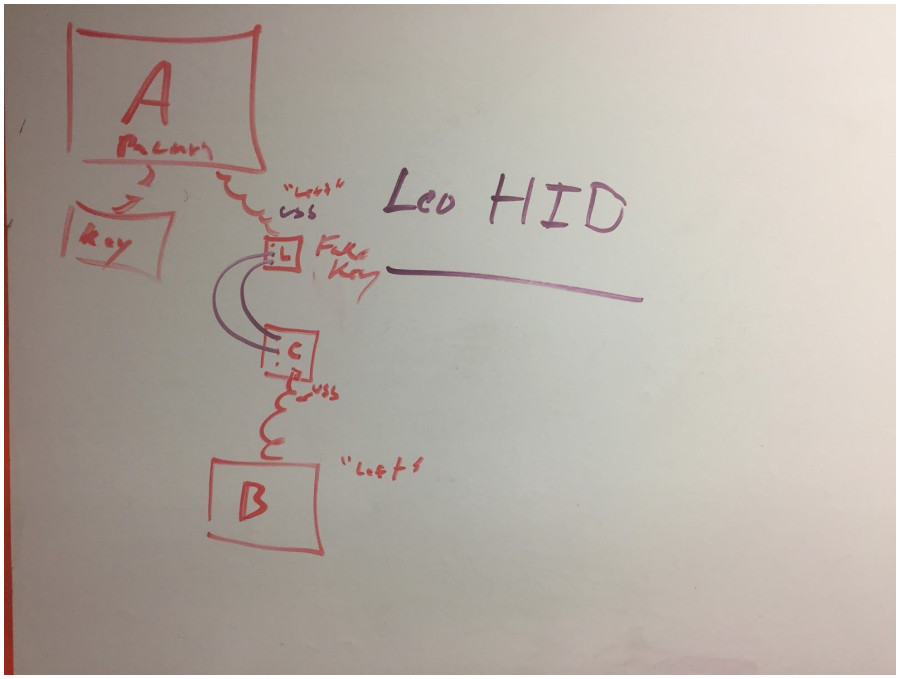
Deep Reinforcement Q-learning Control System Using an Air-Gapped Optical Input Stream

We create a novel proof-of-concept machine learning control system by applying deep reinforcement learning to master a 1990's Car and Driver game through an air-gapped optical input stream. This can be applied to learn to operate legacy infrastructure hardware to maximize efficiency and reduce costs.

Plan (very important figure):



5/19:
Goal: setup like the last time and make progress
Conclusion: unsuccessful attempt at connecting the computer to the arduino

Materials:

- Arduino Leonardo
- https://raw.githubusercontent.com/thearn/Python-Arduino-Command-API/master/sketches/prototype/prototype.ino
- Blue cord semi-transparent cord to connect to arduino
- LED
- Resistor

Process:

- Connect leonardo
- Connect two wires, one with GND, one with 13
- GRND connects to -, 13 connects to breadboard
- Etc. etc. connect everything
- Open Arduino IDE
- copy/paste in https://raw.githubusercontent.com/thearn/Python-Arduino-Command-API/master/sketches/prototype/prototype.ino
- Click upload and immediately press reset button, now it is uploaded and you can now use the arduino command IDE
- Using terminal, run Arduino_Test2.py

Sites to help with Arduino:
https://github.com/thearn/Python-Arduino-Command-API

http://stackoverflow.com/questions/19765037/arduino-sketch-upload-issue-avrdude-stk500-recv-programmer-is-not-respondi
https://www.arduino.cc/en/tutorial/blink

6/2/17

- Did all of the above and ended up connecting the computer to the arduino (LED circuit)

What did I do to make it work?

- Used $ top command to kill avrdude process
- Still pressed reset button right after running connect program
- Changed serial port to /dev/ttyACM0
- Used AVRISP mkII Programmer
- Used MEGA2560 R3 board

- Then, used code from Sivasantosh: Keyboard event handling in pygame **(https://tinyurl.com/yacynffj)**
- Progress: reduced it down to its bare minimum, ending up with this (Gist link: https://gist.githubusercontent.com/rodrigo-castellon/d5d8db1ab3cf52d27b27bfab0a6cd5ed/raw/203e7b56a6a8bb0ee7a43fdffcbe2cd7d6c96fac/pygametest.py)
    - What it does: run it, arrow keys will print out into terminal, up and down will turn LED on and off
    - Before, all our program did was flicker the LED on and off without human input
- Accomplishment today: in the diagram on the whiteboard, achieved the connection between Q-learning computer and an arduino. What is left is to connect this arduino to a keyboard spoofer and eventually to the atari machine.
    - What is the purpose?
        - To test that this keyboard connection between the Q-learning computer and the Atari computer works, we would need to see if a keyboard connected to the Q-learning computer could type out into the other computer (using Microsoft Word, some other text editing software). This is the first step of this process, and by accomplishing this step we make sure that we are able to connect a keyboard to an arduino to achieve part of this connection.

6/5/17 (8:45-12:00)

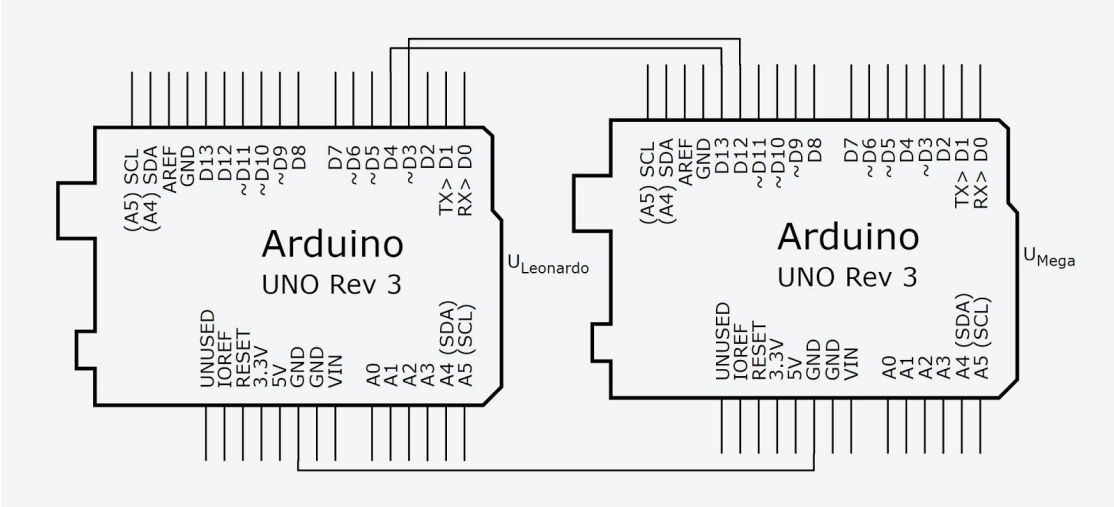
- Executed all of the above successfully in about 5 minutes (previously took hours)
- Website for checking pygame key codes: https://www.pygame.org/docs/ref/key.html
- Next step: to connect the Mega to the Leonardo (second step in the connection between computers)
- Ran the same procedure but instead used my mac laptop and Leonardo (to see if the Leonardo worked)
- Today's procedure:
    - Started with the same circuit as before
    - Took out white wire connecting pin 13 to breadboard
    - Replaced with longer red wire that connects to RX pin in Arduino Leonardo
    - Connected pin 13 in LEONARDO with where pin 13 white cable used to go (next to the resistor)
    - It should look like this:
        - https://ibb.co/gvPKAv
        - https://ibb.co/hSEaiF
        - https://ibb.co/hqBMOF
        - https://ibb.co/j281OF
        - https://ibb.co/dW7viF
        - https://ibb.co/mefgOF
- Action space for a few games:
    - BreakoutDeterministic-v3: 6
    - MsPacman-v0: 9
    - Pong-v0: 6
    - Space-Invaders-v0: 6
    - Breakout-v0: 6
- Later on, the procedure above turned out to not work (too confusing):
    - Started over with Mac laptop connected to Leonardo
    - Made same base circuit for a light bulb (pin 13)
    - Added another light bulb and connected it to pin 12
    - Ran this program: https://gist.github.com/rodrigo-castellon/1a96945e00bde4893df5593a96cd423a
        - What it does is alternate between the two lights, demonstrates that we can use multiple pins (12, 13)
    - This system's purpose: the top two components of the figure on the top (keyboard spoofer)
    - Next step(s):
        - Make pressing keys control LED light bulbs (done same day)
        - connect Mega board to pins on the Leonardo board. Make a program that says "if pin is on (coming from Leonardo), turn on LED (later on spoof as keyboard)"

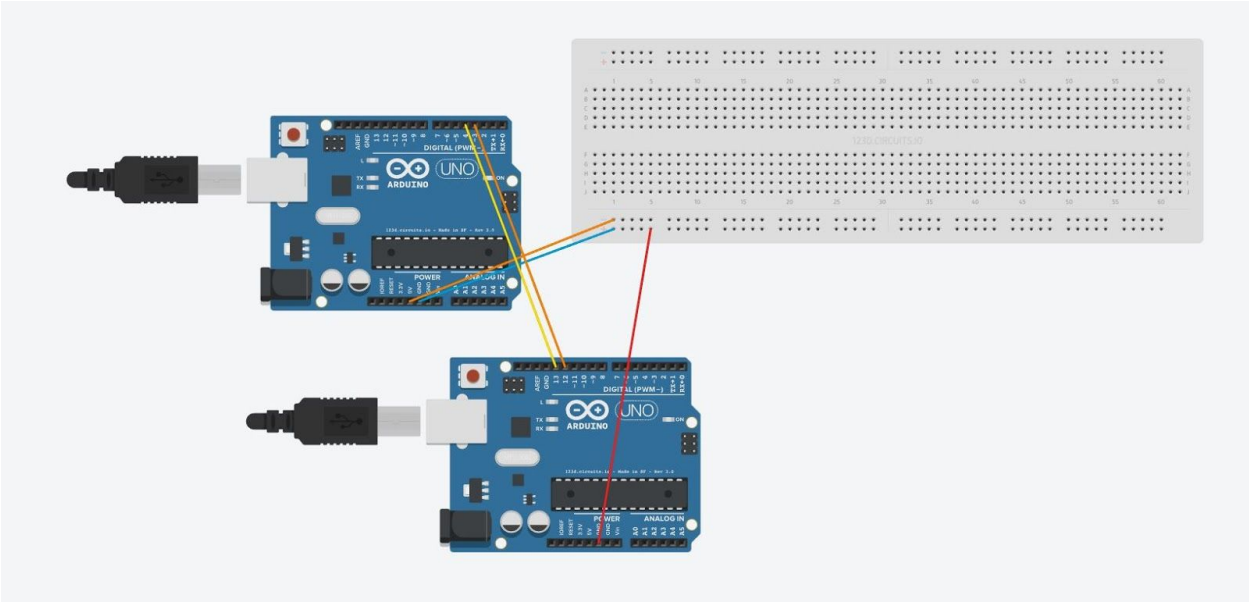
6/6/17 (9:00-12:00)

