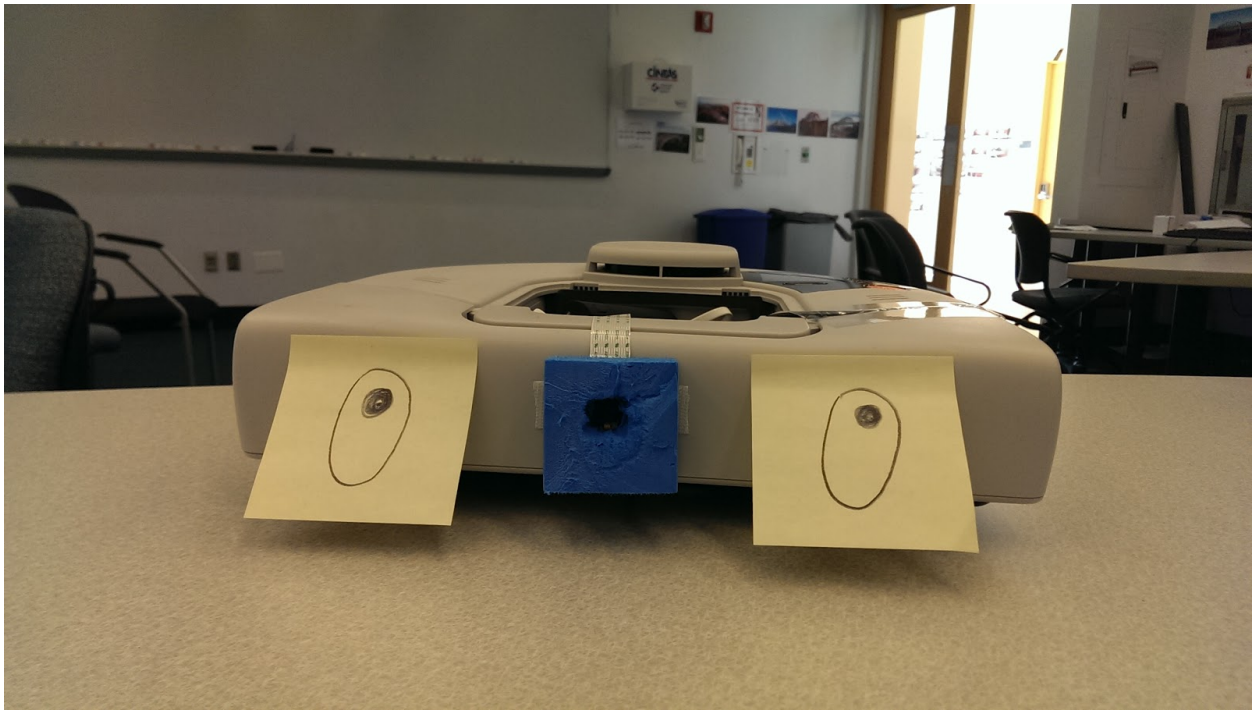
The Vision Project: Building a Smile-detecting Robot



By Adela Wee and Michelle Sit
Computational Robotics, Fall 2014, Section 1
November 10, 2014

Table of Contents

Executive Summary

Our overall goal was to learn more about how computer vision is used to classify objects, different methods of approaching that problem, and what coding processes are necessary to achieve that goal. We decided to explore face recognition and smile detection through OpenCV's libraries and made it a goal to implement smile detection through a computer camera's webcam. Our stretch goal was to use the camera feed from the neato to identify smiles and drive toward the user until it reached a specified distance if a smile was detected.

We began by reading images through a laptop webcam and running OpenCV's built in Haar Cascade face detector to detect if there were faces in the video feed. We then cropped the image to focus solely on the region of the person's face and ran a smile detector to determine if the person was smiling or not. If the person was smiling, we had the neato drive toward the person. If they were not, the neato would slowly back up.

