Getting Started with Raspberry Pi & Open CV

Hardware you will need

- Monitor
- Raspberry Pi
- Raspberry Pi Camera
- Roomba ICreate2 + dock + USB connecter cable
- HDMI to VGA adapter (if not using an HDMI Monitor)
- Power Supply (micro-USB, think android charger)
- SD Card about 16GB
- USB Keyboard and Mouse
- Ethernet Cable

Software you will need

- Raspberry Pi OS
- OpenCV
- Python

Helpful tutorials

Getting Started (https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started)

- This tutorial will go over the basics of what a Raspberry Pi & help you to get your OS up and running

Installing OpenCV & Python on your Raspberry Pi (https://www.pyimagesearch.com/opencv-tutorials-resources-guides/)

- Thorough guide on how to add OpenCV and Python onto your Pi. This will take some time so prepare to let your Pi sit for a while.

Tutorials that explain image processing & detection using OpenCV/ Raspberry Pi

- In depth look at OpenCV and image processing (https://www.pyimagesearch.com/category/tutorials/)
- This was one of the most helpful tutorials we experience. He goes over creating Haarcascades, OpenCV, detection, and getting facial recognition to work on a static image. (https://www.youtube.com/channel/UCfzlCWGWYyIQ0aLC5w48gBQ)

Steps to run Project

I. **Phase 1: Create Dataset**
   a. Open terminal and go into the dataset directory. You will then run "python 01_dataset.py" to create your dataset.
   b. You will need to enter a User Id that you will associate your dataset with. For our purposes, we chose one and left zero in the array empty.
   c. This will take 30 pictures and make them black & white
   d. Check your dataset folder and pictures of the user should be in there. For our purpose we created a separate dataset-picture directory and had our program store them there.

II. **Phase 2: Train Recognizer**
   a. Open terminal and navigate into the file directory where your trainer is. We created a different subdirectory for trainer. You will then run "python 02_dataset_trainer.py" The training is done by a specific OpenCV function called recognizer.
      i. Note – You will have to add the user's name in the name array in the 03_face_recognizer program. This is so that the name of the user will appear when the user is on camera.
   b. We are also using an OpenCV function that is built in to detect the by using a Haar Cascade Classifier.
      i. Note – you can either get a Haar Cascade Classifier online or go through the process of making one. We found a facial one online here (https://raw.githubusercontent.com/shantnu/Webcam-Face-Detect/master/haarcascade_frontalface_default.xml)
   c. Your terminal should tell you how many faces have been trained after the program is through running.

III. **Phase 3: Detect your face**
   a. Go back into your original dataset directory and run the facial recognizer program "python 03_face_recgonizer"
   b. Now you will detect a face. This will call on the recognizer function recognizer.predict which will capture a portion of the face to be analyzed and return the ID of the owner. The closer the level of confidence is to 0, the closer the perfect match.

Summary of Project

The idea of this project launched sometime between the Introduction to Robotics offered during the summer of 2017 and the new 2018 year. We were given a Raspberry pi to explore with the intention of offering a project that combined image processing and autonomous vehicles for our senior design project. These are our findings.

The first steps of many, was uploading the Raspberry Pi OS, OpenCV, and Python onto our Raspberry Pi. We could accomplish these tasks by using a set of peripherals we designated to use only with our Pi. That way, we could use the Pi like a computer with a monitor, keyboard, and mouse. A piece that was important in getting off the ground was the acquisition of an SD card with 16GB that would hold all our memory. This was essential in running the Raspberry Pi because without it, the microcontroller would have no OS. The SD card had to be inserted into another working computer and then the OS installed there. Once the files were onto the SD card, we then added it to our microcontroller and booted up and connected to Wi-Fi via ethernet cable or public Wi-Fi. Through a site called Humble Bundle, we had access to several Raspberry Pi books that taught the basics of GPIO headers and programming on the microcontroller using Python. This was a key component in learning more about the Pi over winter break.

When the Spring semester began, we finalized the purpose of our Senior Design project. The purpose of our project originally was to demonstrate an autonomous vehicle using a Roomba ICreate2. While the Roomba would be completely autonomous, it would also be able to process images for facial-object detection and recognition. This came with many challenges because neither of us were proficient in OpenCV or python when starting the project. The first challenge we started on was simply being able to connect a camera to the Pi and have it take a picture. We realized soon enough that not just any USB camera would work for our purposes. After a few attempts, we decided that the best camera for the project was a Pi camera because it was already configured to use with a Pi and it had libraries we could use in conjunction with OpenCV. We then wrote a basic code that imported the Pi camera's libraries to take a static image when ran.

From there, we decided the next step should be getting faces detected in images since it seemed like the logical stepping point to get facial detection in real time. After some research, we found that the easiest way to detect a face was using a Haar Cascade Classifier. In machine learning, this is one of the most proficient ways in facial detection because its' function gets trained using hundreds of positive and negative images, and then used to detect objects in other images. Originally thinking we had to make our own Haar Cascade Classifier, we found helpful tutorials on how to do that. However, we soon discovered that a facial Haar Cascade Classifier exists online. Using that, we were able to develop code that captured an image, put it into grayscale, and draw a square around the detected face.

Once we were able to detect faces, we were faced with the challenge of facial recognition and detection in real time. Again, OpenCV makes it easier to do this because it has a lot of libraries you can use to accomplish this goal. We first began by breaking down live stream facial detection in three easy steps which are Gathering Data, Training our Program, and Recognizing the User. Gathering Data is the first program. This first program captures a user, loads an OpenCV Face detector using the Haar Cascade file, and creates a list of captured faces of the user in grayscale. It then places rectangles over the

recognized face and inputs all the files into a folder called Datasets. Datasets is a folder in our program that holds all the images of our users.

The second program takes the files from the dataset and trains them using another built in function from OpenCV against a recognizer called LBPH. LBPH stands for Local Binary Patterns Histograms. Faces are composed of visual patterns so what LBPH does, is it takes each pixel of a picture and compares it with another. Every pixel that is greater or equal to the center pixel, gets set to a value of 1 and then 0 for the rest. It extracts it, converts it into binary, and then each block value is then turned into a histogram that forms a vector feature of one image, which has the features we are interested in comparing. As a result, a .yml file is created with all the data the recognizer trained that pertains to that user.

Finally, we take a live video stream and first detect a face using the Haar Cascade Classifier. Once we detect a face, we then call the recognizer.predict() function that takes the recognizer's data and predicts if the face belongs to a trained user. It should then return the users Id and how much confidence the recognizer has in relation to the match. If the confidence is closer to 0, then the recognition is more precise. In conclusion we were able to successfully detect and recognize faces. In the future we would like to test up to how many faces It can recognize and improve accuracy using various frames per second.

Unfortunately, we were unable to make our Roomba ICreate2 completely autonomous. This is in part since we could not connect to Wi-Fi while at University. An idea we had, was to process an image that would determine which direction the Roomba would go. Another idea was to process edges of a room to determine which direction the room was going. Then lastly, was to process an image of a line that the Roomba could follow. Though we saw implemented projects like these online, we decided to control the Roomba using keyboard arrows and software provided on the ICreate2 website.

In the future we hope to improve the software by first making all three separate programs for facial recognition into one streamlined program with potential for a user interface. We would also like to have the Roomba ICreate2 have autonomous capability while processing images for specified recognition beyond faces. Finally, the design of the robot itself could be improved by having a better power supply for the Raspberry Pi as discussed in the beginning of the project.