state provides what is essentially a viewfinder for the user to determine exactly what building they would like to ask the server about. The second state is triggered when the user pushes the “Take Picture” button. When the button is triggered, the current image is sent to the server for processing, and a new view is displayed to the user. In this view the image that was sent to the server is displayed and the information provided by the server is accessible but selecting the appropriate text on the right side of the screen. The “Info” link pops up a dialog with the text of the description. The rest of the links open up a browser that navigates to the associated URL.

Viewfinder State	Information State
	

Discussion:

As mentioned above, we tried to use the MSER descriptors to perform the feature matching, but had problems. The first problem was a limitation of the Android platform. Android OS makes it somewhat difficult to get a hold of the raw image, mostly due to memory constraints. The live preview that we used for the viewfinder is a scaled down image that could only effectively go to a resolution of 640 by 480 pixels. Grabbing these scaled down images was the best way to efficiently get a hold of the image data because causing a picture to be taken was a relatively lengthy expensive process that only returned a JPEG, not the most convenient format for us at the time because we would need to somehow build an OpenCV image from that data. The small size of the images being sent to the server caused a problem for the MSER feature extractor because it could not find enough features to match to other images. Unfortunately, even when we used larger images with the server, the features were still not good enough to match to the correct building. As a result, we decided to move to using SURF features……..

Despite adding proportional comparison and entropy weighting to the feature matching, the accuracy of the building classification was still relatively low. Accuracy was affected heavily by the size of the tree, but we only explored a few possible tree sizes. It is likely that with more descriptive features for database images and a tree sized determined using machine learning, the classification accuracy would be much better. In the end we only managed to get approximately 20% correct identification, which means there is plenty of room for future work.

Future Work:

The main things we would have liked to accomplish, but didn't have the time, involved drawing information into the image. The first one was drawing a bounding box around the identified building as an aid to the user. This would not be too hard to accomplish except we would need to keep track of some spatial information or compute a homography in order to identify the points where the bounding box would go. Another feature that would be much more in line with the augmented reality thrust of this project would be to draw the tour path into the viewfinder preview to help the user follow the path more easily. This also would require finding a projective homography, although it might be a little bit simpler than finding the homography between two images. It would be highly dependent on getting good localization data for the user, which would be a project in and of itself.