Background

Algorithms Used

Open CV's Haar Cascade algorithm (aka. the Viola-Jones Method) is based on Haar-like features, which are digital image features used in object recognition. Haar features look at adjacent rectangular regions of an image, sum the pixel intensities in each region, and calculate the difference between these sums in order to categorize subsections of an image. For example, an image of a face will show that the region of the eyes is darker than the region of the cheeks, so a common haar feature would be the two adjacent angles that lie above the eye and the cheek region, whose position is defined relative to a bounding box that encloses

the target object, or face, in this case. Generally the system runs through a progression of loops that go from coarse to fine feature detection, and runs, or cascades, through images quickly if it doesn't find a face.

People's faces have easily identifiable features, such as the nose, eyes, forehead, chim, and mouth, so a color image of the face can be converted to greyscale and then this image can be overlaid with rectangles that enclose these areas and looks something like figure 1.

Figure 1. The haar cascade finds the average pixel intensity of specific regions in the face and is able to determine if faces and features exist in a specific image. (Image courtesy of eyalarubas.com)

There were cascade data sets for the Haar Cascades algorithm that existed in OpenCV, however, we couldn't determine the parameters for detecting smiles and went to build our own smile detector instead that took advantage of machine learning principles by using logistic regression.

Logistic regression is one of several ways for computers to learn functions $f : X \rightarrow Y$, especially for calculating probability $P(Y|X)$ where Y is discrete valued and $X = \{X_1 \ldots X_n\}$ is any vector containing variables we're interested in. If we assume that Y, the dependent variable, is binary, then the this algorithm is very convenient because it leads to a simple linear expression for classification, where the resulting relation for $P(Y|X)$ follows the form $Y = 1/(1+e^{-x})$ which looks like the graph in figure 2.

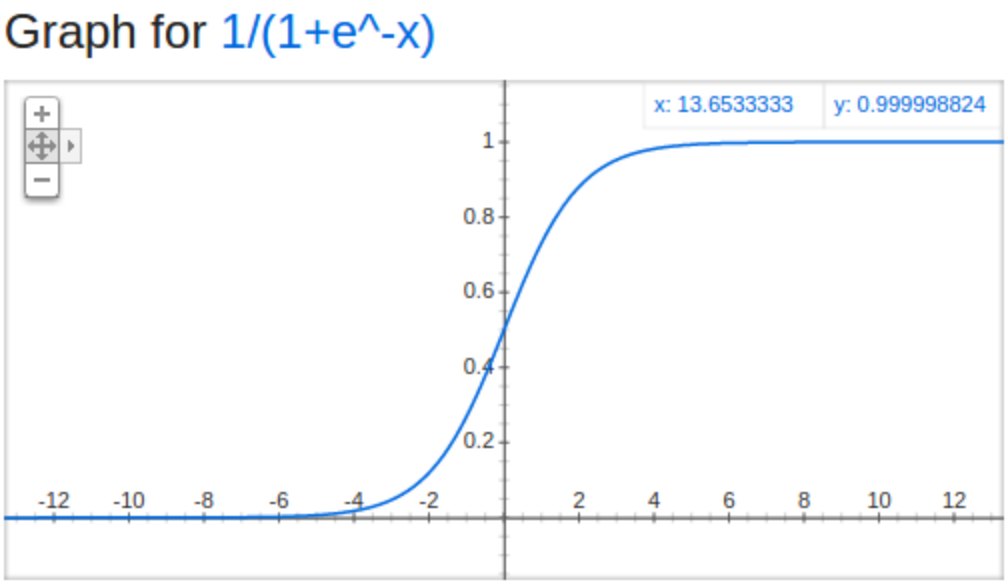


Figure 2. This a graph of a the linear relation given by a logistic regression.

Implementation

Step 1: Used classifiers available here: opencv/data/haarcascades/
Verified it worked by running code on a stock image as shown in figure 3.

