Joshua Korn

Section F

4 December 2014

Project Proposal

In today’s world, people are always looking for the next big thing. Whether that be in the form of new discoveries in robotics, or new cell phone technology, the concept of something *new* is exciting, and shows the world that humans will never stop inventing. For my project, I am trying to do just that, on a smaller scale – something *new* that shows where we have come, and what is possible for the future. My project is partially a showcase of the Leap Motion hardware, using OCR technology to, quite literally, write things in the air, and have them appear on your computer. It provides a new, relatively quick method of interfacing with the computer as opposed to the simple keyboard-and-mouse setup. Along with the Leap is RGB parsing using OpenCV. By combining the two, you can do some pretty cool things – such as making a grocery scanner! The first thing you thought of, right?

As opposed manually entering produce as is done in supermarkets, entering codes and going through countless menus, you can use the Leap, OCR, and OpenCV to quickly recognize the item you have. While the primary application of this project is for quicker (and cooler) item scanning in food markets, this technology can be used to create a database for virtually any items. The only requirement is that the items have names and a color. You can, of course, add whatever items you want to the database, as well as increase samples of the same item to increase search accuracy. You can also browse through all of your existing items.

In order to make this project possible, I have had to use a few outside modules and technologies (besides TKinter/EventBasedAnimationClass). They are listed below:

* The Leap Motion, along with its Python SDK. Seeing as I am using the Leap Motion, this is necessary.
* Google’s open-source OCR program, Tesseract. Seeing as we have minimal time for this term project, and that OCR programs take teams of geniuses years to create, I am unable to actually create my own OCR. However, using mainly console commands and some handwriting fonts, I was able to train Tesseract to more accurately recognize handwriting than it does by default. This took many hours, and unfortunately is not reflected in the code; however, it is the purpose of the tessdata folder included in the zip file.
* Pytesser, albeit heavily modified. Pytesser is a nice little module that interfaces between Tesseract and Python using somewhat complicated command-line code that we haven’t been taught. However, I heavily modified the code (both my name and the original creator’s names are cited at the top of the file) to allow for the following:
  + Tesseract recognition of each input as a single character, which increases its accuracy with my application;
  + Custom languages, which in my case was the training data;
  + and whitelists to search only for letters (not punctuation or numerals) to reduce collisions.
* PIL, Python Imaging Library, used to create and save images from user input that are used for Tesseract, as well as converting image formats and reading RGB values from images.
* OpenCV to take RGB values from the webcam, used to identify items as well as take pictures to add to the database.