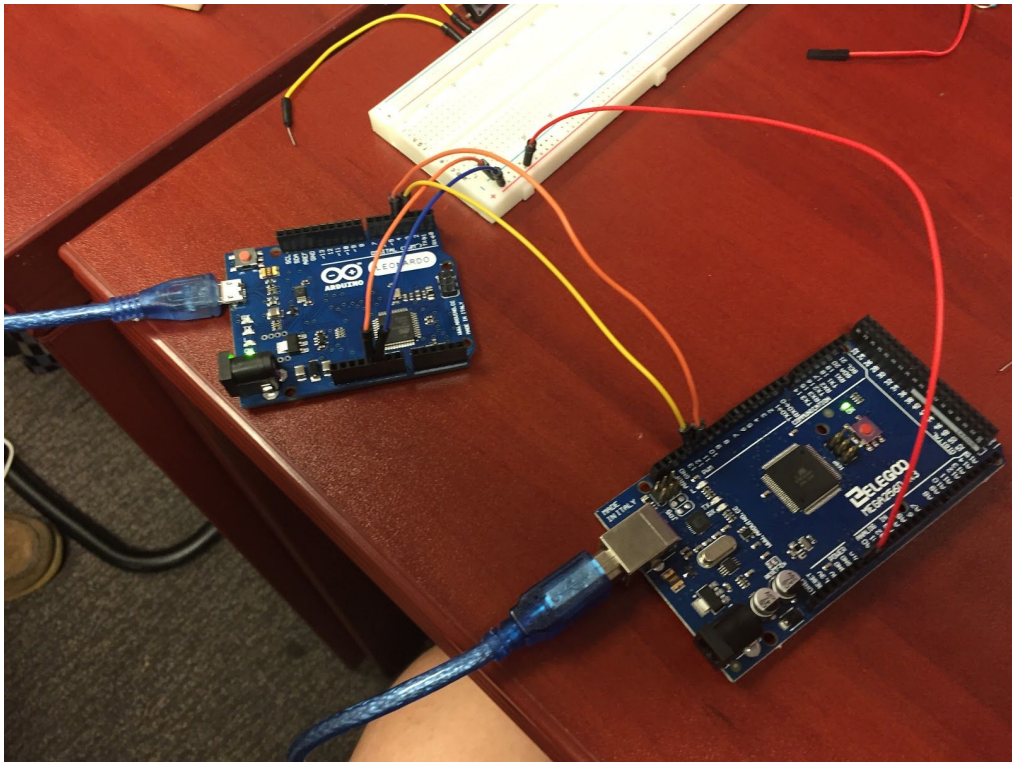
- What we decided to do is test it with output from an accelerometer
    - Each dimension (XYZ) serves as an individual “key”, each mapped to a light bulb, which will light if that dimension’s threshold is passed
- Finished above, Procedure:
    - Acquire redboard, accelerometer, light bulbs, resistors (brown brown black black brown), cables)
    - Setup accelerometer
        - Attach accelerometer to e column of breadboard
        - Attach wire on 5V pin that goes to the VCC, attach wire on GND pin that goes to GND, and attach 3 wires that are on analog pins that go to the three analog spaces on the accelerometer
    - Accelerometer output to light bulbs:
        - Attach three wires on three digital pins and set up light bulbs on the breadboard using the resistor and light bulbs
        - Add a ground wire attached to the - column of the breadboard
- We have demonstrated thus far that we can build an arduino that can detect key presses and serial connection input.
- Next step: connecting a leonardo to a computer and see if we can spoof input
    - It cannot be done through python (Arduino Python Command API, only through Arduino IDE)
    - Problem: uploading any Arduino program to the board
        - Fixed: Plugged in arduino board directly to PC, not through keyboard USB port (voltage may be favorable)
    - Another problem: once keyboard emulator program is uploaded, very hard to reset (interfering keyboard presses)
        - Fixed: command+n to create a new blank arduino program (setup and loop function empty), connect arduino, then hold reset button throughout, select correct board and port, then click upload and release the reset button.

6/7/17 (9:00-12:00)

- Finally wrote program and wired it up so that Leonardo could SAFELY pretend to be a keyboard and press keys
    - Program: https://gist.github.com/rodrigo-castellon/977c1baf563d9f4517e799305cc14545
    - Wiring: https://www.arduino.cc/en/tutorial/button
        - In this case, I used a 10K Ohms 2% resistor (brown black black red red)
        - The purpose of the button is to serve as a switch to start the program under our control (not immediately jump into loop)
- We have accomplished:
    - Leonardo (in a loop) spoofing key commands to computer (today)
    - Receiving signal and acting upon that (using analogRead with accelerometer, 6/6/17)
- Next overarching goal: **Getting two computers connected using these techniques and pass through keyboard presses from the "learning computer" to the "game computer"**.
- Immediate goal: update leonardo spoof code to listen for digital input and press keys based on that
    - Let's try to listen in pin 3, and have pin 13 as output directly to pin 3
- Overarching goal achieved! Process:
    - Obtain Arduino Leonardo, Arduino Mega2560, breadboard, unix-based PC (for python; "q-learning" computer), "atari/game" computer (no requirements), arduino jumper cables, NO resistors
    - Connect wires as such:



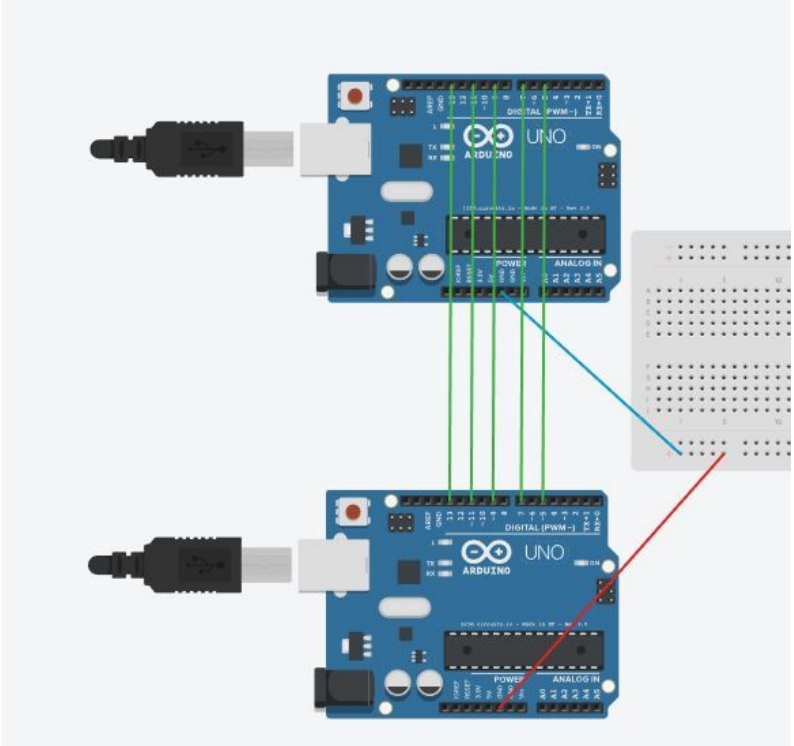


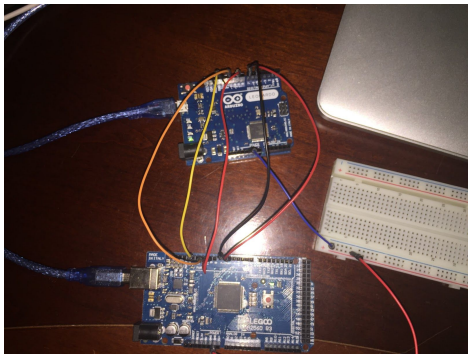


- Run programs mega_blinksend2.py and spoof_test5.ino on both computers (spoof_test5.ino took some trial and error, but eventually got it to run)
- What does it do? If you press an arrow key on the “learning” computer which is connected to the Mega, it propagates through and “presses” either ‘a’ or ‘b’ on the “game” computer (depending on whether you pressed up or down)

- Next step: Add more keys to add robustness to the system (action space for pong is 6, so 5 connections)
- Achieved! Spoof_test6.py is written to incorporate numbers 1-5, and mega_pysend2.py does the same. Also, pins to connect are spaced out every other to free up space. This

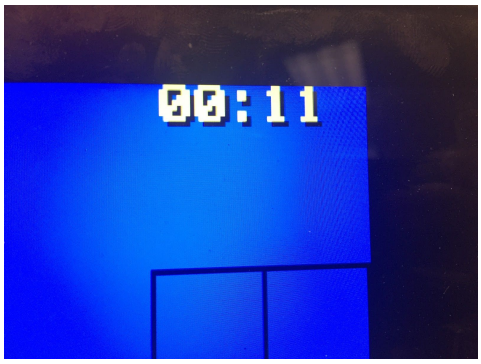
means that we now have an action space of 6 (includes no button pressed), which is required for pong.





6/8/17 (9:10-12:00)

- Now that the physical keyboard loop is completed, I tested it with breakout online, mapping ‘5’ and ‘4’ on my mac laptop (in this instance the “learning computer”) to right and left arrow keys. It was very responsive and did not see any issues.
- Eventually, we would like to take the OpenAI gym code and experiment with it to see if we could send keystrokes with it…
    - We can’t use Docker for the learning computer, since it cannot connect to the arduino. This means we need to use a computer with all of the dependencies pre-installed running modified. (This means at the MPCR Lab station I switched the configuration around, the game computer is now the learning computer and vice versa, since my laptop does not have dependencies installed)
    - Here is the original code we are going to build off of: https://github.com/tflearn/tflearn/blob/master/examples/reinforcement_learning/atari_1step_qlearning.py
- We decided to use Boxer (an emulator for DOS games), specifically a game called “car and driver” because it would have a simple reward mechanism. Reward: how long car lasts on the track before going outside.
- Next step: make a program that can recognize the score on the game “car and driver”. This score is the number of seconds spent at a non-zero speed while not off the road. We will use PyOCR to do OCR
- Achieved! Procedure:
    - Obtain ocr1.py through my gist, get image and put it in the same directory as ocr1.py, then change ... Image.open(‘...’), to image name.
- Next steps: Since it wasn’t perfect and took 1-2 seconds to spit out the answer, we would need to look for a faster way to do OCR (possibly with the same library PyOCR itself), and to account for false cases (both below), pytesseract can be retrained like a neural network, eliminating these false cases.



End-of-week synopsis:

At the beginning of the week, I was barely able to even connect an arduino to my laptop/computer. However, by the end of the week, I was able to make the link between two computers and send keystrokes, utilizing python on one computer to send the keystroke (pygame library, Python arduino command API) and using the arduino language in the other (Leonardo). The next step in this project, which I already started on Friday, is to get a quick OCR program working. The purpose of OCR is to check how much time has passed (top right corner of “Car and Driver” game) as well as to check whether the car has gone off of the road. (“Press Q to return” message in the center of the screen). On Friday, I took a picture of the game screen, and then used a PyOCR program I wrote to read the time digits, which didn’t work perfectly. This week we will first be looking for improvements we can make to the OCR (digit-only, real-time, font-specific) and then eventually modifying this code to take input from a webcam and read the OCR-outputted reward.

6/12/17 (8:45-12:00)

- Got this program working on both computers, installed tesseract, pytesseract, and pyOCR for it to work
- Improved/condensed the previous program to a few lines, using pytesseract instead of PyOCR. Dr. Hahn told me that PyTesseract had additional features, like font-specific training, restricting to digits only, etc.
- I have begun to use https://github.com/tesseract-ocr/tesseract/wiki/Trainingtesseract-4.00 to learn how to train tesseract
    - found directory for pytesseract and tesseract using pip show tesseract
- Rest of the morning was spent on building my computer

6/13/17 (9:15-12:00)

- We should see if we can train tesseract in general first, and then see if we can train it with the on-screen text
    - Using this website as a guide: http://michaeljaylissner.com/posts/2012/02/11/adding-new-fonts-to-tesseract-3-ocr-engine/
    - Maybe this font could work: http://www.dafont.com/pixelmix.font
- Spent rest of the morning building computer to do machine learning at home, finished building and installing OS, etc.

6/14/17 (9:00-12:00)

- Found a website to train OCR with a font: http://trainyourtesseract.com/, which takes the font file and emails you the .traineddata file necessary
- It turns out .traineddata files are the languages you select from in https://gist.githubusercontent.com/rodrigo-castellon/20bd03a98a0573c6b7ab4c5a1c4755f1/raw/c480d6c6ce38c13397e3d6b69483127e6c20bead/ocr1.py
- I then used Glyphr Studio to take out umlauts, weird letters from pixelmix font. After I trained tesseract on digits, lowercase, uppercase, and special character and moved it to the right folder (goes to ~/usr/share/tesseract-ocr/tessdata/), I used ocr2.py (gist) and tested it on many pictures of the 'car and driver' screen (images found in OCR Images folder on Google drive).
- For on the track images testing: (correct means 00:0x or 00: 0x outputted)
    - 00:03- correct
    - 00:05- correct
    - 00:06- correct
    - 00:08- incorrect (' ')
    - 00:09- correct
    - 00:11- correct
    - 00:12- correct
    - 00:16- correct
    - 00:17- correct
    - 00:19- incorrect (':19')
    - 00:21- correct
    - 00:23- correct
- For off the track images testing:
    - 00:24- incorrect
    - 00:25- incorrect
    - 00:26- incorrect
    - 00:48- incorrect
    - 00:51- incorrect
    - 00:57- incorrect
    - 00:58- incorrect (00: 48)
    - 1:05- incorrect
    - 1:08- incorrect ("press Q tc return tc rczu 01:08")
    - 1:11- incorrect
    - These did not detect text in the center of the screen that the game ended
    - However, if image is cropped to envelope the "Press Q…" text, program will return: "Fress Q tc return .:O rczc"
- At least now the OCR program recognizes the time elapsed, but it cannot recognize text in the center of screen…
- We think we might have to look for a "target" function on pytesseract so it can target "Q" for example

6/15/17 (9:00-12:00)

- Next step: look for a target function or something that can detect the text in the middle of the screen
- We have been using PyOCR, but I found TesserOCR (https://github.com/sirfz/tesserocr), which by the example page, seems to support more functions