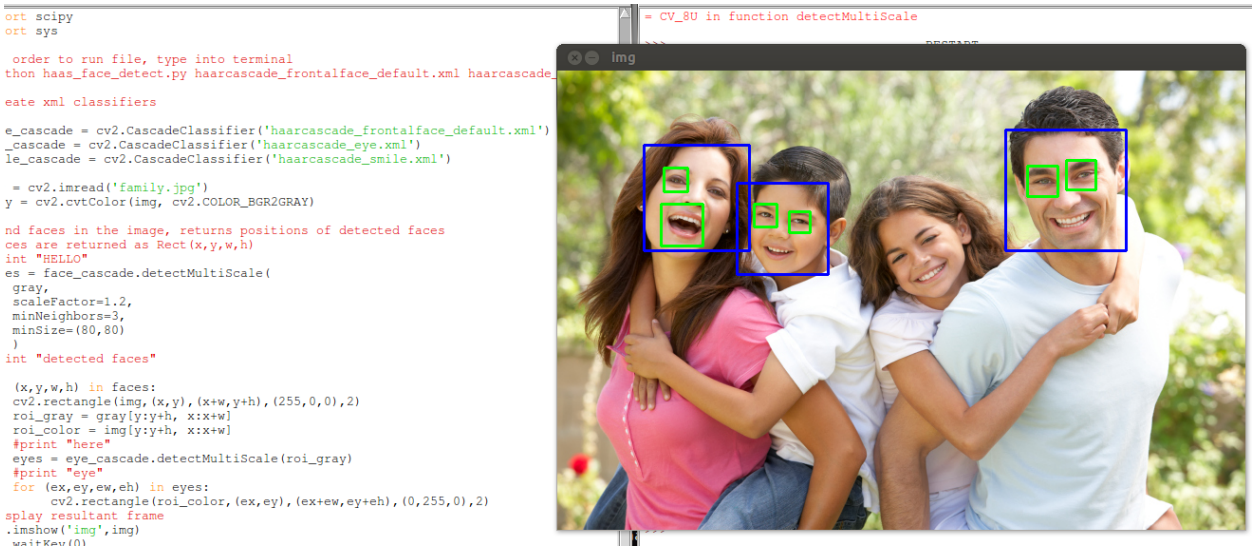


Figure 3. This is the base example for the Haar Cascade classifier that identifies faces and eyes. The file cannot detect faces that are not close to horizontal, and sometimes wrongly classifies facial features.

Step 2. Ran code on webcam feed. Example code from site was in C++, and a PoE team had figured out facial detection from a webcam and shared their python example with us. We verified that it worked, as shown in figure 4.

