DSC190 WI24
Team 1: Donkeycar
(Computer Vision and GPS autopilots)

Bryan, Rohan, Nick

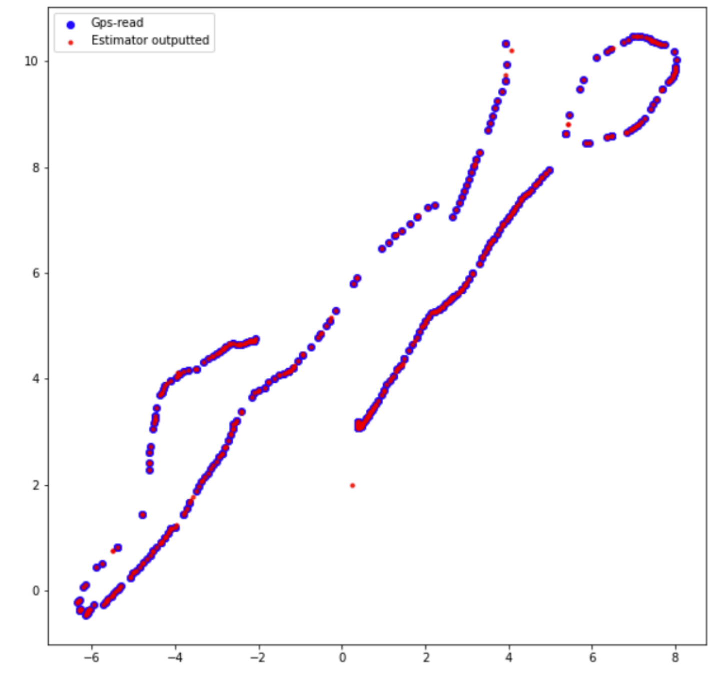
GPS Path-Follow Autopilot (utilizing IMU data to predict pose at a higher frequency)

- What we have promised
    - To ensure that the class has a working fallback autopilot through donkeycar for the pre-mapped racing category
    - To learn and improve Donkeycar's GPS path-follow autopilot in any way possible
- Must have
    - Functioning car to test on, with permissions to make edits to files
        - Car must have a GPS
    - Testing area with stable GPS signal
- Nice to have
    - IMU with accelerometer and gyroscope
    - Additional testing platforms
- What we have done
    - Tuned an initial PID autopilot for the 1/10th scale car
    - Modified the imu.py part to access the OAD-D Pro camera's onboard IMU
    - Created and tested a 'simple position estimator' part that predicts the car's position in-between GPS readings
    - Tested running donkeycar at higher frequencies than the initial recommendation (recc'd 20Hz, successfully ran at 200Hz)
- What worked
    - Modifying and testing the oak-d pro w's IMU went surprisingly smooth with minimal issues
- What did not work as expected
    - GPS inconsistency
        - Issue: In the first half of the quarter, we struggled a lot while figuring out how to tune the default DonkeyCar GPS path-follow autopilot.
        - Why?: After 2 weeks we figured out that the problem was that we were listing the wrong USB port in the config file; this is because the GPS creates two USB entries when plugged in (USB0 and USB1) - both of which work when used in the config file. However, USB0 (the one we started with) returns data inconsistently with many missing values.

