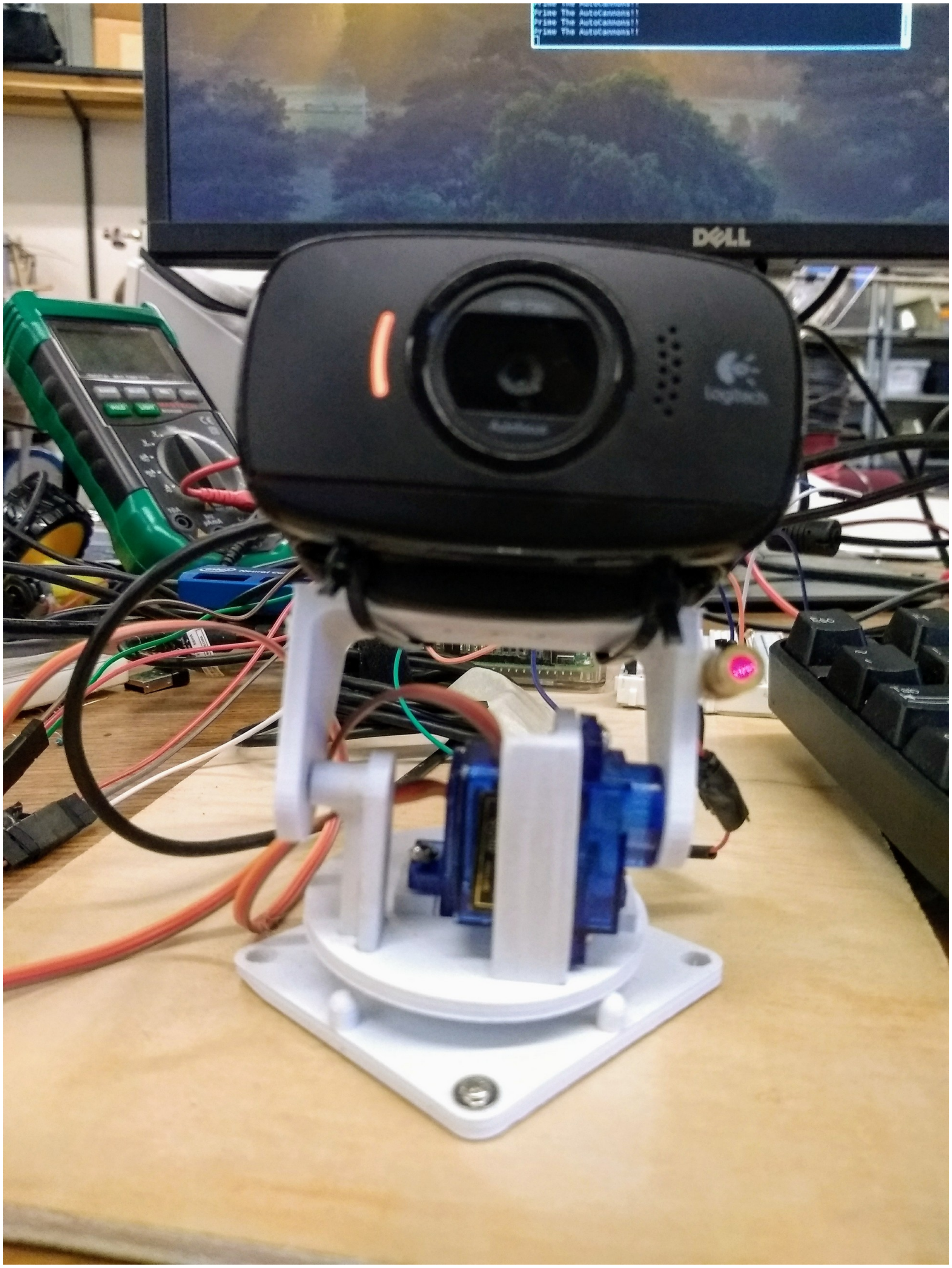
Auto Turret


Prime The AutoCannons!!
Prime The AutoCannons!!
Prime The AutoCannons!!
Prime The AutoCannons!!
DELL
logitech

Overview:

My goal in this project was to create auto turret capable of looking for human targets in a scan mode, locking on and following its target. In the end, I decided to use facial recognition to lock onto human targets. I achieved everything I set out to do, however there are several changes I would make to this project to make it better and more cool

Hardware Requirements:

- Raspberry pi: The raspberry pi has the ability to control and power peripheral devices. The raspberry pi is used to control the positions of the vertical and horizontal servos on the pan tilt camera. Also, it acts as an interface for the neural compute stick 2 (nsc2).
- NCS2: The ncs2 allows for faster processing of neural networks. There are two main bottlenecks of doing neural network processing on the raspberry pi: ram and cpu. Neural networks, especially CNNs, take up huge amounts of ram and can cause lag which inhibits the ability to track targets. The ncs2 alleviates this ram and cpu bottleneck on low powered devices and vastly increases inference time[1].
- 9G Micro Servos: servos which allow for 180 degree rotation[5].
- Pan Tilt Camera: Decided to use a 3d printer for this[6].

Software Requirements (for Raspberry Pi):

- cv2: Need to install opencv library for python.
- Numpy: Need to install numpy library for python
- OpenVino Environment: Intel neural network inference platform. It is the software that the neural compute stick needs to make inferences. Guide found here: https://docs.openvinotoolkit.org/latest/_docs_install_guides_installing_openvino_raspbian.html
-

Code Description:

There are four main parts to my tracking code: angle calculations and servo control, asynchronous classification, and the scanning code.

- Angle Calculations: For this project I assumed a uniform distribution of pixels across the horizontal and vertical field of view (FOV). For example, if the vertical FOV is 40 degrees and there are 512 pixels vertically spaced then each pixel spans 40/512 = 0.078125 degrees vertically. The vertical and horizontal FOV can be derived mathematically from the diagonal FOV of the camera[4].

- Servo Control: The servos have a range of 180 degrees and a pulse width from 500 to 2500 microseconds. As we did in class, degrees can be converted directly into pulse widths. Thus, simple ratios can be used to determine the angles necessary to bring targets into view.
- Asynchronous classification: Inputs are queued on the neural compute stick using the net.setInput command. A call to out.wait_for(0) can alert the user when a prediction becomes available. Asynchronous classification allows for a high frame rate and seamless switching between scan mode and tracking mode.
- Scanning Code: After 1 second of not seeing a target, the scanning routine will begin. When a target moves out of view, it continues scanning in the direction it was previously scanning in.

Conclusion:
This project was a lot of fun and I learned a lot. However, there are several flaws with the current implementation. For example, there is no calibration to account for the curvature of the camera lens. The code and a video of the automated turret can be found inside the zipped project folder.

Sources:

1. https://medium.com/@aallan/benchmarking-edge-computing-ce3f13942245
2. https://www.amazon.com/gp/product/B071FT9HSV/ref=ppx_yo_dt_b_asin_title_o05_s00?ie=UTF8&psc=1
3. https://www.circuitsdiy.com/lm317-constant-current-source/
4. https://www.learnopencv.com/approximate-focal-length-for-webcams-and-cell-phone-cameras/
5. https://www.amazon.com/gp/product/B07MLR1498/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&psc=1
6. https://www.thingiverse.com/thing:107957