Determining Building Location Based on an Image

Carter Peterson and Morgan Ludtke
University of Wisconsin-Madison

Abstract

Keeping track of where a photo was taken is becoming easier every year as the use of smartphones to take photos increases. GPS systems within phones can geotag the photo and store that information along with the photo. Although it is an ever improving field, there is always room for improvement and one of those places is inside buildings where GPS systems abilities decrease. Rather than relying on the GPS systems entirely, it is possible to create a geotag for an image by guessing its location based on the image characteristics. In this paper we propose an algorithm to guess what building an image was taken in purely by computing its characteristics and comparing them to a pre established database of already tagged images. Our easily enhanceable methods generated promising initial results that proved significantly better than random guessing. We show that similar methods are a viable option to improve the way we geotag photos.

Introduction

In the last handful of years photography has become less an activity that needs to be planned for as a large percentage of people now carry around a smartphone in their pocket. Photos taken with these devices not only dramatically outnumber photos being taken with traditional cameras, they also have the ability to incorporate other sensors on these devices to allow for extra data, such as location via GPS, to be added to the metadata of these photos. Relying solely on GPS to provide the location data for these photos can be unreliable and forces the devices to consume additional power to utilize the GPS chip to geotag these photos. One circumstance where GPS is virtually unusable is indoors. Our

method aims to improve the reliability of geotagging these indoor photos using image analysis instead of relying on GPS.

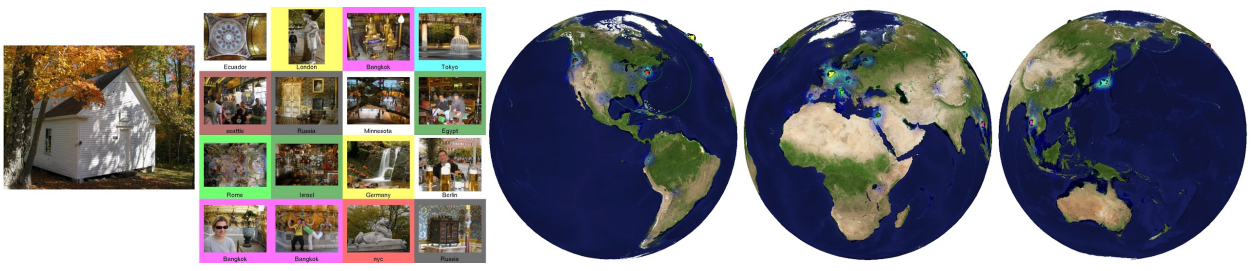
Motivation

Geotagging is “tagging” an image with it’s geographical location. This tag keeps track of where the image was taken. Often this is done by collecting the GPS information for the time that the photo was taken. To do this a GPS device must be available on the device that took the image. Because of this the most commonly geotagged images are ones that are taken on smartphones which have a built in GPS. We were motivated to do this project because there are situations that you don’t always have access to GPS, but it would be easy to determine where you were based on the image’s characteristics. One of these locations is indoor buildings because that is an area GPS abilities on a smartphone reduce in quality. We figured we could make a geotagging application that would work for buildings on the UW-Madison campus.

Related Work

The article IM2GPS [1] explores the possibilities of geotagging a photo based on characteristics within the photo. They used 6 different characteristics of a photo and compared them to those characteristics in a database of photos with already known locations. These techniques are Tiny Images, Color Histograms, Texon Histogram, Gist Descriptor, and Geometric context. All of these techniques ran their own calculations on the image to select the potential location of the image. Some of the techniques worked better than others, but combined together they created a better output than they would individually in sort of a weak classifier method.

IM2GPS Result Visualization [2]



Their program differed significantly from ours in that it was intended to determine your location on the Earth based on the image analyzed. The intentions of their work was more to get an idea of the general region that a photo was taken in and less to determine the exact location of the photo. Our approach differs from this in that we are attempting to determine an indoor location based off of a more select set of images.

Theory

Photographs taken of interior locations have many characteristics that aid in determining which building the photo was taken in. Color is one of the most obvious feature of an image. By extracting their color, an assortment of information can be calculated. For example a lighter image will have higher RGB color values and a darker image will have lower RGB values. Another more complicated feature than can be taken from an image is the amount of lines that it holds. An image can have a lot of lines, be that horizontal, vertical, or a mix of lines. An image with more lines will most often have a lot of distinct features such as walls or windows, or maybe even a lot of bricks. Another, more brute force approach, is to directly compare an image to another and be able to tell if they are similar. The best way to do this is to shrink down the images before you directly compare them because the pixilated image will be more closely related.

Methods

Our method for determining the location of a building relied on four image characteristic identifiers: average color, most frequent colors, line energy, and tiny images. These characteristics were precomputed on a set of images to create a database and input images were tested against this database to determine the location of the image. Before all characteristics of the image were computed, the image was first resized to 640x480 pixels. This allowed our classification methods to obtain enough data to

accurately compute the characteristics while significantly reducing the run time of the algorithm when compared to using original images extracted from the camera.

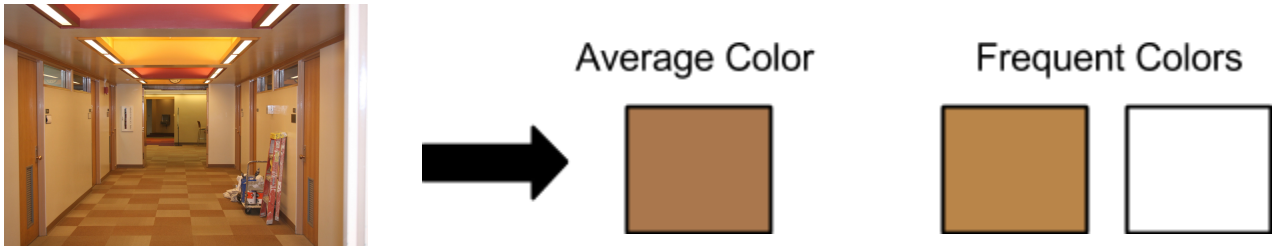
Average Color

Our average color classifier computed the overall average color for the all pixels contained in the photo. This average was computed in the RGB color space using a pixel-wise averaging method. To compute the average the Red, Green, and Blue values for every pixel were summed into an independent overall color values. These values were then divided by the amount of pixels in the image (which in our case was always 640x480 = 307200) to obtain the average color for each channel. The RGB color channels were then combined into a single sRGB, mostly for encapsulation to allow easy transportation of the data through the program

Most frequent colors

Many buildings have interiors with a limited, but distinct, color palette. Often images from within a specific building have large amounts of pixels with the same color. By determining the N most frequent colors seen in a photo (in our case five colors were used), we can directly compare the color palette of our input image to the images of the buildings in our database

To define the most frequent colors in the image our method started with creating a histogram for the colors of every pixel in the photo. From this histogram it can then determine which color values were used most frequently in the scene. We faced an issue in that small inconsistencies in lighting or texture in the image can cause a wall of continuous color to be detected as multiple colors. To combat this our final list of the most frequent colors only included colors that were noticeably distinct from one another. After the N most frequent colors are produced they are returned as an array of sRGB color values.

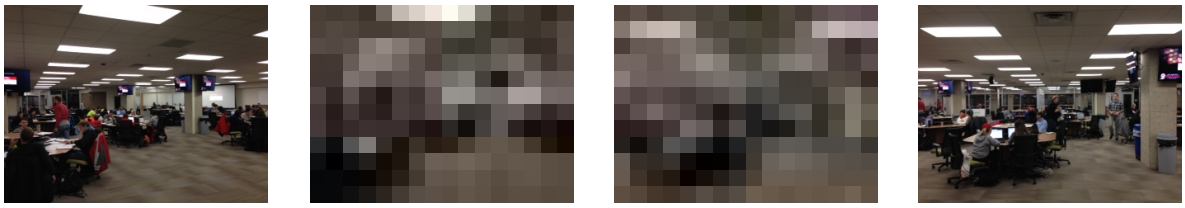


Results of the Average and Most Frequent Color Classifiers

Tiny Images

Many images of the interior of a building will have a similar overall structure to one another, but pixel-wise comparison of these images will yield a wildly unpredictable result that usually can not be used for classification at all. To allow the classification

of the structure and color of an image our algorithm used images that were dramatically reduced in size from the original image (in our case a 16x12 pixel image is used).