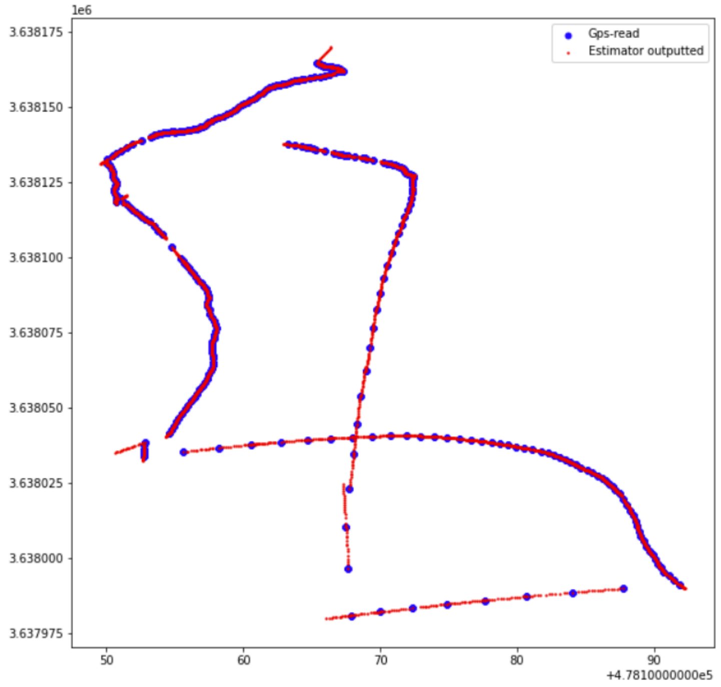
        - How we solved it: After Raymond showed us a method to directly test the GPS, we noticed that it used USB1 instead; switching to USB1 in the config file yielded consistent data with no missing values (in other words, working as intended).
    - Seemingly slow position estimator
        - Issue: After our initial implementation of the simple position estimator into the donkey loop, we noticed that it was only outputting 1 ‘predicted’ position in between each GPS point. We had expected many more.
        - Why?: We made the assumption that the pos/x and pos/y that we were seeing in the terminal while driving were instances where the GPS was read. In actuality, these were separate values that are related to the CTE part that calculates cross-track-error - a process that is not threaded, and locked to the donkey loop. On the other hand, the GPS runs as a threaded process - apart from the main loop - and was not setup to produce a log message. This lead to us believing that the position estimator was only running once per GPS read (much slower than desired)
        - How we solved it: After some consideration on whether or not the position estimator should be a threaded part (so that it could be run more often without affecting the main loop), we decided that the only way to get more predictions per GPS-read was to increase the main loop frequency. After increasing it from 20Hz to 50Hz, we noticed a greater number of predicted points - meaning that the solution was just to increase the main loop frequency. We pushed as far as 200Hz.

- Data collected
    - Position estimator performance: Predictions vs GPS-positions
        - (Note that the important part is the space filled in-between GPS readings; the weirdness of the graphs below is due to GPS interference)

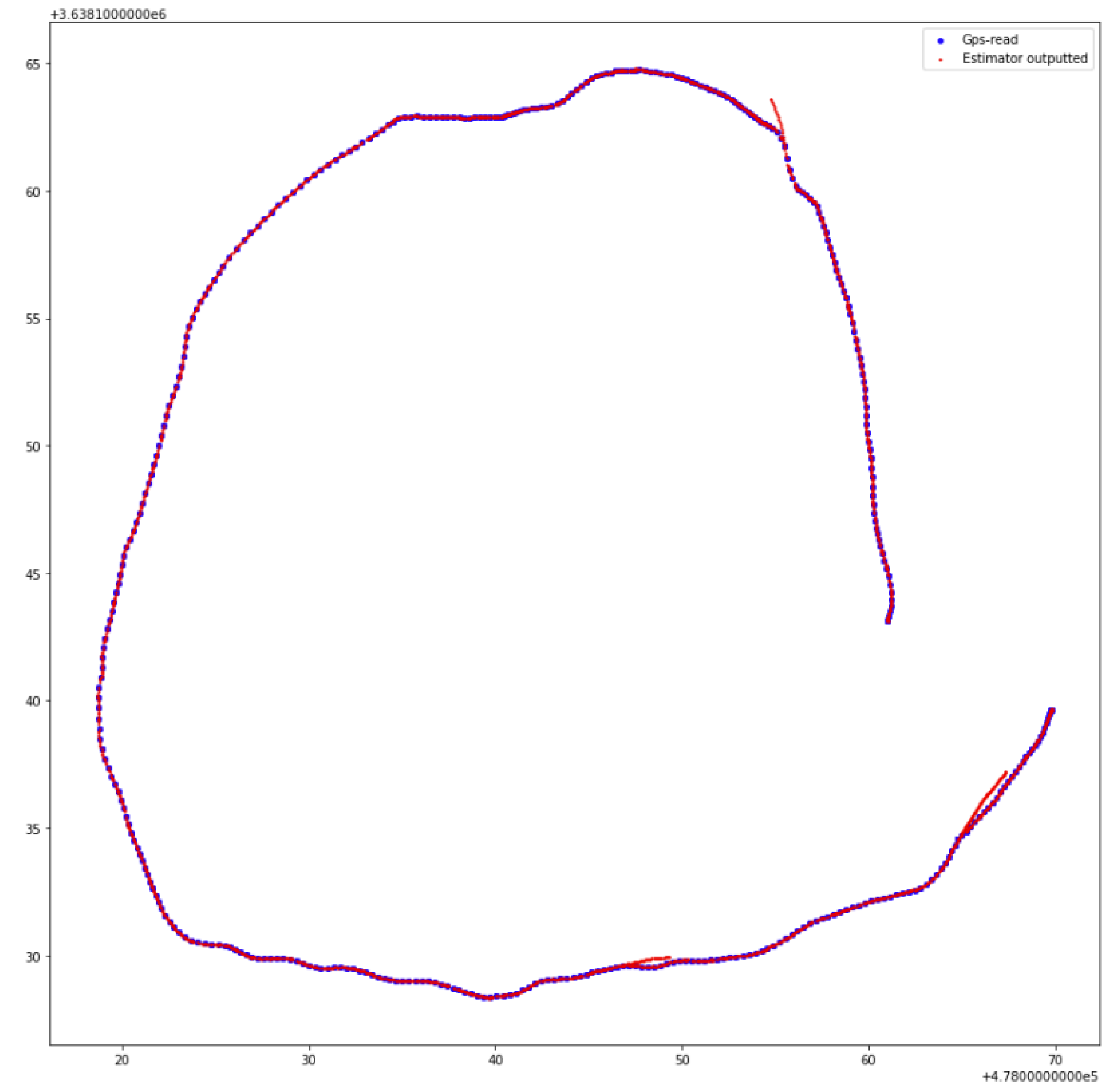
Initial, Low-Frequency (20Hz):



Later, High-Frequency (200Hz):



Later, Mid-Frequency (50Hz) with larger track:



- Data science techniques and Data analysis
    - Data visualization (plotting and animation of data using python)
    - Lots of python coding (modifying and adding on to the existing DonkeyCar files)
    - No other special techniques used for the GPS autopilot; mostly just visual confirmation of part performance. More data-science specific stuff found in the other autopilot section
- Next steps for the next team building on your work / If we had more time
    - Test the performance of the PID autopilot when using the simple position estimates in place of the standard gps-position. (The performance difference is likely more easily observed with something faster than the 1/10th)

    - Further fuse the GPS/IMU data using an Extended Kalman Filter to get even more accurate position estimates
    - Create an alternative to the PID autopilot (such as Pure Pursuit) which requires less tuning (and can make use of the car’s orientation)
    - Clean up and document code so that it may be added to DonkeyCar’s github as a contribution
- Lessons learned, if you could go back in time what would you do differently to increase your chances of success, what work could someone do after yours, and references
    - Get familiar with the Donkeycar framework sooner; look at how the vehicle is created, where the drive-loop occurs, and how parts interact with each other. Understand what it takes to add a new custom part.
    - Make sure the car is fully functioning before taking it out to test; we lost a lot of time to small things that were easy fixes - such as something not being plugged in (USB cable missing or soldering undone during use) or needing to be restarted (steering and joystick-controller connection). If it’s not code-related and you don’t know why the car isn’t working, ask someone who’s familiar with the mechanical/electrical components sooner rather than later.
- Important Resources used
    - https://docs.donkeycar.com/
    - Github issues on the main donkeycar repo (such as https://github.com/autorope/donkeycar/issues/1040, and https://github.com/autorope/donkeycar/issues/1060)
    - Communications with experts outside of class (i.e. Ed)
    - MATLAB youtube playlist on sensor fusion https://www.youtube.com/watch?v=6qV3YjFppuc&list=PLn8PRpmsu08ryYoBpEKzoMOveSTyS-h4a&ab_channel=MATLAB
- In summary:
    - We spent the first half of the quarter familiarizing ourselves with DonkeyCar, and learning how to operate the 1/10th car with the stock GPS autopilot (plus some PID tuning). For the second half, we investigated how to use the IMU on the OAK-D Pro (which required modifications to Donkey’s imu file, which assumed a connection over I2C), and then wrote / tested a simple position estimator that uses the car’s velocity, acceleration, and orientation to predict its position in-between GPS reads. We also began educating ourselves on Kalman filters and sensor fusion, as well as alternatives to the PID autopilot, in preparation for next quarter.