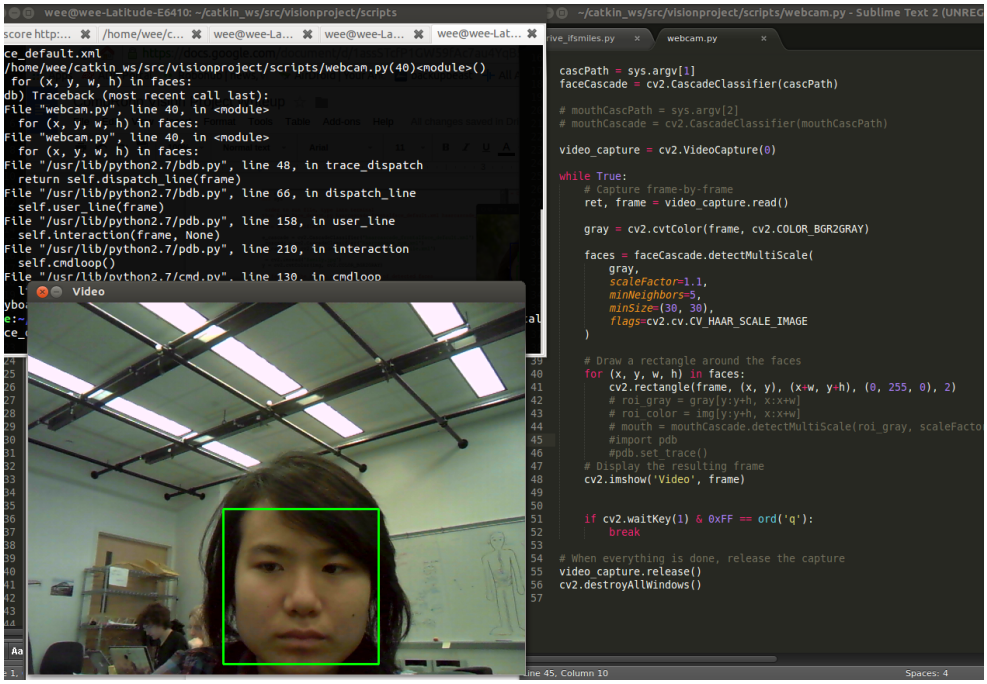


Figure 4. Face detection via laptop's webcam using OpenCV, so a still of facial detection from a video.

Step 3. We modified the code such that we didn't have to call the classifier's xml file every time and tried to add in the data to run the smile cascade. After many hours, we decided our time would be better spent creating our own classifier.

Step 4. Using the tagged database of images from UCSD, we were able to create a smile detector with sci-kit py that used logistic regression to determine if we were smiling or not. Our successful detector is shown in figure 6. It only runs in the greyscale image that has been cropped to the face.

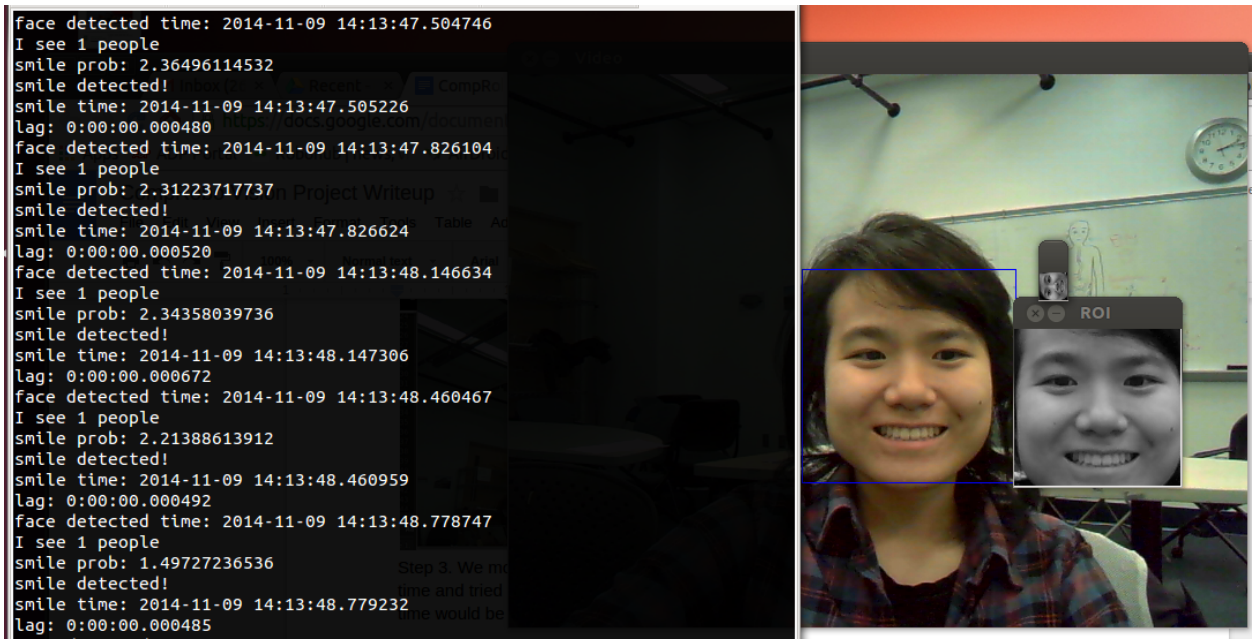


Figure 6. Smile detection algorithm works and runs only over the part of the image that a face was detected in. The tiny figure of the face shown is the image used to compare against the original database of faces.