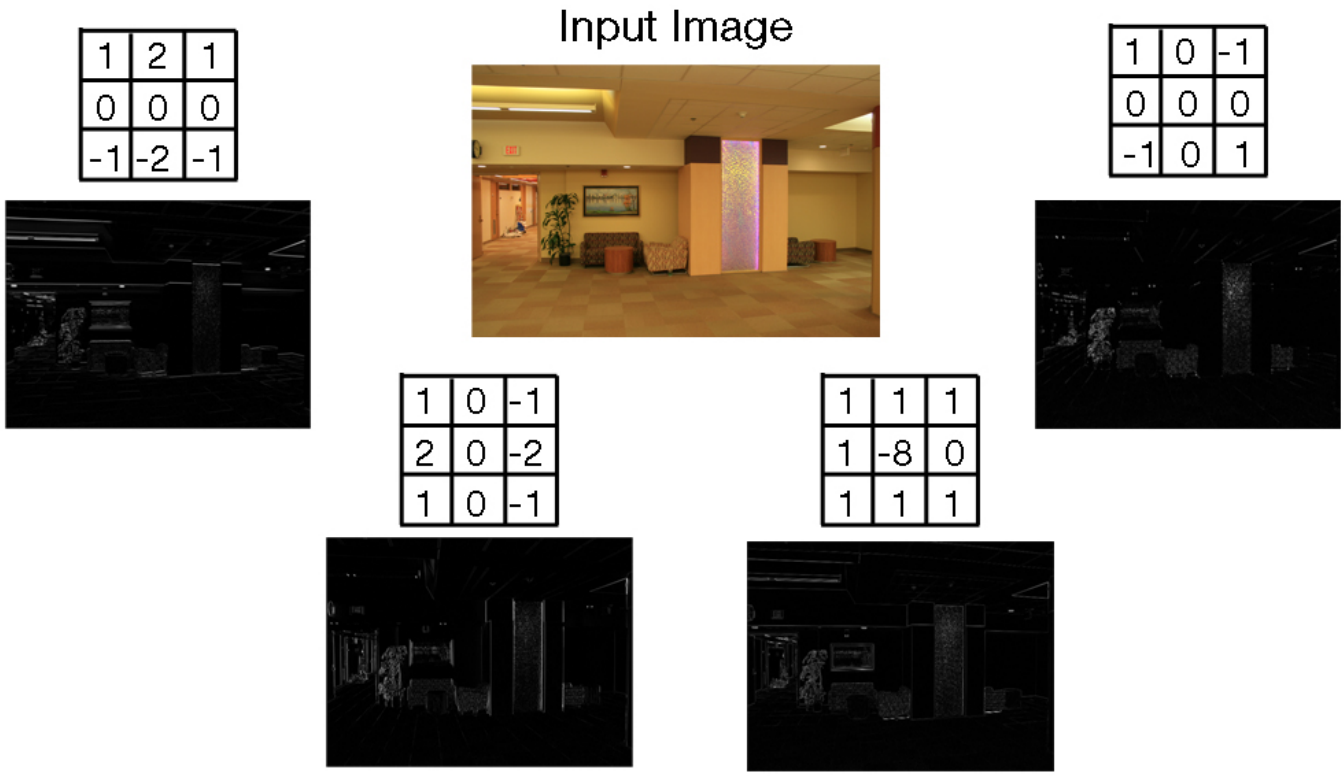
Two images of Wendt Library and their tiny image counterparts.

This method of tiny images allows for a direct comparison of the color space of the image to also incorporate location within the image. This can be useful in identifying consistent themes within the building such as wall color, overhead lighting, and large objects placed within the scene. This classification allows holistic analysis to be performed on the scene using minimal computation power and memory.

Line energy

Interior photographs will differ dramatically in the amount of implied lines that they have. A photograph containing a brick wall will contain dramatically more lines than a photograph containing a

smooth wall of one color. This “lineiness” can be calculated and used as a classifier to decide which building a photograph was taken in.



Input image with applied kernels and gradient outputs.

To estimate the line features of an image we must first convert the image to grayscale. After adding a 1 pixel padding around the image using reflection, we are able to iterate a 3 pixel by 3 pixel kernel over each pixel in the image to calculate the gradient image for the applied kernel. We computed 4 gradient images with unique kernels: vertical edge detection, horizontal edge detection, Laplacian of Gaussian, and diagonal line detection. We then performed a pixel-wise summation of the energy in these gradient images to obtain an overall “lineiness” for each kernel. These “lineiness” numbers were rounded to the nearest whole number and stored in an array.

Populating the Database

By using these 4 calculation methods, we were able to pass in all of the photos for our database and create .txt files to function as database files for the four classifiers. This allowed there to be a precomputed database of classifiers with associated locations to reference when running the algorithm on an input image from one of the building in our database.

We collected 20 to 30 photos for each of five buildings we chose to do our project on. These locations were the Computer Science building, Engineering Hall, Wendt Library, Bascom Hall, and Union South. Along with these photos we also collected photos from those locations that we didn't put into the database so we can have some test photos.

Running the Algorithm

To run the algorithm on a given input image we first rescale the image to 640x480 pixels. This allows for normalization across elements that perform pixel-wise operations, and allows the classifiers to run in a more reasonable amount of time. After rescaling our four classifiers are run on the image, creating the same data that each photograph in the database has. This classifier data is then fed into matching functions for each classifier.

These matching functions perform Sum of Squared Difference of the classifier data to independently compute which images in the database match the best. The classifier matching functions then return the location of the N (in our case three) photographs that matched our input photo the best. The locations returned are in order of which locations matched the best, and are given a weight depending on their rank. To decide the final result we sum the assigned weights as "votes", using the location with the most "votes" as the estimated location.

Results

We ran our algorithm over photos of the interiors of the buildings that we had built our database

on. These photos were much more varied than the photos we used to calculated the database. The time of day of these photos were taken included the time when we took the database photos as well as random other points throughout the days surrounding. These photos were also taken from multiple cameras whereas the database photos were all taken from the same camera. Despite these differences our algorithm was able to correctly identify the building ~70% of the time. While this isn't quite the accuracy we were hoping for it is significantly better than the 20% that would be achieved through random guessing.

While the training and test set were slightly limited in that they only documented five building around campus we were able to notice some trends in the results that were produced. First, as it stands this algorithm is very lighting/white balance dependent. As many of the classifiers that we used to determine location had some dependence on color the lighting and white balance of the photo can significantly alter the overall hue of the photo, degrading performance with our algorithm.

Two photos of similar hallways in Engineering Hall.
Left: A blurry image with a different white balance taken with a point and shoot (Poor Results)
Right: A photo taken with an Apple iPhone (Good Results)

Second, uniform buildings perform very well with this algorithm. This could also be predicted as the algorithm references other images of the same buildings, but the algorithm perform very well when

the building had a lot of uniformity in the color space or in the "lineiness" of the building. This helped the most when using the same camera that was used to populate that database as the color space in the photo being analyzed would match the database color space closely.

Due to the heavy dependence on color space recreation in our algorithms the ideal application of the algorithm would be use in relatively uniform cameras, such as in the Apple iPhone. One area that could be explored further would be a white balance normalization process. This could bring the accuracy of the algorithm up when using different cameras and would also help to correct for slight discrepancies in the lighting.

Future Work

We only used four different types of characteristics for the photos. The program could be enhanced (and potentially become more accurate) with more characteristics computed. One possible feature to add would be a GIST descriptor which detects objects in the image. For example it could detect large structures that would be similar in images taken of the same thing. This would be very useful because it could find similar scenes in two images. However, GIST is fairly hard to implement. There are already established GIST descriptor codes floating around, but, none of them are in java. We used java for our entire program and our translation of GIST to java gave poor results. There are Matlab and C code for GIST descriptors and are free to download which could be added fairly easily. Another possible addition to the program could be texture histograms. This would enhance the program because many buildings have distinct textures and some buildings can be picked out of a group solely on textures. Along with those there are many more image descriptors that could be implemented to further enhance our program.

Lastly, the results would improve with the increase of photos to the database. The more

pictures taken within a building will increase the chance of matches to the correct building. For our project we only used about 20 to 30 pictures per location, just by doubling that number would greatly improve the output. One possibility to increase the database would be to add the calculated picture to the database if the correct location is selected. So in theory every time the program is run the accuracy improves.

Referenced Work

[1] James Hays, Alexei A. Efros. IM2GPS: estimating geographic information from a single image. Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2008.

[2] IM2GPS Visualization. http://graphics.cs.cmu.edu/projects/im2gps/results/0112_Vermont_00002_113886360_6cb9ab47bd_41_87434398@N00.jpg

Additional Info

This project was written entirely in Java using no external libraries. It was written in ~1100 lines of code with one method that converts an Image to a BufferedImage found at http://blog.pengoworks.com/index.cfm/2008/2/8/The-nightmares-of-getting-images-from-the-Mac-OS-X-clipboard-using-Java.

Morgan wrote all of the code that dealt with classifying and comparing the average color and most frequent colors.

Carter wrote all of the code that dealt with classifying and comparing the “lineiness” and tiny images.

We plan to continue work on this method and you can find current info on this project at: http://pages.cs.wisc.edu/cpeterso