CS51 Final Project: Photographing the John Harvard Statue

Final Report

Amy Kang, Anna Liu, Haoqing Wang, Cecilia Zhou

amykang@college.harvard.edu, annaliu@college.harvard.edu
haoqingwang@college.harvard.edu, zilingzhou@college.harvard.edu

May 1, 2015

I. Project Overview

Our Demo Video: https://www.youtube.com/watch?v=40RwYGdMLWk

Motivation for our project arose from the difficulty of acquiring quality photographs of popular tourist attractions, such as the John Harvard Statue. Heavy traffic, whether composed of tourists, locals, or pigeons, seems to ruin every picture taken. Our solution to this problem is to take multiple images of the same location, each with its own set of noise. Then, given this input of images with noise, or "tourists," we will remove the noise to produce the desired image, sans traffic.

We accomplished this first by reading the RGB color values at the same position within each image of the set. We then took the medians of these RGB values through median filtering and used those median RGB values to produce the pixel in that position within the final image. Given that there are more images in which there is no noise covering this pixel than those in which there is, the median should return the RGB values of the target subject, not the noise. We find the median through a randomized algorithm based on QuickSelect, with O(n) runtime. We also implemented median filtering through counting sort, which is especially effective for fixed maximum values. Because RGB values are limited to the range 0 to 255, our data is a good candidate for counting sort. We also implemented a deterministic algorithm which has an expected runtime of O(n).

If given a video rather than a set of images, we also created a method to produce a noiseless image from the frames of the video in order to improve both the convenience and accuracy of our algorithm. We used OpenCV's VideoCapture feature to read in still frames of the video. We then converted each frame to an Image object using Python Imaging Library. Once we accumulated

our set of images from the still frames, we ran our median algorithms on these frames to produce an output image.

We evaluate our software by determining how similar our output image is to an image we took of the scene with no noise. We do this using three methods: a pixel by pixel comparison method and color histogram methods using correlation tests and chi-squared tests. For the pixel by pixel comparison method and the correlation test, a similarity value close to 1 indicates high similarity whereas for the chi-squared test, a p-value close to 0 indicates high similarity. Although the pixel by pixel comparison produces a good representation of similarity for our purposes (i.e. there is no change in frame between the control and the output image), it fails with any small shift, translation, or rotation between the images due to displacement of RGB values. Even the same picture then, when one image is shifted 2 pixels to the right of the other, would not be considered similar when they should be. In order to account for this, we create color histograms of the output and control images and compare the color distribution across the two images. Color histograms are widely considered as a sufficient signature, or "thumbprint," or images. We compare these histograms using two separate statistical methods, the chi-squared test and correlation.

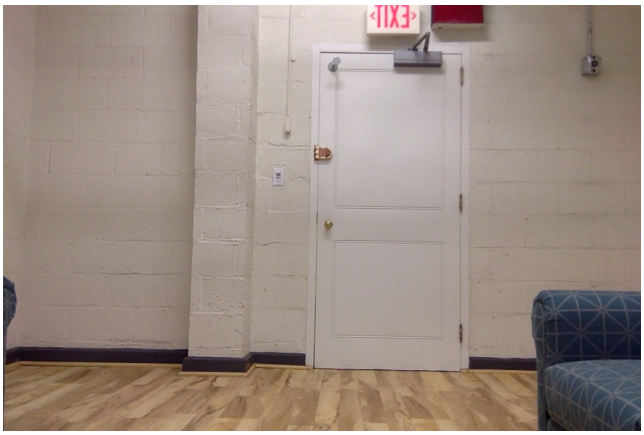
II. Instructions

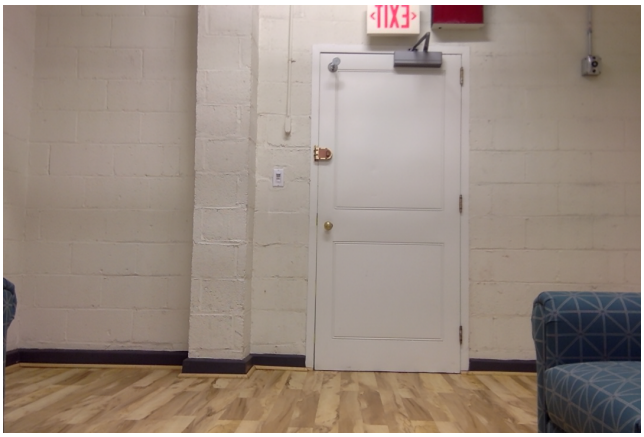
Please see the code/README file within the project folder for instructions to run our project.

III. Results

We've reproduced our results below along with similarity measures with the control photo to indicate the success of various methods and the variability in similarity calculations.

Successful Example! (consistent frame)





Amy in Room

	Brute	Correlation Similarity	Chi Squared p-value
Quick Select	0.74	0.99	7.455e-121
Median of Fives	0.81	0.99	1.651e-138
Counting Sort	0.75	0.99	7.455e-121
Built-in Algorithm	0.76	0.99	4.883e-122

Less successful example, but artsy! (inconsistent frame)

John Harvard Statue

	Brute	Correlation Similarity	Chi Squared p-value
Quick Select	0.38	0.99	0.00
Median of Fives	0.36	0.99	0.00
Counting Sort	0.38	0.99	0.00
Built-in Algorithm	0.38	0.99	0.00

IV. Report

Milestones and Timeline

We successfully implemented the core functions in the time that we expected. We utilized the Python Imaging Library (PIL) to process the images, which allowed us to efficiently and conveniently iterate through the photos and grab the median values without storing all of the pixel values into arrays. We were able to create a noiseless photo from a series of photos taken from the same frame with our various median algorithms. We also implemented a large portion of the cool extensions we included in our specification. Because it is easier for the user to maintain a stable frame while taking video than while taking multiple pictures and to upload a video file rather than uploading multiple image files, we implemented a method to generate a noiseless photo from a video rather than a set of images by applying our algorithm on the frames of the video. We also wrote several algorithms to compute image similarity to compare the noiseless photo we create with our algorithm with a control photo taken without noise (or tourists).

Testing was relatively simple once we took sample photos and videos since we just had to run the program and see the resulting picture. Although we expected that python would be an easy language to pick up, it was difficult at first to get used to the syntax; for example, rather than curly braces or other indicators, python uses indentation to determine scope.

Our choice to implement our project in Python served us well. The Python Imaging Library (PIL) made retrieving the RGB values and size of pictures in various formats fairly simple. Integration of Python with the package OpenCV also simplified the implementation of our project with video. OpenCV's VideoCapture object simplified the retrieval of image frames from video while converting the color format of each image to RGB; however, OpenCV has a very steep learning curve, so it was very difficult to figure out what aspects of it we needed and know whether a function was already available in OpenCV.

We also found that PIL made it very convenient to work with images. However, we also realized that when we were attempting to create a noiseless photo from photos that weren't taken from the same angle, we obtained results that appeared similar to impressionist art, with noise removed but an extremely blurred background. In order to remedy this, we decided to implement an image or video stabilization algorithm (in stabilize.py), but discontinued our efforts. We began to implement an algorithm to stabilize a set of images by finding the best features to track in the current image and then by finding the transformation from the current image to the next using those features. We used a C++ implementation of a video stabilization algorithm using OpenCV (http://nghiaho.com/?p=2093) as a starting point to identify helpful functions within OpenCV

and to learn about how the relevant functions within OpenCV work. From here, we intended to adapt the algorithm to create a more robust stabilization taking in image sets rather than video, and accounting for not only rigid transformations (translations and rotations in the plane) as this algorithm does, but also shearing and scaling by using findHomography and warpAffine functions in OpenCV rather than estimateRigidTransform. We quickly found, however, that OpenCV, as well as methods to calculate optical flow and transforms have a very steep learning curve, and the time could be used to work on our other extensions instead. We decided to leave our attempts for now to move on to other extensions (video, similarity and image compression). We noticed that the image produced by our method was considerably larger than the images within the set used to produce it and decided that image compression would not only be an interesting algorithm to implement (we used Huffman Encoding to compress the file) but also a reasonable extension given the increase in size of the output image.

Future Works

We would have liked to create a front end, or user interface, to allow users to drag and drop image sets and download the output image. We would also have liked to explore OpenCV in more detail. Some of the sets of images were not completely in a single reference frame, so when we ran our algorithm, it resulted in a slightly blurred image (aka impressionist art). Therefore, we tried to implement a way to normalize the reference frame so that the final image wouldn't be blurred. We found that what we were trying to do is very similar to video stabilization, so we did a bit of reading on using OpenCV for video stabilization. Since OpenCV has such a steep learning curve, we weren't able to completely understand many of its functions to implement our idea from scratch. Instead, we tried to adapt code used for video stabilization to achieve our own goal of creating a clearer image with our algorithm. However, we weren't able to make that work either, since existing code was all in C++, and it was hard to reconcile the differences between C++ and python and at the same time figure out how to deal with the differences between video stabilization and what we were trying to do.

If we were to begin our project from the beginning, we would definitely spend more time reading documentation of OpenCV and also potentially explore other useful Python libraries so that we have a good grasp of our tool belt. In particular, if we understood OpenCV better, we could use it to "stabilize" our images, so if someone were to take lots of photos that are slightly different reference frames from each other, we would still be able to run our noise-removing algorithm and create a clear, noiseless photo. We found that taking the time to familiarize ourselves with available libraries and functions is very valuable, especially considering that we ended up writing code that was no longer needed given the available functions.

Contributions

Amy:

- Wrote portions of the Picture class, located in picture.py
- Researched and implemented histogram similarity algorithm using chi-squared test
- Implemented histogram similarity algorithm using correlation and OpenCV
- Implemented method to allow video input rather than image directory
- Began researching and implementing image stabilization
- Took test pictures and video
- Contributed to the final write-up

Anna:

- Wrote basic implementation of main.py and some parts of picture.py.
- Created basic UI through terminal in main.py
- Implemented brute force image similarity algorithm.
- Took pictures of John Harvard.
- Worked on testing similarity and median algorithms.
- Wrote README file.
- Contributed to the final write-up

Cecilia:

- Created and edited demo video.
- Wrote initial implementation of QuickSelect algorithm.
- Took test pictures and video.
- Began researching and implementing image stabilization.
- Researched statistical methods for chi-squared test and correlation
- Contributed to the final write-up

Haoqing:

- Researched median selection algorithms in terms of theoretical efficiency as well as practical speed for the size of inputs we were expecting.
- Edited and fixed some errors that were present in the QuickSelect algorithm.
- Wrote "median of fives" algorithm.
- Wrote counting sort algorithm.
- Implemented Huffman coding algorithm to find a symbol-by-symbol encoding for RGB values in order to compress an image.
- Contributed to the final write-up

Reflection

Teamwork and communication are key. Having a fresh pair of eyes look at code is really helpful for debugging. Using modularization to split up the work made it easier to fit in our work with our different schedules. We also learned the value of adaptation and adapting our previous specifications and milestones to better fit our goals and timelines; for example, we had originally intended to try comparing the different median algorithms and with mode, but later we decided it would be more interesting to work on using a video instead of a set of images.

We learned that anyone considering projects involving image or video manipulation should look into OpenCV as a first step. Understanding OpenCV was a big learning curve, so getting started early is essential. We also think structuring our timeline to have leeway for making mistakes or not being able to understand certain algorithms was useful, since we spent a lot of time implementing code that we later didn't use or decided was too difficult to continue. It's also important to get a prototype of the project working early on so we know it's viable, and then work on one extension at a time. If we had more time, we could implement more extensions, but if we ran into road bumps along the way, we would at least have a prototype working.

V. Previous Specifications

Our draft specification can be found here, and our final specification can be found here with annotations regarding what we were able to implement and include in our final submission.