Step 5. Tried to implement same thing but using the video feed of the neato. After spending considerable time on it, we still could not get it to work consistently so decided that we were going to use the webcam video to drive the robot forwards and backwards. Eventually we got the neato to respond successfully to the smile probability results. However the results were not always consistent. We believe that a lower resolution image from the neato caused the inconsistencies in responses.

Step 6. Implemented control of neato via smile detection from laptop webcam. It works fairly consistently but has a tendency to drive backwards more often than not. This might be because of our slower ROS update loop (in order to try to control the robot from going back and forth too often and impacting the motors). Future work would include better tuning of the smile probability values in order to better drive the robot.

Results

We were able to reach our goal of creating a smile detector that could identify smiles based off of the images from the webcam feed and have the neato respond. Although we were not able to get the neato to identify faces from the raspberry pi camera and had to use a computer webcam, we were able to implement a neato component in the end. In the end we have learned a considerable amount about computer vision, the different methods of detecting images, and how smile detectors work.

Reflection

Lots of downtime just spent working out python and openCV syntax. Biggest issue was with trying to find information on specific tools like Haar Cascades worked in OpenCV. Because of lack of scaffolded code, we ran into a few debugging issues in the code to detect faces on the neato cameras, and that was mostly because we didn't know how to call our different functions in order to run them. So lots of learning along the way, and in general, good practice for building ROS packages! Ultimately our lack of skill with python held us back and prevented us from implementing the program with the neato for a long period of time. Fundamentally we understood what we wanted to do, but we were not able to figure out the finer details. If we had had more time and help debugging python problems, we would have been able implement a more sophisticated neato component. It would have been fun to implement a proportional control function that drove the neato forward or backward based on the smile detector probabilities. Through this project we were able to learn the fundamentals about how image detection worked and a high-level overview of how machine learning worked to train a program to identify images. We can apply this knowledge to future image detection projects as well as other mobile robots.

Resources

Face Recognition with Python, in Under 25 Lines of Code
https://realpython.com/blog/python/face-recognition-with-python/

Face Detection and Recognition (Theory and Practice) by Eyal Arubas, Written April 6, 2013
http://eyalarubas.com/face-detection-and-recognition.html

Cascade Classification: Haar Feature-based Cascade Classifier for Object Detection¶
http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html

CV Bridge tutorial
http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython

Tutorial on OpenCV Haar Classifier
http://lab.onetwoclick.com/face-reader/

Smile detection via corner detection
http://cnx.org/resources/8748c712de264786a3a35b46f356d2f0/Smile%20Identification%20via%20Feature%20Recognition%20and%20Corner%20Detection.pdf

Debugging with the Python Debugger (Pdb)
http://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/

Smile detection classifier training
https://sites.google.com/site/datascience14/lectures/lecture-14

Appendix

Commands to run the robot

roscore
roslaunch neato_node bringup.launch host:= IP address of the py
to run camera: rosrun image_view image_view image:=/camera/image_raw

to run code to drive robot: rosrun visionproject drive_ifsmiles_v3.py

Project Proposal

Thrust A: Run OpenCV's facial detection algorithm to find person in frame of view and command robot to drive in that direction (keep human centered at all times)

- Run face detection algorithm, get centroid of face
- Compare centroid location to vertical axis of picture
- Direct robot to drive the wheel opposite from the turn direction at a faster speed (with some exponential gain times the distance away from the middle of the picture) in order to get the human to be centered in the field of view

Thrust B: Using database of smiling faces, run sci-kit learn to create a basic machine-learning algorithm to detect smiles

Integration: Detect faces, run them through the machine learning algorithm, and see if people are smiling. Change robot-drive algorithm such that robot only drives towards smiling people.

Links to useful resources:

Facial expression detection (talks about FACS too)
http://www.cc.gatech.edu/~vbettada/files/FaceExpressionRecSurvey.pdf

Open CV with face recognition
http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html

Convert between ROS images and OpenCV Images
http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython