Software Development Work Diary

Alexander Ollman
Software Team Lead
VeinCam ENGN4221 Project - 2018

Week 3

Loaded Raspbian image onto provided Raspberry Pi. Installed required packages, including OpenCV which was used by the Venenfinder project.

The Venenfinder HackADay code was not working as intended. The networking segments of code where not applying, and as such we were not able to get a remote stream on a separate device. However, this code, supported by the Picamear documentation, gave us a great introduction to initializing the Raspberry Pi camera and taking photos/images with its library functions.

References:
https://github.com/Myrijam/Venenfinder
https://hackaday.io/project/26158/logs?sort=oldest
https://picamera.readthedocs.io/en/release-1.13/recipes1.html

---

Week 4

Script found to stream video to bit stream, which could then be fed into HTML server.

References:
https://randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/

Week 5

Began exploring image processing techniques to make veins clearer, more apparent and stand out from the rest of the image. The first technique trialled was performing a histogram equalisation on the image, which normalizes the contrast over the image. As a result, with veins absorbing the most infrared light, the veins were very dark and visible in comparison to the rest of the image. This processing was performed using code similar to the OpenCV tutorial on histogram equalisation. The processing occurred in what appeared to be real time.

For the ANU Open Day, two video streams (one with image processing, one without) was setup locally on the RasPi and displayed via a connected external monitor. There appeared to be no issues with the processing or the stream.

Needed to find some way to apply the histogram equalisation to the video web stream. As incoming frames were being fed directly into the web stream, many hours were spent attempting to "intercept" this video stream to perform image processing. This proved to be unsuccessful, as the web video streaming script used functions that were quite hard to understand their internal workings with my limited knowledge of Python.

References:
https://hackaday.io/project/13329-yet-another-ir-vein-detector
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html

---

Week 6

Original streaming script was hard to modify, as classes from script found online were confusing to interpret how they were setting up the bit streams. After hours of attempting to modify this code, it was eventually scrapped all together. Instead, a Flask script was found which allowed for OpenCV functions to be parsed into a web stream. Initially, there was difficulty feeding a continuous video input from the PiCam directly into the stream. The solution to this was to iteratively have the PiCam take the last captured frame from the video stream and save it locally as a jpeg image, apply a histogram equalisation, and then feed that saved image back into the stream. This is performed every time an image is taken, which is dictated by the given framerate.

A video framerate of set to 60 frames-per-second was chosen rather than the previously used 32. Increasing the framerate gave for a much darker yet smoother stream, and allowed objects focused under IR light to illumine far greater than they did at the lower framerate. This is most likely due to the lens having less exposure time, thus taking in less external interfering light sources between each shutter. As a result, veins appear are far more clearly visible at the higher framerate under infrared light.

References:
https://blog.miguelgrinberg.com/post/video-streaming-with-flask

---

Week 7

Decision was made unanimously by the team to have the primary focus for software development for the remainder of the project to be the Flask/JS web interface to allow for manual variable control. This will include sliders and/or buttons to allow for the manual adjustment of camera attributes (saturation, brightness, contrast, etc) and the level of equalization/processing occurring to image. This decision was made in line with the decision to focus on the VeinCam to be an educational tool, and the manual control will allow users to have more control to see what variables will get the clearest vein images.

Began into finding articles relating to sending variables (such as slider values) from HTML to Python via Flask. This required learning a lot about how a Flask server works in a very short period of time, most of which was through trial and error. Found a script that allowed for a variable to be sent and saved in Python upon pressing a button. Most attempts were not successful as each Flask "route" (function) a HTML form could trigger would take the webpage away from the main page. Could not find a way to refresh page and/or redirect to the main webpage upon clicking aforementioned button.

References:
https://stackoverflow.com/questions/43677564/passing-input-from-html-to-python-and-back
https://stackoverflow.com/questions/29884654/button-that-refresh-page-on-click

---

Week 8

Discovered all that was required was to have each button-triggered function return the index() function after saving the parsed variable, which would then reload the main web page. Variables could now be sent/functions could now be triggered from a HTML page to a Python script.

Week 9

Attempted many trial-and-error methods of modifying the camera_opencv and base_camera scripts to be able to take in the saved Python variables from app.py as an imported library. This did not work. Figured instead to save variables locally to a text file that could be read and written by all scripts involved.

Parameter sliders were designed in two parts. The first part was to have the value of a HTML slider (0 – 100) be saved to a text file. The second part was adjusting the camera_opencv.py script to read the value in the text file and set the camera's associated parameter to that value. Using W3School's slider tutorial and Flask application instructions, able to take current slider value and save to text file. Learned how to read and write text files in Python.

Used same read/write functions to read a text file and use value as camera parameter (such as camera.brightness). After a few tests, it was determined that it was best to have these text files be read during the image processing of every frame, so that the parameters would be updated for the following frame. This would incur a visible update delay of 1 second to the stream, but we figured this was still suitable for purpose.

References:

https://www.w3schools.com/howto/howto_js_rangeslider.asp
https://stackoverflow.com/questions/6648493/open-file-for-both-reading-and-writing

https://picamera.readthedocs.io/en/release-1.10/api_camera.html

Week 10

Made the decision with client to move away from sliders to adjust parameters of camera and use large buttons instead, so that all devices can easily use.

Buttons adjust +-10 of each parameter.

Attempted to see if variables updating camera parameters could happen dynamically, or without requirements to refresh page. Discovered this will most likely require JavaScript. Given time constraints, this was figured to be left as a future improvement should the project be continued.

Webpage HTML updated to final design, including colour scheme and button styling. Used imported PiJuice functions to gather status of battery. These functions returned as python dictionaries, of which I was unfamiliar with. Found functions to extract values for charging status and battery charge percentage, which were then used in app.py to parse to the web page. This allowed users to know the battery percentage of the system at any time, and know if the device was charging properly when connected to power (seeing as, at this point of hardware development, there were no visible battery indicator lights outside of the enclosure).

Feedback from Ben was to have stream portrait as opposed to landscape, as most devices will be portrait orientated and thus can better view a persons arm. Rotating image using camera.rotation object parameter or in HTML/CSS did not properly rotate the image, or scale properly for multiple device browsers. Found OpenCV script by Adrian Rosebrock which performed perfect matrix rotation of image array without cropping or resizing image. Parsing the rotated image straight to webpage presented no issues with resizing.

Attempted to localise font within CSS so that all devices can view desired 'Baloo' font. Was not successful as this is not a default HTML5 font and with the device streaming over a local WLAN, connected devices will not have internet access to import the font. Instead, we will opt for similar but compatible font to use during poster presentation.

References:
https://github.com/PiSupply/PiJuice/tree/master/Software
https://www.w3schools.com/css/css3_buttons.asp
https://www.pythonforbeginners.com/dictionary/how-to-use-dictionaries-in-python/
https://www.pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/