Donkeycar Vision

- What we have promised
    - Ensure we can create a pipeline that allows for easy facilitation of driving the donkeycar, then subsequently using data from the car's output to create a model that mimics human driving behavior
- Must have
    - Functioning car to test on, with permissions to make edits to files
        - Car must have a OAK-D camera
- Nice to have
    - OAK-D Pro camera to get more "bang for our buck" in terms of more data (3 images - left, center, right - at 20Hz compared to 1)
- What we have done
    - Wrote code to support the 3 image output from the OAK-D Pro camera (oak_d_camera.py)
    - Changed config and complete.py files to support the three image setup
    - Achieved multiple autonomous laps on the EBII courtyard track after roughly 12 minutes of training
    - Created a system, using the 3 image output, that now allows for faster data collection
- What worked
    - Though documentation was not always abundant, most software issues were relatively easy to debug and was generally not an issue
- What did not work as expected
    - Lighting
        - Issue: The main issue we had with deploying the model was to recreate the lighting conditions in which the model was trained on. If, and only if, the model was trained and deployed in quick succession could it make the right judgements in highly active steering situations
        - Why?: It's easy for a human to look at the images we're giving to the model and easily identify where a turn needs to take place or when to step, but when you have a model that is not pre-trained to make such correlations beforehand, it gets easily confused on what it should exactly be focusing on. For example, if one lap of data is given to the model - focusing specifically on the turns, did the car's steering adjust because of the curving white lines or something in the periphery? Only through scale can these properties be discerned.
        - How we solved it: Beyond just creating a faster pipeline from training to deployment, collecting 3 images every 20Hz instead of 1 gives the model additional data points to draw from, thus mitigating this issue.

- Data collected
    - A successful training run will collect around 5000 images, as well as their steering and throttle values

- Quick example of deployment (car at 0.5 speed): https://discord.com/channels/1195781930264821892/1195785105898213426/1217958105631948801

- Data Science techniques used
  - Computer Vision, while we had no involvement in creating the Convolutional Neural Network that we used to train, this led to other Data Science techniques, specifically in Computer Vision such as image augmentation (3 images at a time instead of 1, using Donkey UI to gather further data points - on the roadmap) and also exploring how other representations of this data, such as depth images, can be useful.
- Data analysis
  - On the roadmap, we'd like to explore how incorporating different aspects of the data available to us may contribute to more successful deployments (i.e. adding depth data, types of image augmentations) and being able to quantify that
- Next steps for the next team building on your work
  - Explore how transfer learning could create a more "generalized" model that we could pair with GPS
  - Fix lighting inconsistency issues by training our model at different times of the day and using transfer learning as a means of gathering more data to mitigate this issue
  - Explore the shortcomings of our model. So far we've only done training and deployment runs around the EBII building, but it would help to explore this in different settings
- Lessons learned, if you could go back in time what would you do differently to increase your chances of success, what work could someone do after yours, and references
  - Become familiar with the Donkeycar framework sooner, as in accumulating experience just driving the car yourself and getting comfortable with fixing issues that come along the way (controller not connecting, steering not working, WiFi not connecting, etc.)
  - Understand the problem fully and think of it with a First Principle approach and work backwards from there
  - Recognize that errors and debugging are absolutely part of the process in trying to engineer anything. Solving the many little problems in a larger problem generally takes the majority of the time.
  - https://docs.donkeycar.com/guide/train_autopilot/
- In Summary
  - In implementing a system that utilizes three images at 20Hz from the OAK-D Pro camera, we significantly improved the efficiency of data collection, which, coupled with adjustments in training and deployment, helped overcome challenges related to lighting conditions and model performance. This approach not only facilitated more successful laps but also allowed for further advancements in model generalization and adaptability to varying environments.