    - Installed `tesseract-ocr libtesseract-dev libleptonica-dev`
    - Installed using pip
- To find out how to train the font under this library again is too complicated. I went back to the github for PyOCR (https://github.com/openpaperwork/pyocr), and there seemed to be more options there originally than I had thought
- There isn't any target function, but maybe I can crop the image in the same python program and detect text in the top right corner and also focus on the center of the screen
    - You can crop using PIL and pillow (https://automatetheboringstuff.com/chapter17/)
    - Program to crop:
        - From PIL import Image
        - Import os
        - Im = image.open('on5.JPG')
        - croppedIm = Im.crop((2000,0,3264,500))
        - croppedIm.save('cropped.png')
    - Pictures from iPhone are 3264(width)x2448(height)
    - Im.crop((x1,y1,x2,y2)) x is pixels from left, y is pixels from top to bottom
    - To crop right hand corner for time: 2000,0,3264,500
    - To crop center for "Return to track" message:
- Actually, we should scrap all of the OCR since the OCR is very slow (2 seconds). New Plan: Dr. Hahn suggested to instead just train a neural network to detect the message. We could drive the car around for 10-20 minutes on and off the road, collect thousands of frames of on and off the road, then use TFLearn network to train on those images, then it would be much faster.
- But I tested the OCR again, and it only took about 0.4 seconds to run, so maybe we can just use OCR, since after cropping an image it takes much less time to run
- We got an Xbox Kinect as camera, and connected to it, but were not able to connect using python

End-of-week Synopsis:
This week I built my computer so that I could run programs from home; I do not have a Linux computer at home, so this would help me throughout the year when I can just run code from home. This took about 2 days in total, since it was my first time building a computer. I also began to experiment with OCR and try to extract the time score and “Press Q to return to road” text from the screen with pictures from my phone. I initially used the entire image of the computer screen and default English font for PyOCR (wrapper around Tesseract library) to use. I figured out how to train Tesseract to recognize a more 8-bit like font, which helped to increase accuracy substantially. However, another improvement I made reduced the time it took for the program to do OCR on the image and also increased accuracy for the “Press Q to return to road” text. In the end, it took only about 0.4 seconds to run OCR to detect that text, which means we can use OCR in our final experiment. On Friday I began to connect the computer to the Xbox Kinect with libfreenect, and we got it to connect and show a stream, however we were not able to use python (pyfreenect) to connect, which is something I will work on. In summary, this week I was able to make progress in the camera portion of the loop, writing a program that can recognize the time passed and whether the car has gone off road, which means the “learning computer” can now recognize if each episode has ended or not, and how much reward it is receiving that episode (time passed).

6/19/17 (9:00-12:00)

- The goal of this day is to try to connect to the Kinect (haha) using Python, and seeing if we can write a program in python to connect. Previously, it was showing an error for using Pyfreenect, but we could see the kinect without python
- When I ran the test for libfreenect2: ./bin/Protonect, it kept returning this error [Error] [protocol::CommandTransaction] bulk transfer failed: LIBUSB_ERROR_IO Input/Output Error -> [Info] [Freenect2DeviceImpl] closing…
    - I solved it with `sudo sh -c 'echo 64 > /sys/module/usbcore/parameters/usbfs_memory_mb'`
    - Seems like it was an issue with the RAM/memory usage for the USB?
- Then I tried to run python test.py inside of the pyfreenect2-master folder, and it did create a window for a second, but then stopped responding and greyed out, returning [Info] [DepthPacketStreamParser] 7 packets were lost, although it keeps outputting [Info] [TurboJpegRgbPacketProcessor] avg. time: 15.5687ms -> ~64.2315Hz
    - Then I did this command: `LIBFREENECT2_INSTALL_PREFIX=~/freenect2 python setup.py install`
    - Now when I run test.py, it opens the “RGB” Window, but it doesn’t even receive anything from the camera, just says: init done
- Restarted computer -> ran ./bin/Protonect, did not work and said [Info] [Freenect2Impl] found 0 devices
    - but waited a while and tried again and it worked
- Then I ran python test.py, and it showed a frame, but then stopped responding like Issue #3
- Then I tried python test.py again and it didn’t even show a frame, also like Issue #3, also tried changing/debugging the code, and it seems like it gets hung up on line 42 (frameListener.waitForNewFrame())
- Then went to run ./bin/Protonect and that didn’t work, error: [Error] [protocol::CommandTransaction] bulk transfer failed: LIBUSB_ERROR_TIMEOUT Operation timed out
- I think it is because since libfreenect doesn’t work and pyfreenect builds off of it, maybe I should try just pyfreenect first
    - I unplugged and replugged Kinect, and it gave me the same error as in Issue #2
    - Ran python test.py several times again without replugging Kinect, kept giving me same issue as Issue #3
- Restarted computer and tried to run python test.py, returned ImportError: libfreenect2.so.0.2: cannot open shared object file: No such file or directory
    - Used command: `export LD_LIBRARY_PATH=$HOME/freenect2/lib:$LD_LIBRARY_PATH`
- Found someone with the same problem! https://github.com/OpenKinect/libfreenect2/issues/785
    - Maybe I have to update Linux Kernel Drivers or something like that?
- There’s another camera by Intel that we could use and see if it can connect… (using this to install drivers: https://github.com/IntelRealSense/librealsense/blob/master/doc/installation.md)
- Maybe this one too: https://github.com/toinsson/pyrealsense
- Haven’t been able to completely install them

6/20/17 (8:30-12:00)

- This time we're going to install librealsense -> pyrealsense
- Librealsense:
    - When I type in cmake ../ command, it returns this
- Tried installing everything on my mac laptop, and it installed perfectly. I then ran the "online usage" program here and added in a bit of code from here at the end like this, which outputs the last frame of 60 taken.
- I then wrote a program that took the OCR code and combined it with this code, like this

6/21-26/17: I was on a trip for the TEAMS competition (Meghna was also in TEAMS and went) so I did not do anything on these days.

End-of-Week Synopsis:
The first day I tried again reworking through the errors and trying to use python to connect to the Kinect, but I was unsuccessful. Fortunately, Dr. Hahn had another spare camera (not Kinect) called Intel Realsense that had a library that could connect python to it, and we did. I wrote a program that took a constant (should be 60 FPS, but only about 2-10 FPS in practice) camera input stream from the Intel Realsense camera, and did OCR on each of those frames, outputting text to the terminal. I am at a very good position in terms of my project progress. I have completed the arduino loop between 2 computers, and now I am almost completed with the input stream aspect of my project. Regarding the camera, I only need to add some minute details that allow me to optimize the OCR on the Car and Driver Game (cropping image). After that, I will begin work on the actual q-learning aspect of the project, switching out lines of code that do OpenAI environment-specific things, and replacing them with camera stream input, reward signal, etc.

6/27/17 (9:45-1:00)

- Now that the computer can extract the input stream from the camera and do real-time OCR from the game screen, I need to make a .traineddata file that specifically recognizes the digits or the “Press Q to …” message.
- I mounted my Intel camera and pointed it at the screen, then modified my program by cropping the image to only include the “Press Q” message, and then ran the modified program
    - Result: It outputted very infrequently when not off-road, but had constant output when off-road, although it could not read the text on the screen very well
    - Next step: implementing an “isoffroad” function into the program, which outputs “yes” or “no” at every timestep, instead of verbose OCR output (currently)
    - Since output when there IS a message “Press Q…” is usually very long, and on-road output when there is is very short (<10 characters), I will use this as a measure for the function
- I did this, in this new modified program
- Next step: crop the image on the top right and take the time at every frame

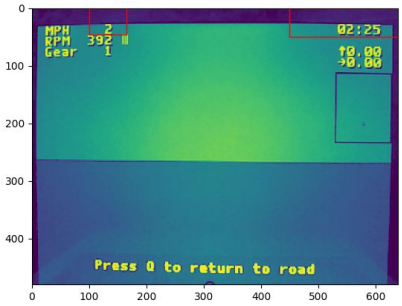
6/28/17 9:00-12:30

- Tried the above program after setting the camera up again, but since it wasn't exactly the same setup as before, the cropped image did not include the text of interest, which means I might have to either: automatically detect the text position on the bottom or set a standard for camera setup at all times
- The problem is that the OCR mechanism has to work with a very well defined image, and if it's given more noise than necessary, it won't be able to pick out the "Press Q to return" text. I keep moving the camera positioning every time I come to lab, so cropping out the text in the same pixel positions every time does not work.
- I will try to find ways of detecting text position, and then if it doesn't work, I'll have to make a standard camera setup.
- While looking at this: https://github.com/openpaperwork/pyocr I noticed that there is a box.position, which tells you the position of txt in question and is exactly what we need. What I can do is do a much larger setup crop at the beginning of every day at research to determine the position of the text boxes, and once that position is noted, I can use that position coordinates for the rest of the day
- I've managed to add in a rectangle to the image with this program, but now I should see if I can crop the image in the same place as I add this rectangle in.
- I did! I modified the above program and now it crops and shows the image with the rectangle indicating the crop, and it even works in detecting on/offroad surprisingly well

- Now I will try to have my program detect the position...
- I tried to do the example on the github, but it didn't work, returning that the list object did not have "contents" or "position". Maybe what I could do to solve this ENTIRE problem is to do what this guy did and greyscale the entire image and increase contrast to make the "Press Q to return to road" pop out more for the computer
- After I grayscaled it, I didn't even have to crop the image, and it worked VERY well, with NO false positives over the course of about a minute of driving!
- Now that I have managed to accurately detect when the episode is terminated, in the next step, I will do OCR on the top right of the image, and detect the time and turn that into an integer variable in the program
- To do this, I have already cropped out the top right part of the image, and modified the program to ask for what you want it to output. I'm going to modify to ask for raw_input at the beginning of the program instead

6/29/17 (8:50-11:00)

- I'm going to finish the code that adds in some user input…
- OK, it is finished, and it asks what you want the OCR to look for: either output if it's on or off the road, verbose road OCR output, or specifically cropped to the top right corner verbose output (Time detection)
- For time detection, it's not doing too well, so I'm going to modify the pixelmix font so it only includes digits and a colon
- OK, finished that, and I have the .traineddata file ready to go, with a modified program here, and it can detect on and off road fine, but time is a big issue. It picks up on the text right beneath it, adding noise to the text output which is not good…
    - What we could do is train a neural network to recognize the time and ignore what's underneath, or just crop it out. Actually, cropping it out is much easier, I'll try that.
- I've modified the crop parameters, and it works very well! Actually this is sort of pointless because I could just have a timer within the Q-learning program itself, which is way more accurate than this one, although I suppose I could use this OCR mechanism as a backup? Program here
- Since the "done" mechanism is pretty much ready to go, maybe apart from the screen observation the agent is getting, we can add in the MPH on the top left since that's pretty important
- I have added this into the program, and it does very badly with single-digits, but does very well with 2-digit numbers



-
- OK, now that this is done, we can start building the Q-learning algorithm framework around what I have done so far. First step maybe is just to get an OpenAI Gym-like environment running. Eventually, we will have to install tflearn and tensorflow
- Instead of using OpenAI gym dependency, I will substitute regular commands in gym for specific python commands for the car and driver game. Maybe what I could do is make a module for python which has the commands pre-defined, and after I import the module I can just use that.
- We'll have to define:
    - gym.make()
    - env.reset()
    - env.render()
    - env.step(action)
    - env.action_space #The action space is only 4, although to terminate the episode, we need [enter]
    - Done (here is where the OCR comes in)

- This seems like too much to do at once so I will just try to set up the “done” mechanism, where it spits out the time when car goes off road
- The program is done, so it spits out the time passed once the car goes off of the track (not the best accuracy though)

6/30/17 (9:00-12:30)

- We have different alternative paths we can go down in this project:
    - The way we are training the game computer
        - Definitely to do: somehow measure the MPH and have an “is_moving” function. When it’s not moving, it is accumulating negative reward over time, and when it is moving, it will accumulate reward like normal
        - Maybe: Instead of ending the episode on “off road”, there are other alternatives. This is something to ponder and implement maybe later.
            - Don’t end the episode there, just have it accumulate negative reward when off road
    - Technical/methods aspect
        - Keep doing what I’m doing, using Tesseract as OCR to recognize on/off-road
        - Instead have some software on the “Car & Driver” computer that takes screengrabs and extracts info perfectly reliably (instead of relying on messy and noisy OCR), although this could be seen as cheating?
        - Just train a neural network that can do everything we need in terms of “done” mechanic, returning MPH or “is moving”
            - We’d have to take a bunch of pictures from the realsense camera and hand-label them all (although it shouldn’t be too difficult)
        - Output layer will probably be three “one-hot” output neurons: on road, not moving; 2 on road, moving; 3 off road
    - Another comment: the race begins after 3 seconds, and you are DQ’d if you start early, this could pose problems for the Q-learning mechanism?
- Next step is to take a lot of image data from the realsense camera, and I am making a program that will do that automatically…
    - I will just record myself playing and export pictures every few milliseconds, changing camera angles every few minutes maybe
    - After that I can get rid of images I don’t want
    - OK, I have taken about ~1,200 images from various angles, now we can start to load them into a neural network
- Actually, we can just use the offroad code for detecting end of episode, use in-background time-keeping for tracking time, and for the MPH we could just use the bar to the right of the MPH. (we thought of optical flow using frame differences to determine speed, but the agent would probably learn to spazz out left and right instead of moving forward). We don’t have to train a neural network.
    - For MPH, I will code that right into the fullocrtest1.py program. We could base it on the MPH bar to the right of the number by measuring the overall brightness after greyscaling it (white bar is very bright), instead of relying on wonky OCR to detect the number (it mistakes 3’s for 8’s, 0’s etc. and can’t even read single-digit readings)
- I did this and even setup an formula that approximates FPS based on brightness reading of the cropped image, but it’s not very stable at tracking FPS. We are setting up a separate computer with DOS on it, to run the Car and Driver game on a fixed monitor/camera setup so hopefully it will approximate better
- For the Q-Learning aspect, reward will be held off until about ~5 seconds or something like that, so it doesn’t just DQ itself at the very beginning over and over again, and maybe add time elapsed as an observational metric as well

- Also, instead of using the arrow keys only, we should use 1→0 number keys to control and brake on the down arrow key
- DOS is installed, now we just need to install C&D and run it

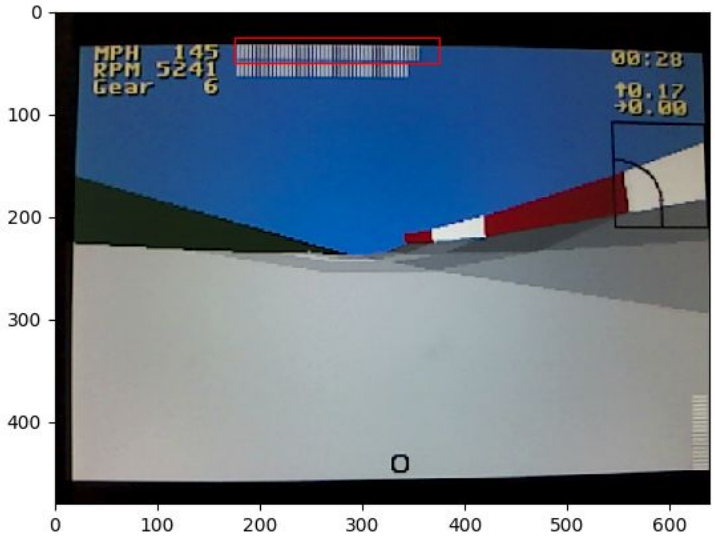
End-of-week Synopsis:
At the beginning of the week, I was not able to incorporate the episode-ending “off road” function, but by the end of the week through post-processing greyscaling I was able to reliably detect it. Additionally, I figured out a way to detect MPH to include that as an observation by measuring the brightness of the area in which the white bar could take up. I also brainstormed a few ways of setting up the reward system for Q-learning. Finally, I installed DOS on a computer and Car and Driver, but it did not work. My first step this week is to set up a permanent configuration.

7/3/17 (8:45-12:00)

- I keep trying to run command $ C&D.exe but it doesn't do anything… I think we are going to have to reinstall MS-DOS and then reinstall car and driver and see if it works then…
- Ok, we finally accomplished this by installing Ubuntu onto the old computer and then installing Dosbox, then car and driver, then mounting the car and driver folder within DOS shell, then running car and drive, commands:
    - Z:/> mount c /home/caranddriver/car-anddriver
    - Z:/> C:
    - C:/> C&D.exe

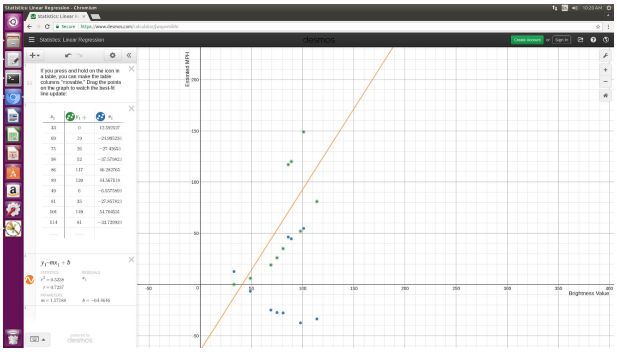
7/5/17 (8:50-12:30)

- It doesn't go fullscreen, which would probably be preferable, so I will try to do that
- Ok, I figured it out, and these are the shortcuts: ctrl + F10 to lock mouse -> alt + enter to fullscreen
- Also, to increase FPS just press ctrl + F12, to decrease prresss ctrl + F11. This just modifies clock speed in Dosbox
- I also setup the intel camera so that it is looking at the monitor, so permanent setup is pretty much complete
- Now we can setup a mph counter which runs on a function of the brightness of that little square where the bar can fill up (coordinates: (175,25,375,45))



-

- Ultimately, our goal is to have the agent traverse the course without going off road. As a result, we want to have the agent avoid staying still (because that's boring) and going off road. To know whether it is off road, we have already made a program that detects that using OCR. For the MPH, OCR is not reliable at all, so we decided to use the white bar to the right, since it should reduce larger deviations from ground truth and be much more reliable
- I plotted out the Brightness Value vs. Estimated MPH on Desmos with a linear regression, but it turns out it didn't do very well, with an accuracy of +- 54 at most, which is NOT good considering this car has a top speed of ~150. We should aim to have this very close to ground truth.



-

- Another way of estimating MPH from the brightness rectangle is to instead of recording a scalar brightness value which is more susceptible to noise, make a “curve” of brightness across the cropped rectangle. Wherever the curve drops off is the pixel value that indicates estimated MPH
- Maybe I could split up the MPH rectangle into maybe about a dozen sub-rectangles, where each has a brightness value. I could have the program start looking at the rightmost sub-rectangle and iterate towards the leftmost sub-rectangle until it runs into a sub-rectangle whose brightness value passes a certain brightness threshold that I pick. Which rectangle it runs into first determines the estimated MPH.
- Since the MPH rectangle has a delta X of 190, I will use 10 sub-rectangles.
- Ok, I made a program that returned the brightness of the 10 rectangles, and non-activated brightness is ~55. Activated brightness is around ~130. Threshold should be around 90.
- Actually I’m just going to have a measure of change, not a threshold, because it would be more robust
- Ok, it is done, and the new program is on gist

7/6/17 (9:00-12:00)

- I'm getting rid of the time rectangle in the program because it is useless
- Now I'm going to see if I can kill time to see if I can get a constant video feed/stream instead of a one-time thing, because that would make the process a bit better. Seems like this is too hard, since our team used OpenCV and that is convoluted to install so no hope
- Ok, I will start to blend my program into the one on tflearn examples
- What Dr. Hahn suggested is to start with the example one that works and strip it away until it's left with the core pieces. Then keep checking to see if it runs on Atari, and once it works just switch out the pieces for the car and driver
- So I have started to delete stuff and functions, etc. to see if I can simplify it down
- I have marginally reduced this program down, because it is very convoluted and I need to understand it first

7/7/17 (9:00-12:00)

- I simplified the program by taking out the “build_dqn” function and others, but when I ran it the reward output and Qmax output was definitely not looking okay. I’m going to reduce this down step by step and run it at every step on tensorboard to make sure
- Thread should always be 00, Step should always be going from ~700 up about 1000 every run, reward should be around 100-400, Qmax should be ~0.05, Epsilon should start at around .99 and slowly move down to .98, Epsilon progress should be 1-Epsilon
- I’ve restarted the computer and rerun the program but can’t get tensorboard to show values...
- I’ve reduced it down again and it’s functioning properly thus far. I also took out the final epsilon function and replaced final epsilon with a constant 0.1. I’ve also taken out other functions, updated on the Gist.

End-of-week synopsis

In this week, I completed all of the requirements to make the diagram on the first page of this document complete. I finally finished the OCR and MPH mechanism, which meant I could finally get to work on the Q-Learning algorithm part of the project. This part of the project will probably not be too hard, since I don't actually have to design something new. This is taking preexisting code and repurposing it for this particular scenario (optical input stream, no OpenAI Gym, etc.), and seeing if it learns to improve under these more noisy/less discrete inputs. By the end of the week I had simplified down the program substantially, keeping the parts that are important. The purpose of this is to be able to explain every line of code and fully understand it. I will continue to do that this week to come, and when the "simplifying" part is complete, switch out lines of code that use Gym for lines to read input from camera, calculating rewards for Car and Driver, etc.

7/10/17 (8:45-12:00)

- I cut out a bunch of lines by removing the evaluation functionality of the q-learning program
- Then I took out the “build_summaries()” function and just put it into the “main(_)” function
- Now I’m taking out the n_threads feature, because it’s not necessary to have arrays with only one element in them
- Also, talked to Elan about the whole Q-learning thing that I did before and this project, and he suggested to keep things simple and have a baseline first before moving onto Car and Driver. Instead of using Pacman to test the code, use breakout because we know it should be able to learn on a more simple game like that. Then try breakout on the DOS machine and get the loop going to verify that it can work, and then switch over to Car and Driver.
- Dr. Hahn helped me reduce this program down farther and farther and got it down to ~360 lines, although Tensorboard stopped working, so we are trying to build it up from the ground up.

7/11/17 (9:30-11:30)

- Got tensorboard baseline working (on Gist)
- I took out all of the tensorboard functions on the program, reducing the entire program down to 263 lines instead of ~460 which was the original source code
- Now we will start to build up tensorboard code
- I started to build up the tensorboard code, which did not work, I will continue this tomorrow

7/12/17 (8:50-12:00)

- I reduced the program even further by deleting an entire function (learner_actor_thread()) and putting its contents right into the main(_) function
- I also made the tensorboard code not throw me an error, but it's been a struggle to actually have it show up on tensorboard itself
- Finally! Tensorboard is working! It turns out that when I was executing the tensorboard command, the logdir flag needed a / before the entire path
    - The tensorboard command has to be: tensorboard --logdir=/tmp/tflearn_logs
    - Tensorboard has to be one word, two -- for the flag, =, and the / infront of path
- It only plots reward, which is probably fine. I tried adding in Q-max, but it kept throwing me a data type error, so I'll just keep it like that
- I should probably keep pruning this program down further, then commenting by it in Jupyter Notebook
- I realized that this code has an epsilon annealing timestep of 400k, while it takes about 20M steps to completely train, so I changed this annealing timestep variable to 20M to hopefully aid the learning process. https://github.com/coreylynch/async-rl Shows that it takes about 20M steps.
- I set the code running, then I started the jupyter notebook file and commented until the Deep Q-learning network, which I don't really understand how it works. I'm going to learn about how tflearn builds these deep networks with the titanic example, then I will figure out how to explain that Deep Q network

7/13/17 (8:45-12:00)

- I left this https://gist.github.com/rodrigo-castellon/897b92a1a9ba548274e1aae53dd25eaehttps://gist.github.com/rodrigo-castellon/897b92a1a9ba548274e1aae53dd25eae running for 20M steps, and it still didn't learn, which is very concerning. Now I'm leaving this https://raw.githubusercontent.com/tflearn/tflearn/master/examples/reinforcement_learning/atari_1step_qlearning.py to run for 20M steps to see if it learns there.
- If it learns, that's good, and that means we did something to the code somewhere along the way that broke it somehow.
- In case it fails, again we are trying different examples of Q-learning like the one on here that is at least functional (https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0) to use it with the final control system
- Seems like it's not learning, I started using this code: https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df

7/14/17 (9:00-12:00)

- I tried running this code https://github.com/gtoubassi/dqn-atari which claimed to be able to almost reach DeepMind's scores, but it never actually learned anything, stayed around ~1 point on breakout
- Now I'm trying this code https://github.com/DamiPayne/Reinforcement-Learning-Game-AI#dependencies on space invaders. Maybe space invaders will allow it to learn???
- While that's running, I'll try this code which is a direct implementation of the DeepMind Human-level control paper (https://github.com/devsisters/DQN-tensorflow)

End-of-week Synopsis:

This week I explored the q-learning algorithm implementation side of things for the first time. It turned out to be a lot more challenging than I initially thought to find functional code that converged as well as the ones in the official published paper. I went through, installed and ran multiple implementations (listed below) of the DQN by Deepmind without modifying any code, and none of them seemed to converge. All runs seemed to hover around the same score without learning! According to OpenAI, a non-profit founded to help democratize and research AI, many implementations have small bugs or deviate from intended interpretations of papers, which can render hamper the performance of the implementation and produce sub-optimal results. This could have been one of the reasons why these implementations failed to learn, although I'm still not quite sure what happened.

This weekend Dr. Hahn found a new OpenAI open-sourced project called Baselines, which was created in an effort to reproduce published RL results. This was exactly what we needed! I installed it at home and ran the default cartpole example, which converged very well! (reached top score consistently) Although I was not able to run the Breakout version (installing gym atari has been a pain), this shows some promise as the final implementation for my personal project.

Dr. Hahn also suggested that we should use a supervised learning approach and build an apprenticeship network to teach the car how to drive, then use the "correct" network to help train the Q network and add in a "policy" output neuron. This way, we would be training policy and q-values from the same network. Dr. Hahn thinks this will converge much quicker than just blank RL. I'm not too clear about how this would exactly get implemented, but we might use some code from the rover at MPCR.

Github repos to avoid (or possibly has bugs or untuned hyperparameters):
https://github.com/tflearn/tflearn/blob/master/examples/reinforcement_learning/atari_1step_qlearning.py
https://github.com/coreylynch/async-rl
https://github.com/gtoubassi/dqn-atari
https://github.com/DamiPayne/Reinforcement-Learning-Game-AI#dependencies
https://github.com/devsisters/DQN-tensorflow: This is an example of an implementation that deviated from the papers' details (it was not completely clear), although that issue has been resolved

7/17/17 (9:30-12:00)

- Installed OpenAI Baselines on MPCR Lab computer, but couldn't run because of OpenCV, and even when it was installed because it was opencv 2 instead of 3 and python 3
- We could use a docker container that has python 3 and opencv 3 to run the baselines code, but we need to fix our docker first (a student in the lab who's an expert on Docker can help)

7/19/16 (8:30-12:30)

- We were not able to install docker on any of the two computers
- Now we're trying to run the intel realsense code, and it worked fine!
- Dr. Hahn suggested to use the rover code because he wrote it himself and the rover works, then substitute the camera feed into the rover feed
- Since Dr. Hahn is busy, I'm connecting both computers with arduinos. I only got to connecting the arduinos up, but was having trouble with uploading programs.

7/20/17 (9:00-12:00)

- So now we're reverting back to the very beginning of the project to complete the loop (We will come back to the software/RL part of it in a bit)
- Dr. Hahn wants for a human to train a policy network first (ideally this would happen in a real scenario; no one would want a Q-learning algorithm to start from scratch in a factory/flood control system. We need an adequate baseline to improve from), so we don't need to worry about the Q-learning part just yet. We should capture some data from human drivers first.
- I got the Python Arduino Command API Code uploaded. It took a few tries, but what I did was this to launch Arduino IDE as root, copied and pasted from here into the Arduino IDE, set appropriate "board" and "port" etc., uploaded and pressed reset button.
- Then to run a python program using the Arduino API, for some reason it wasn't recognizing the board port, so I had to write this: board = Arduino('9600',port="/dev/ttyACM1") instead of just this: board = Arduino('9600')
- Ran this program which was modified from this, and what it does is incorporate more controls in more comfortable controls (arrowkeys + shift) for training the policy network
- Completed the loop! All I need to do now is make it a bit smoother so it doesn't spazz out when I hold down a key. I think this is happening because of the Keyboard.releaseKey() command that I make every loop. Maybe I can modify it so that there are no "spazzes"
-
-

7/21/17 (9:00-12:00)

- I also noticed that when I played the game through the training input keyboard, after about a minute the game computer would suddenly start just holding down the up arrow key, even after unplugging the arduino.
    - I tried to play the game normally without any arduinos and it didn't do this, so I know it had to come from the arduino. I think it was because of the Keyboard.releaseAll() not working or something?
    - What's also strange is when it did that it only did it in the game, and didn't press the up arrow key outside of the game
- Another issue: Whenever I press two keys then let go of one of them, the other one isn't detected anymore. I need pygame to detect which keys are pressed at every timestep, no matter what happened before
    - I modified my controltrain program to do this, but it still spazzes out too much
- Finally works perfectly! Very small lag, no spazz, very responsive. Programs are here and here
- Time to get some data for training a policy network.

Skipped a week to go away to visit colleges…
7/31/17 (8:50-11:00)

- Alright, first thing to do is train a policy network, which is a supervised learning algorithm that will learn to mimic human actions on the car and driver game.
- First we will generate a dataset of just me playing the Car and Driver game, and at each instance/timestep take a picture of the game screen and the human action input a
- I made a new file called “controltrain1.py” that will be the program to create this dataset
- Program features:
    - Allow me to press a key to start and stop data collection and provide feedback (through terminal)
    - On start of program, create a folder called “policydata” if it’s not there already
    - At every step of the loop, add an H5PY file
- Finished the data collection piece of the program and updated to the github repo, so now it records 100 frames of playtime and saves it to a file in a folder “dataset”, which can be played back with “dataview.py” which was taken from some rover code

8/1/17 (8:50-12:00)

- Now, what I want to do is to have an ongoing loop (not just 100 frames) and have a data collection on and data collection off switch (spacebar)
- It works somewhat well, but it's very buggy and it slows down. This is probably because it's using a lot of RAM by continually concatenating into this really long numpy array, which makes it run very slow as time goes on
- Fixed this by resizing down the image by half on each dimension and greyscaling it. Now it takes about 2 minutes to slow down to 0.2 seconds per frame
- Added in printing out to terminal function in dataview.py that helps me visualize better
- The track I'm using is called Dobbs Raceway, which takes about 40 seconds to complete a lap
    - Monterey: 1:20-1:40
    - EA Speedway: 1:00
    - Oval Speedway: 1:40
    - Arkansas Route: 4:50, 3:00, … Should use this one
    -
- To start data collection, I should have it collect about 2 minutes of data, then save to disk, then collect two more minutes, then save…
- Actually for data collection I also need MPH, which I will work on tomorrow

8/2/17 (11:00-4:30)

- Added in MPH feature, saved as ‘Z’ in dataview.py
- Next step is to modify the program to create batches of saved h5py file every ~2:00 minutes
    - Specifically, have an ongoing infinite loop
    - Then have a 100 frame recording batch timeframe that you can toggle on and off
    - After 100 frames of collection, start another batch
- Added in and updated on Github
- Now updated dataview.py to support multiple dataset files
- All I need now is an “erase last 10 seconds” or “erase current batch” feature, have data collection save to new files and not overwrite every time you run the program, and maybe audio cues to let me know when file is going to save
- Did the audio cue and the no overwrite feature, and updated on Github
- Turned out no overwrite feature didn’t consider some edge cases, and could overwrite, so I ironed that out and now it shouldn’t be able to overwrite
- Next time I’ll put in the delete collection file feature, then collect data

- Side note: I was talking with Dr. Hahn about other directions this project could take, and he said we could apply this same algorithm to a simulation of a nuclear power plant or something like that, and we looked up a DOS power plant simulator and found one that could be used in the future (http://www.myabandonware.com/game/the-oakflat-nuclear-power-plant-simulator-3vp)
- This could be really useful to show people the applications my project could have in a concrete way.

8/4/17 (8:50-12:30)

- All that's left to collect data is now the delete collection file feature
- Added in and now committed on Github repo
- It's slowing down very fast again… Need to correct this
- I took out the line concatenating the images to the dataset and it didn't slow down, so it has to be the images that is slowing it down somehow
- What we could do is just save more often in the middle, clearing out the memory
- Actually it's not the RAM and it's probably just a problem with the python numpy code, where concatenating onto a really big array just gets slower
- Solution is just to save to disk more often, like every 10 seconds or so
- It takes about 0.15 seconds to save to disk, which is not really that bad
- Actually what I can do is concatenate separate arrays each lasting 10 seconds, then at the end concatenate them all at once into the data class!
- Psuedocode:
-
- Initiate a sub_batch list [] containing 5 np.zeros((1,240,320,1))
- Initiate current_sub_batch integer as 0, which denotes which sub_batch index to concatenate on
- For each batch loop:
    - Concatenate onto sub_batch_list[current_sub_batch]
    - and every time frame_num is a multiple of 100, I'll perform the following:
        - Current_sub_batch += 1
- Concatenate sub_batch_list[0], sub_batch_list[1], sub_batch_list[2], sub_batch_list[3] sub_batch_list[4] onto d.images

End-of-week Summary:
At the beginning of the week, I had gotten the full control loop up and running. This was good news because then there was two steps left for training a policy network: collecting data and training off of that data. This week I finished writing the program for the collecting data piece, which allows me to easily store data onto .h5 files as greyscale images and their corresponding key presses, as well as MPH. On Monday I will collect all the data I need to train the policy network.

8/7/17 (9:00-12:30)

- I collected about 30 batches * 500 frames = 15,000 frames of data (15,000 frames * 0.05 seconds / frame = 750 seconds = 12.5 minutes of data)
- Turns out there is an issue with RAM when I try to load in too many files at once, so I can only load in a few at a time to view the files
- Also, there are some blank frames scattered throughout because of the way I stored the batches, but it shouldn't affect the performance of the neural network
- I'll use 11 different neural networks to train on the data, and I'll use some of the code from the rover at the MPCR lab
- I've run into some errors, I will continue modify the program to correct these tomorrow and hopefully finish training

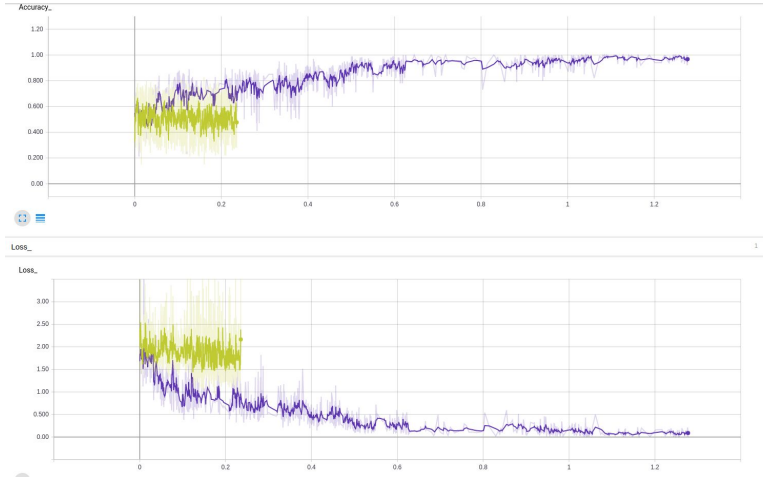
8/8/17 (8:30-2:00)

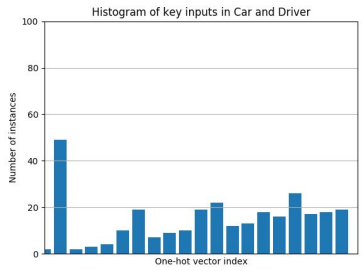
- Now trying to get training working, but to no avail...
- It says that there are too many image frames for the number of action frames (because of the extra blank frames)
- I made a program that deleted the blank frames (deleteblankframes.py), and it did delete them and made a separate file, which I verified did have less frames
- Finished writing the policy_train.py program, and ran it on tensorboard with alexnet, and it was not converging at all
- Then I tried using the DNN and the loss function seemed to decrease over time, which is good news, but the accuracy is still stagnant
- Eventually the DNN reached and plateued at around 0.6 loss, which basically means that we squeezed everything we could out of this neural network
- We need a bigger network to compensate for the complexity, so I tried Alexnet and GoogLeNet, which didn't work and threw errors… I will try to resolve this

8/9/17 (8:45-12:00)

- Now I'm trying the Conv1 network, which is slightly more complex
- I tried it with 350 training batch size, which caused a memory error (this shouldn't happen… I have a GPU with 8GB at least VRAM), but I reduced it down to 10 and see what happens
- In about a minute, loss has decreased from 0.8 to 0.66, which is good, but it has been plateauing there…
- Conv1:
    - 50 Batches
        - 5000 epochs
        - A
    - 100 Batches
- Next step: Since I don't know how to interpret the accuracy with binary cross entropy, we should switch to one-hot and compare…
- I wrote a function that took one label instance from an h5 file and converted it into a one hot
- Added to program and updated on github
- Changed batch size to 100 and broke it… Fixed with changing Z dimensions…
- Now didn't anticipate a key combo '[1 0 1 1 1 0]' or others, so added them
- Also memory issues and I got the type for each pixel and it was a float64, which is way too much… it should be probably float16
- Tomorrow I will make a program that will convert all the pixels from all the frames from all of the datasets into float16 then run it through the same policy_train.py script

8/10/17 (8:50-3:00)

- Today I'm going to change all of the pixels of all of the datasets from float64 to float16 to reduce memory consumption so I can run more complex networks with more efficient batch sizes
- Actually since all of them are between 0 and 255, I'll just use uint8 instead, which rounds to the nearest integer
- Converted with program now uploaded onto Github
- It worked! Now I'll try to run maybe Alexnet or GoogLeNet with 100 batches
- Running Alexnet! Accuracy starts at about 50%, but expected to start at around 5% (random chance), so Dr. Hahn suggested to make a histogram of the labels to see if it is biased towards the "up" arrow key, for example
- Reduced learning rate from 0.0001 to 0.000033 and it converges much better! 1.8 loss all the way down to 0.15 loss and continuing to go down!
- 



- So the up key is very biased toward (index 1), and then followed by up + shift + right and up + right
- Maybe I could test the output from the network and make a histogram of that too
- First I'm going to leave GoogLeNet running in the background
- GoogLeNet did not converge well, looks like we need to reduce learning rate again by x3

8/11/17 (8:40-12:00)

- Now I'm trying to run GoogLeNet with 10x the last learning rate and seeing if it converges faster
- I still need to investigate why each model is starting out with ~40-60% accuracy, which should NOT be the case
- Somehow I need to run data through untrained and trained network and see if there's a difference in the histogram plot between the untrained, trained, and just raw labels
- If trained and labels has a difference, that is not good and if untrained and labels or trained has no difference that is also not good
- GoogLeNet is not doing to well… Seems like AlexNet is the best for this task. It converges the fastest and the best
- Now I'm running Alexnet with 200 batch size, 0.000033 learning rate, and it will save to a text file the action output prediction log.

8/15/17 (8:40-4:00)

- For some reason, someone deleted the Alexnet results?!? Running again
- It seems like the validation isn't doing too well… maybe adding in random noise or something to add data augmentation, since it seems like it's overfitting (validation and training curves diverge)
- Also I created a simulator program for a nuclear reactor, and Dr. Hahn suggested to go research the diff-eq for control rods, etc. and implement that in.
- I tried with random noise as data augmentation, but it was still overfitting??
- I added in extra features on the simulator program to allow two variables: one you control and the other that is dependent on the first one
- Maybe it's not actually a problem… I'll try to test the model on the actual game

8/25/17 (3:00-6:00)

- Now that that the policy network model is trained, I can now run it on the C&D Game using policy_run.py
- I further adjusted parameters in policy_run.py to my setup, and now it's running but is only predicting 6, which I need to see why that is the case.
- I outputted the predictions and they stay the same no matter what the image and those same outputs stay the same somehow in every time I run it
- I figured it out, turns out self.image which is what it takes from the camera is just a zeroed out ndarray, so it's not getting any image at all
- I will fix this soon. So all this needs to go to a full working autonomous version is:
    - Proper camera stream input into the neural network model
    - Arduino output onto the game (this will involve again decoding the index of the one-hot into the actual arrow keys to press, etc.)

9/1/17 (3:00-6:00)

- I'm also trying to fix my computer. We got it to boot into ubuntu, but it got stuck in the login loop, so I'm cloning another drive over
- Now I'll try to finish up the run script
- I forgot to bring my realsense camera, so I'm going to try to use a webcam instead
    - It runs!
- I got the computer to boot up. Now I'm moving the project folder to it.
- Home computer all set up! Now just returning transfer USB contents
- The webcam camera runs perfectly and predicts at real-time!
- The only thing left is to send it through the arduino loop
- I made a function that returned the prediction in a form that the arduino code can read (dictionary)
- Obstacle: for some reason the arduino code to connect to the board needs to run as root, but running as root throws an error when importing tensorflow
- I tried to setuid in network_run.py but it didn't do anything, I will try other ways to run as non-root
- Hopefully now that I have my computer working I can work on this on home

9/20/17 (2:00-6:00)

- When I got here everything was unplugged
- 4:30: computer is plugged in, setup is almost complete
- I also tried to clone over my lab computer hard drive over to mine but the cloning toaster shorted, so I'll have to wait until Friday for a new cloning toaster to arrive at lab
- Still trying to get the policy_run.py program to run, but it doesn't because of root and user-run complications
    - Tflearn can only be imported as a user
    - The arduino board can only be setup as root
    - Requires me to switch from root to user, which I am using os.setuid(1000) for, but it doesn't work
-
-

9/22/17 (4:40-6:30)

- Still trying to figure out the root user->user issue
- Found out a possible way out!
    - The error came from the Arduino module for python and it's an issue with it not having root permission to access the /dev/ttyACM0 port (https://stackoverflow.com/questions/27858041/oserror-errno-13-permission-denied-dev-ttyacm0-using-pyserial-from-pyth)
    - So I'm gonna have to grant it access with chmod 666 /dev/ttyACM0 or a more sustainable solution
- Ran the policy network on the actual game, didn't do as well as I hoped but it did definitely learn how to drive!
- Next step is to implement Q-learning into the loop and have it learn from experience

9/29/17 (1:30-6:00)

- Goals today:
    - Collect more data from human player, most importantly include instances where human player recovers from mistakes
    - Retrain on that data with different neural nets. Also try neural nets without pooling (pooling is very good for image recognition, but not as good for gameplaying agents), or just the one that deepmind used for its atari paper.
    - Think of a way to quantitatively assess performance (score measure)
        - (Time on road)/(total time)
        - Seconds until car leaves road
- 1:30: collecting data
- 2:15: training on all data (new and old) with alexnet (test1)
- Turns out I forgot to delete blank frames, so deleting blank frames now
- 4:00 Training on all data (test4)
- I stopped training at around step 1.2k (1 hour), ran it, and it stays still a lot. It can get into a rhythm where it stays at ~60 mph sometimes, but mostly it spazzes out at 0 mph.
- Next steps to optimize:
    - More data
    - More training

10/5/17 (4:45-6:30)

- Collecting more data today
    - I'll collect data of me going from off road to on road especially
- The data might not be completely valid/precise. There is a slight delay. Update: that is actually just the OpenCV streaming feature, there isn't a discrepancy.
- Starting from dataset50.h5 is the offroad to onroad data
- Actually instead of teaching it to fix mistakes, I'm going to have it press Q whenever it detects green on the bottom of the screen

10/13/17 (4:00-6:30)

- Dr. Hahn said that I should get students to collect more data
- I created the setup
- I'm also bringing home a computer that can be used as the simulation computer so I can collect data at home

10/27/17 (4:30-6:00)

- Working on abstract/intro. I finished the first part of the intro rough draft
- Also finally copied over lab computer drive to my drive so I can collect data at home.

11/1/17 (9:30-10:30)

- Today's goal was to try to run on all of the new data collected by the FAU students!
- Some of the data was weird so it didn't go through correctly but I found the exact datasets that were the problem and fixed it

11/2/17 (4:30-6:30)

- Adding onto the Intro
- Dr. Hahn came and we brainstormed some ideas/applications my project could have and vision, so I'm going to implement that into the paper

11/17/17 (3:00-6:30)

- Took some more data.
- I decided to take data on a simpler track because it would be easier to train the network and it would take less data
- Policy Network Evaluation Automation:
    - Have a script like policy_run.py and have it run
    - While True:
        - Wait 2 seconds for time counter to count down
        - Start time
        - Let policy network run normally until offroad
            - At every timestep check for off-road
        - Take time at offroad and log into file
        - Press [Enter] key through Arduino

11/21/17 (4:15-)

- Writing program to automatically evaluate neural network called policy_eval.py

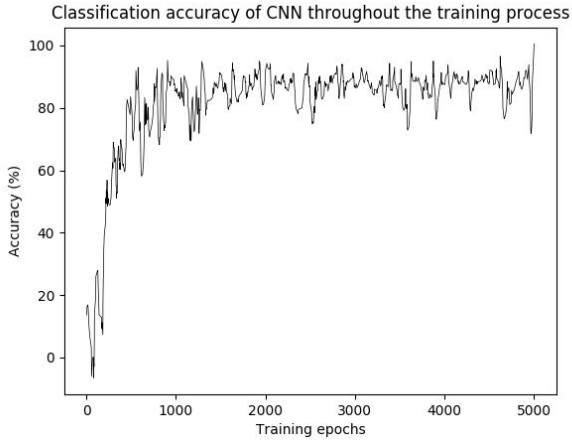
11/23/17

- Took more data (now ~7GB in total), looks like it's improving and has learned how to drive better than the last version
- The rules-based OCR detection mechanism (for going off road) will not work because it isn't good enough to filter out false positives and get it right all the time
- I'm going to train a separate neural network that will do this task, as well as output the miles per hour on the screen (I'm going to call it eval_collect.py, eval_run.py (to see if it works?), eval_train.py)
- It's going to be a hand-labeled dataset
- From an image, it should output: [mph, is_off_road] where mph is an integer and is_off_road is a boolean (0 or 1)
-

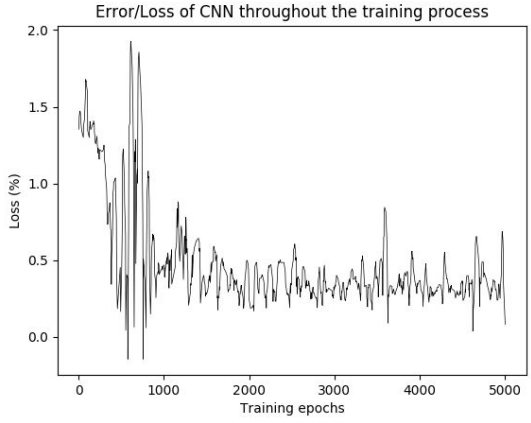
- Also going to simplify the data collection process in general. Right now it works like this: press space bar -> collect 500 frames -> disabled until double press spacebar -> collect 500 frames -> …
    - Want it to work like this: press spacebar -> collect... , if you press spacebar again stop
- Done! Updated on Github (also added in pygame interface to make it easier)

12/8/17 (2:30-6:30)

- Talked to Dr. Hahn about the project in the long-term, decided to start working on a nuclear power plant simulator environment in NetLogo because I think this will be a more effective presentation of what the algorithm is capable of being applied to. I think I'll be able to get this into the paper because I can work from home and it's just a matter of adapting my old configuration.
- Plotted some new results/figures!



-



-

12/15/17 (10:45-3:)

- Now I'm going to try to implement my control system on a nuclear reactor sim
- I need to do the following:
    - Write a script that takes the CSV output from netlogo and saves it into a temp .h5 file containing actions for each frame and the frame number.
        - done
    - Write a script that collects image frames and the frame number saves it into another temp .h5 file
        - Working on the OCR all day and couldn't get it to read the numbers accurately, I think I should just do it manually. It actually should be pretty quick to hand label them
        - Actually I'll use this: https://www.pyimagesearch.com/2017/02/13/recognizing-digits-with-opencv-and-python/, which explains how to automatically crop it and then do OCR on that
            - I couldn't figure out how to crop the rectangle I wanted so I'll just figure out a good crop rectangle
        - I'll try to improve the OCR that I have right now
    - Write a script that combines these two .h5 files into one
        -

June 4th: (8:45-2:00):

Progress today:

- Took video of Pine Crest campus (pre-recorded)
- Took video of FAU campus (10,282 frames, ~5:30 minutes of footage, walked near the lab around campus with multiple angles at each position)
- Generated image sequence from both videos on computer
    - Took hours because of large size of files, then hours to transfer
    - VisualSFM on windows only accepts .jpg, so I had to transfer again, but the external drive stopped working
- Talked to Dr. Hahn about my personal project for Econ, and he suggested writing it up and publishing it in an econometrics journal
    - Will be doing that as well

June 5th (8:45-1:30):

- Got all of the images onto the windows lab computer
- Trying to generate SIFT and 3dsMax files from these images, but can only do one at a time
    - These files are necessary for the point cloud/image labeling process
    - Clicked something and it started outputting “SIFT: 0372, 1920x1080, 3512, 0.05sec” where the number after SIFT went up incrementally and the number to the left of 0.05sec changed randomly from around 200 to 8000
    - I’m going to wait and start writing my paper until it reaches SIFT: 10282 (the last image)
        - This week I’ll also be reading other econometrics papers to get an idea of conventions (structure/format)
    - The computer froze at 10:37 and has not unfrozen since, so I will repeat (now I have to wait to ask someone for the password to the computer since it locked)
- Trying on a Linux computer with a docker image instead
    - Ran it on 100 images from the hallway, generated an output with points with coordinates (10 different numbers for each image (??))
    - Download docker: sudo docker pull ryanfb/visualsfm
    - Put images in /home/DeepInspect/sfm
    - Commands to execute:
        - Sudo nvidia-docker run -ti -v /home/mpcr/DeepInspect:/home ryanfb/visualsfm /bin/bash
        - root@a5c1fec29718:~# cd /home/
        - root@a5c1fec29718: /home# cd sfm/
        - root@a5c1fec29718: /home/sfm# pwd
        - root@a5c1fec29718: /home/sfm# VisualSFM sfm+pairs+pmvs ~/home/sfm/ BS12_4th.nvm @8
    - Should output BS12_4th.nvm in that folder
    - VSFM on Matlab -> put in the file, etc.
    - Tried visualizing in matlab, but it was gibberish visually
- Found opendronemap, seems newer: http://opendronemap.org/
- Found regard3D: http://www.regard3d.org/index.php/documentation/details/project-tree
- Found COLMAP (for windows)
    - Ran it on a windows computer, seemed to be working
- Found BigSFM
- Found https://github.com/openMVG/awesome_3DReconstruction_list
- Met Connor (the blind student at the lab) and recorded the route from the lab to his dorm building and back for the project
- Project name ideas:
    - “BlindEye”
    - “Copilot”
    - “Navigator”
- Also considered eventually deploying the project on phones (more widely available) and necklace-mounted with the camera facing outward
- Advice/suggestions from Connor:
    - If this were an app on the phone, do NOT use vibrations because they use too much battery

        - On phone, the most useful way of giving feedback is a voice that would come through one of his earbuds
            - Many times he gets completely lost because if you start walking in a slightly different direction, after 100 steps you will be in a completely different place compared to where you thought you would be
            - This is why it is important to have a guidance system
- As far as I know, no one has made something like this, closest I have found:
    - https://github.com/chekoduadarsh/Fast-Blind-Guidance-System-using-Deep-Learning: a combination of many different functionalities (OCR, image recognition, facial, expression, tell time, etc.) but no navigation
    - www.mdpi.com/1424-8220/17/6/1371/pdf: obstacle detection (off-site computing but capturing from a phone)
    - https://itunes.apple.com/ca/app/talking-goggles-camera-speech/id602999586?mt=8: image recognition app for blind
    - https://gowithfloat.com/2016/10/9-outdoor-navigation-apps-people-visual-impairments/
- Also thought about other functionalities this project may have:
    - Help new college (or high school) students get around campus and implement this into their schedules
    - College tours, tourists in cities, navigating the area and where to go based on what their phone sees from mapped landmarks, etc.
    - Add GPS tracking to provide a “safety net” for the neural network to predict within
        - So the neural network would make the miniscule location tracking, while GPS tells you the general area (several meters radius, which is not necessarily helpful for a blind person trying to find a specific room, etc.)
    - Make a web interface where people can add video and adds it to the database

Econ paper:

- Read two papers

June 6th (9:00-1:45):

- Going to try running the new images of the hallway (hallway-4) the same way I ran the 100 frames of the video and visualize on matlab, maybe that will give me results
    - If it doesn't give me results, that gives me strong evidence to believe that this method is very unreliable
    - Bad result, so I shouldn't use visualsfm from docker to generate the .nvm file
- Meanwhile COLMAP has been running on the windows computer at the lab but will take ~2-3 days to finish
- Going to try docker for opendronemap (https://github.com/OpenDroneMap/OpenDroneMap/wiki/Docker)
    - Started running it by this command: docker run -it --rm -v "$(pwd)/images:/code/images" -v "$(pwd)/odm_orthophoto:/code/odm_orthophoto" -v "$(pwd)/odm_georeferencing:/code/odm_georeferencing" opendronemap/opendronemap --mesh-size 100000
    - First had to put all of the images in the /images folder
    - It worked, at least in generating a point cloud mesh!
        - Now trying: docker run -it --rm -v $(pwd)/images:/code/images -v $(pwd)/odm_georeferencing:/code/odm_georeferencing -v $(pwd)/odm_meshing:/code/odm_meshing -v $(pwd)/odm_orthophoto:/code/odm_orthophoto -v $(pwd)/odm_texturing:/code/odm_texturing -v $(pwd)/opensfm:/code/opensfm -v $(pwd)/pmvs:/code/pmvs opendronemap/opendronemap
        - This will get all of the "intermediate outputs" which will hopefully include the coordinates/directions of the camera at each image
    - Success! In the /opensfm/reconstruction.json there is all of the information in a nicely formatted and labeled JSON tree for each image!
        - It even outputted a reconstruction.nvm!
- Now what I have to do is run OpenDroneMap on the bigger dataset that I recorded on the route to the dorm building
    - Started running the ffmpeg conversion command on the video (ffmpeg -i IMG_6840.MOV -r 20 output_%04d.jpg)
    - Moved it over to the linux computer (to generate labels)
- Trying 100 images first to see how long it takes
    - Since this program only uses CPU, it will be far too slow to run the 10,000 image dataset
    - Need to try more options

June 7th (9:00-4:00):

- BigSFM is for unordered image collections
- Will now try OpenSfM
    - https://github.com/mapillary/OpenSfM/
    - Seems like it doesn't support GPU
- Trying visualSfM again
    - Actually using it on Linux using this repository: https://github.com/anders-dc/vsfm-linux-x86_64
    - So you have to go into ~/vsfm-linux-x86_64 and then type the command ./visualsfm.sh
    - It worked!
    - Since it has sequential image pairing functionality, it is also much faster
    - Ran 500 images in about 5 minutes
    - Plotted the .nvm output file using the https://gist.github.com/rodrigo-castellon/3d406ddea3815acdca2d835bfb84ef00 matlab code
- Now going to run all 10,000 images
    - Ran it but then it didn't save correctly, so I'm running it again
- When reconstructing it keeps jumping around because it loses position so we may have to use a different method of finding location
- Control System:
    - Set up camera, arduino boards, etc. √

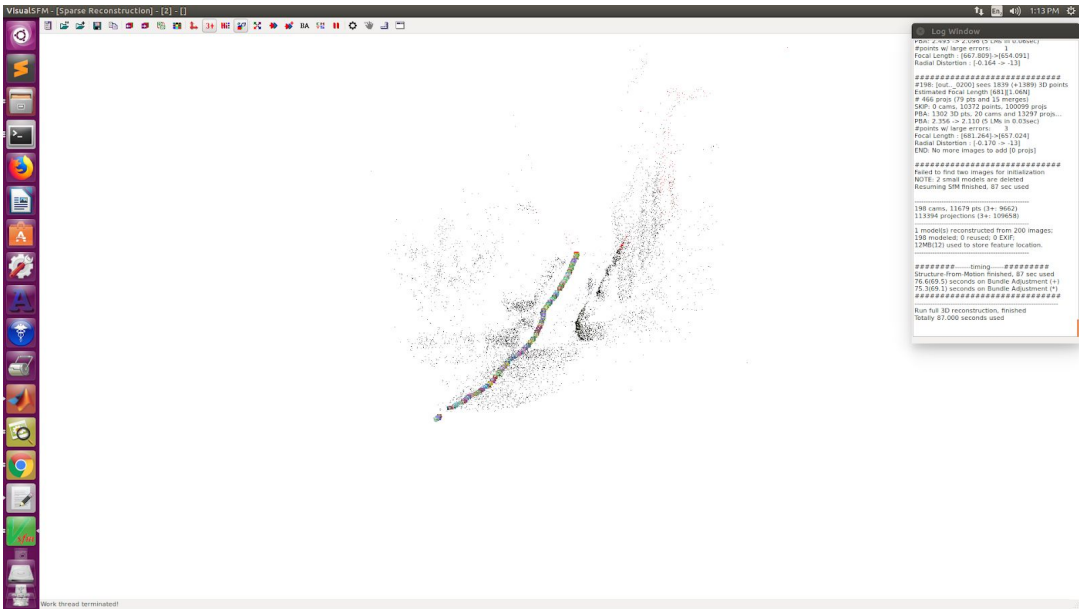
June 12th, 2018 (8:50-4:00)

- Problem: Reconstruction does not work on image sequences (keeps jumping around and thinks that it's a different scene every ~50 frames or so)
- Need to find the paper that demonstrates SfM and machine learning together to see how they did it (http://mi.eng.cam.ac.uk/projects/relocalisation/#publication)
    - https://www.unavco.org/education/resources/modules-and-activities/field-geodesy/module-materials/sfm-intro-guide.pdf
        - Says to not use video because sfm algorithms might not work well on video
        - Looking for iphone app that takes controlled timelapse 5 frames per second or so
            - Couldn't find one, maybe taking pictures manually?
        - Found burst feature on iphone, which does this but has some blur which confuses the SfM algorithm
- Took 21 pictures inside the lab space and it reconstructed perfectly!
    - Potential reasons why the other ones didn't work:
        - Glass door with sticker on it messed up the structure from motion (relatively big moving object right in front of the camera, confused it)
        - Used "compute sequence match" pairwise matching instead of "compute missing match" which compares all images to each other
            - Maybe what I should do is compute sequence match but have the sequence step/range to ~50, combine that with taking fewer frames (which could be redundant and add to the noise in the dataset) -> very good reconstruction
- Now I will take more images of the inside of the lab and add them into the dataset to see if it can reconstruct at the same fidelity or even better
    - Started at 10:21:50
    - Finished pairwise matching in 22 seconds
    - 66 images total
    - When I ran "Compute 3D Reconstruction," it jumped around again
- Reasons why it works:
    - No rotation, stable moving in direction of camera image
- Tried with video split into frames while keeping it stable and not rotating, but did not work (probably because of artifacts)
- Attempting with 284 frames of lab and hallway (taken with burst feature on iphone)
    - Started at 11:12AM
    - Jumped around, probably because of the blurring from the rotation of the iphone camera
- In nvm file output:
    - Index 5, 6, 7 represent X, Y, Z respectively (6, 7, 8 starting from one)
    - Index 1,2,3,4 represent angle/direction (2,3,4,5 starting from one)
- Tried outside
    - Spent the rest of the time importing the images, removing duplicates, running SfM feature detection matching (sequence)
- I hypothesize that once the neural network is trained on the dataset, it will be able to deal with blur, etc. (unlike traditional SIFT algorithms which look for salient corners, edges, features)

June 14th (8:45-4:00):

- Finished missing pairwise matching in 28 minutes
- Reconstruction jumps around again
- Ideas for future:
    - Automatically generate coordinate labels with neural network
- Found app: Lapse It, allows for picture to be taken at a fixed frequency
    - For iPhone and better than burst
    - Going to go through the same route and take time lapse video (10 frames per second)
- Took timelapse video and 5152 frames, transferred to ubuntu computer
- Talked to Doug, Will, and Elan about the project and they suggested to map out the shortest/easiest route to measure accuracy, then move from there
    - Say 20 ft, or something like that
    - Derive a unit conversion between the visualsfm virtual world and the real world
    - Make a measurement in the visualsfm virtual world between two known points in the real world, and compare the distance in the visualsfm virtual world with the “ground truth” distance in the real world
- Took more data (now ~7GB in total), looks like it’s improving and has learned how to drive better than the last version

June 15th (9:00-5:00):

- Started running the 5k frame one, will take ~2 hours
- Idea Will came up with:
    - Have a origin and destination system with the best route line, then if the person walks off the route, it will automatically guide him/her back onto the route like an attractor (vector field, like a water basin)
- Ran SfM on 200 initial images, and actually worked very well!
- 
    - You can actually sort of tell that this is from outside of the building (you can see a sign too) and the images are all in a line, exactly how they were taken
- Wrote code to generate dataset based on the images and npm output file (https://gist.github.com/rodrigo-castellon/5cd4d7170541bc41ece8bc82cf61a2a8)
    - Although this dataset will not be the true dataset we'll use to train the neural network, the code is now ready to generate any dataset given the images and the NPM output file
- Reading https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Kendall_PoseNet_A_Convolutional_ICCV_2015_paper.pdf this paper, and realize now that we may have to use transfer learning in order to compensate for the low amount of data we would have
    - Convnets only converge after millions of examples, so transfer learning speeds up the convergence process by bootstrapping this new problem onto one that is already largely solved
- For transfer learning, will use this: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets-2/#1
    - Going to go through the steps

June 18th (8:45-4:00):

- Today was the first day of the MPCR short course, for which I will serve as TA
    - Today was only an orientation, so I met all of the students
- Going to try to get the neural network to train on the generated dataset
    - Changed it to use pickle and not .h5, which is a lot faster, also added argv
- Fellow student also suggested “binning” since apparently convnets aren’t too good at regression? Dismissed that idea because the paper used regression which worked fine
- Reading literature about convnets used for regression, since there apparently are difficulties with using convnets for regression:
    - https://stats.stackexchange.com/questions/335836/cnn-architectures-for-regression which led to this: https://lmb.informatik.uni-freiburg.de/Publications/2015/FDB15/image_orientation.pdf (tried to regress orientation error in range [-360,360])
    - And this: https://arxiv.org/pdf/1708.05628.pdf
- Also going to simplify the data collection process in general. Right now it works like this: press space bar -> collect 500 frames -> disabled until double press spacebar -> collect 500 frames -> …
    - Want it to work like this: press spacebar -> collect... , if you press spacebar again stop
- The rules-based OCR detection mechanism (for going off road) will not work because it isn’t good enough to filter out false positives and get it right all the time
- Done! Updated on Github (also added in pygame interface to make it easier)

June 19th-22nd (9:00-4:00):

- This week was the Deep Learning short course, for which I served as TA

June 25th (9:00-3:30):

- Now I'm going to train a network on the frame/label pairs
- Got the data loaded in with the lines
    - dataset = pickle.load(open('data.pkl', 'rb'))
    - X, Y = np.array(dataset[0]), np.array(dataset[1])
- Added some image augmentation like random crop (which the original paper did) and gaussian blur (5 sigma max)
- Didn't understand exactly how to code multivariate regression
    - Saw that tflearn.regression() had the default loss function set to categorical cross entropy, so spent about an hour or two reading about loss functions, etc.
        - Now I understand pretty much exactly how categorical cross entropy works, and it is only used in classification problems
    - Commenting the VGG network to make sure I understand what my code is doing and the architecture
    - https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/
        - Followed this to check shapes
- Thinking of using VGG-16 pre-trained weights to expedite learning

June 26th (9:00-4:30):

- Started running VGG (regular because could not find pretrained weights anywhere on the internet, expect for those for the Caffe library), and took very long because tensorflow was not using GPU acceleration
    - Currently installing tensorflow-gpu
- Tensorflow-gpu installation ended up not working, so I'm using a docker image now (https://blog.sicara.com/tensorflow-gpu-opencv-jupyter-docker-10705b6cd1d) which is using python 2.7, tensorflow-gpu, and tflearn
- Tried vgg, but ended up running out of memory, I'm going to try GoogLeNet now, since it's supposed to be more memory efficient
- Been trying to run GoogLeNet, and actually ended up running it in a docker container, but couldn't get tensorboard to output graphs, so trying to do that now
    - Figured it out!
    - I opened a terminal window in jupyter and typed the following:
        - bash
        - hostname -I
            - Capital letter "i" as the flag
            - Gives you the IP address
    - Then I opened up another terminal window and typed tensorboard --logdir=/tmp/tflearn_logs
    - Then I typed in the IP address with the port (6006), and there was tensorboard
- Decided to run an alexnet as well on another machine, did the commands below to SSH in:
    - ssh -X -L 8889:localhost:8889 10.16.202.230
        - Password: asdfjkl;
    - Jupyter lab --port=8889
- Ran this program which was modified from this, and what it does is incorporate more controls in more comfortable controls (arrowkeys + shift) for training the policy network
- Completed the loop! All I need to do now is make it a bit smoother so it doesn't spazz out when I hold down a key. I think this is happening because of the Keyboard.releaseKey() command that I make every loop. Maybe I can modify it so that there are no "spazzes"

June 27th (8:50-2:30):

- GoogLeNet finished training 100 epochs after 4:38 hours, but loss did not change at all
    - Stayed at around 20
- I think the reason is because there is are few training examples (200) that it was essentially impossible for a huge network like that to learn all of its weights from just that many examples
    - At least we know how to go through the process of creating a dataset and training on it
- Read at this article from Stanford on transfer learning: http://cs231n.github.io/transfer-learning/
- Thinking about training alexnet on imagenet and then transferring the weights and training the last layer on my dataset (that's what the PoseNet paper did, but with GoogLeNet)
    - Investigating how long it will take for it to train, maybe just try to find a pretrained GoogLeNet
    - Comprehensive overview of analysis of constructing CNN thesis: https://arxiv.org/pdf/1707.09725.pdf#page=73
    - Can't find any for tflearn, but looking to see how to convert other pretrained models (Caffe, pytorch, etc.) to tflearn/tensorflow models
    - https://github.com/Cadene/pretrained-models.pytorch#inception
    - Found a vgg16 checkpoint, but I don't think it was formatted right
- I'm going to start training a googlenet on imagenet on a machine with two 1080 GPUs, then let that go while I'm trying to load imagenet weights
    - It will take probably 5 days, so decided to keep looking for pretrained models
- This could potentially be useful: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/index.html#2
    - MobileNet is used here
    - Looked through the code, I can get the .pb (protobuf) file for tensorflow but it would be hard to format it into a .tflearn model for tflearn
- Going to try to do pytorch or caffe
    - Installed pytorch on the lab computer with anaconda
    - Looking here (https://github.com/Cadene/pretrained-models.pytorch) at pytorch pretrained networks
        - GoogLeNet is not there, but AlexNet is, so I will be using alexnet
        - It worked! I put in an image of a border collie and it correctly classified it
        - This verifies the pretrained weights
- Dr. Hahn also said that he was interested in going through the process of pretraining an ImageNet model in the lab
    - So we'll have a pretrained model anyways
    - I'll be doing that tomorrow, but it will likely take ~5 days
- Figured out how to get feature extraction working, can probably do it in the same script as with tflearn
    - All you have to do is:
        - load in the pretrained model
            - model = pretrainedmodels..__dict__[model_name](num_classes=1000, pretrained='imagenet')
        - Load the image as a torch tensor

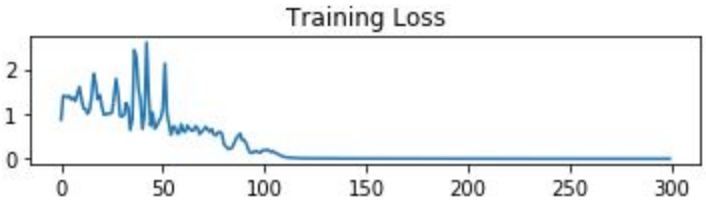
        - (which you can convert from a numpy array easily with: torch.from_numpy(numpyarray))
    - input = torch.autograd.Variable(input_tensor, requires_grad=False)
    - output_features = model.features(input)
- Output_features will then be shape: [1, 4096]
- This can be converted into a numpy array and then fed into a FCN, and we train the FCN!

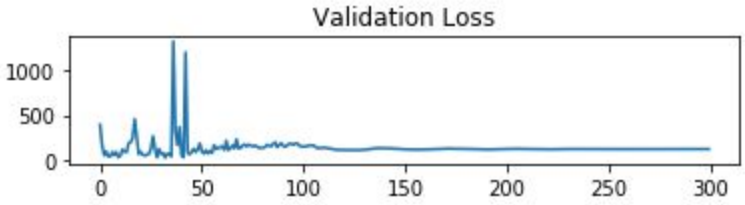
June 28th (9:00-3:30):

- Just going to use pytorch for the entire network
    - Tried that, but it didn't work because of a weird error
- Going to use the pretrained network as a fixed feature extraction
- Actually ended up training a pretrained network, resnet18, but it didn't improve too much
    - Possible reasons why/courses of action:
        - Learning rate wasn't adjusted right
        - Try a different network (VGG, Alexnet)
        - Turn off data augmentation
        - Try visualizing the input data and verify that they are all labeled correctly
        - Try passing in random numbers (instead of images) and see if it behaves differently
        - Try using a standard regression dataset
        - Check if inputs are standardized (mean = 0, unit variance)
        - Try only predicting one value instead of 7 (simpler problem)
        - Check loss function
            - !!!! I remember in the paper that they weighted the quaternions with a weight B, so that coordinates and quaternions would balance out in the loss function
        - Check weights initialization (Xavier, perhaps)
        - Try freezing the earlier layers of the network

June 29th (9:00-4:30):

- Graph for the first 300 epochs:



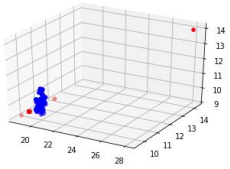


  -
  - Doesn't look too good, going to try one of the things I suggested in yesterday's log
- Tried multiple things, including:
  - Only predicting XYZ coordinates
    - First tried from raw data
    - Then tried scaling up the magnitudes of each coordinate by 1000 and adding 1000 to fight against the weights just averaging at 0 and coincidentally predicting that
      - This is what happens with normalized output data
      - Didn't end up predicting well
        - It got it in the general area, but not very well
        - Blue: real ground truths; Red: Predictions

```
In [122]: fig = plt.figure()
          ax = fig.add_subplot(111, projection='3d')

          ax.scatter(*outputs, c='r', marker='o')
          ax.scatter(*filtered_grounds, c='b', marker='o')

Out[122]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f6ef15d99e8>
```



        - It looks like its clustering into a very very precise volume, except for a couple of outliers
- Going to try to feed it a random image
  - Ended up giving me the same exact coordinates no matter what
  - This makes sense because loss began very very high on the first epoch, then decreased and stayed there for the entire training time
  - Since the outputs are very very concentrated in a small region, then the network figures out that it is just better to learn the bias and keep weights at essentially 0
  - To help solve this problem, I will scale all of the ground truth coordinates up and subtract by the mean to normalize with a unit variance (roughly)

- Should try Inception v3
    - Tried but got a weird internal model error in the weights
- Will try predicting only Z coordinate (since that is the one that changed the most)
    - Tried this but before scaling up and didn't learn anything
- Try to implement batches (might not even have any effect, but good to know)
- Other ideas based on the paper:
    - add another FC layer with feature size 2048 to increase generalizability
    - Test time: evaluate using center crop and 128 crops of the image and average the pose vectors
        - Would increase comp. Time (according to paper)
    - Should try training the network on their data to see if it's my problem our if it just doesn't work
    - After to do analysis on the network, do saliency analysis (saliency map, based on https://arxiv.org/pdf/1312.6034.pdf)
- Dr. Hahn suggested to feed the model more data and use unsupervised learning for that
    - Have an unsupervised network learn features from the dataset which reconstruct images precisely and sparsely
        - Feed it the feature linear combination vector, specifically
    - Also, should maybe get a GAN to output the next 5 frames as extra input for the network
- I collected about 30 batches * 500 frames = 15,000 frames of data (15,000 frames * 0.05 seconds / frame = 750 seconds = 12.5 minutes of data)
- I'll use 11 different neural networks to train on the data, and I'll use some of the code from the rover at the MPCR lab
- I've run into some errors, I will continue modify the program to correct these tomorrow and hopefully finish training
- Turns out there is an issue with RAM when I try to load in too many files at once, so I can only load in a few at a time to view the files
- Also, there are some blank frames scattered throughout because of the way I stored the batches, but it shouldn't affect the performance of the neural network

July 2 (9:00-):

- Will try the “scale up” “normalize” outputs around zero again, see if it trains better with unit variance around zero
    - Didn’t quite work, validation loss stayed around the same value, and training loss decreased (overfitting?)
- I think the dataset is not good enough, so I was reading a survey on SfM
    - Found out that SfM doesn’t work well for datasets where you’re going forward in a straight line (autonomous vehicles) because the useful features of the images are only present in the periphery for a short duration of time
    - Should think about taking data differently
- Going to take more data that is less in a straight line and instead focuses on big buildings, then run sfm on that
- Tried running another sfm (which is supposedly better) on the same photos and it didn’t really work either (jumped around)
- Took video from 360 camera and split it up and ran it through both SFM, but still didn’t really work
    - Going to try to crop each image to only the center and the sides and see if that works better
    - Didn’t work that much better, still spazzed out
        - Went in and checked the specific order of the images, and even images that were taken after others (in front of) were registered behind in the 3D reconstruction
- Went to film video of the FAU stadium, seeing if large open spaces would help (Dr. Hahn already did this)
- Also going to try training on the Cambridge Posenet datasets
    - See if it’s just the bad data or something intrinsic about the pretrained network or my code
- Downloaded the dataset, made some observations:
    - They covered a lot of spatial area (zig zagged through instead of just walking in a linear path)
    - They focused on one object (a building, at least for King’s College)
    - They took *a lot* of data (about 4,000 image frames)
    - They took video in different conditions (sunlight, shadow, etc.)
- Made a little script to compile all of the data into one folder (on gist)
- And also should use https://stackoverflow.com/questions/33181757/batch-convert-png-to-jpg-and-output-images-to-new-folder-imagemagick mogrify on linux to batch convert png to jpg
    - Commands:
        - mkdir OutputDir
        - mogrify -path OutputDir -format jpg -flatten -quality 100 *.png

July 3rd (9:00-3:00):

- Running the visualsfm on my home computer, will train neural network on the reconstruction nvm provided
- Extracted into a dataset (data3.pkl) and will start training
- In the meantime, will take more data outside, sort of how the posenet team did it (covering more space, etc.)
    - Took 6 videos each 30-60 seconds long

July 5th (9:00-):

- When I tried to load in the pickle file of the sequence from the posenet paper from cambridge it still hasn't opened
    - Going to try to see if I can encode the dataset into a friendlier format
- Maybe what I can do is something different
    - Instead of actually loading all of the image files at once into RAM in the notebook, I can just load it maybe in batches or one at a time
- There is a dataset_train.txt and dataset_test.txt that has the image paths and their corresponding labels
- I collected about 30 batches * 500 frames = 15,000 frames of data (15,000 frames * 0.05 seconds / frame = 750 seconds = 12.5 minutes of data)
- Turns out there is an issue with RAM when I try to load in too many files at once, so I can only load in a few at a time to view the files
- Also, there are some blank frames scattered throughout because of the way I stored the batches, but it shouldn't affect the performance of the neural network
- I'll use 11 different neural networks to train on the data, and I'll use some of the code from the rover at the MPCR lab
- I've run into some errors, I will continue modify the program to correct these tomorrow and hopefully finish training

Extra daily notes can be found at
https://github.com/mpcrlab/DeepControl