I will firstly outline the current plan for the development of the car follower AI, and then discuss some more ambitious, longer-term plans as learned in my weekly research.

The plan remains that the first step in the AI development will be to train a classifier to recognize the orientation of the leader car relative to the follower car (assuming the camera is pointed straight forward). The three most important orientations to classify are “straight on”, “left turn”, and “right turn”. While there is certainly room to classify more orientations (e.g. “reversed straight on” if the lead car was on a head-on collision course with the follower car) or higher granularity of orientations (e.g. “sharp right turn”) our current plan for data collection limits the amount of training data we can accrue for these less common orientations.

In addition to classification of the lead car’s orientation, it’s important that the follower car be able to have some idea of its following distance in order for it to act appropriately. The follower may want to increase the throttle if it is farther away than it would like, or in the opposite case it should ease off the gas if it finds itself too close to the lead vehicle. Although my planned backbone architecture of resnet is not particularly tuned to recognize precisely where in an image an object occurs, it should not be too hard to extract this information from the network—this intuitively makes sense because a model must having some understanding of where an object is in an image in order to try to classify it, and likewise there is practical examples that this should be the case in fastai Lesson 9. As outlined in the research notes, we can use bounding boxes as an additional training input and output of the model, and change our throttle according to the size of the bounding box (or more precisely let the neural network come to figure out this behavior on its own).

This type of model may do alright on its own, but my suspicion is that at best its behavior will be jerky, and will as such not quite resemble how one drives a car. The problem is that the system is more dynamic than a simple image can convey: we also have state in the form of our current throttle and direction data. In other words, our previous outputs should affect our future outputs, leading to the idea that our model should have some amount of recurrent layers. I haven’t fully worked out how this architecture would work, but my plan would be trying to add a small head of recurrent layers onto the visual (convolutional) backbone of the network. There is precedent for this kind of network, so I suspect it is worth pursuing; however, it is difficult to say when our training data is not yet collected and the performance of our simpler model has not yet been evaluated. Perhaps in consideration of the fact that our hardware is relatively weak, and that not much will need to be remembered to drive the car relatively smoothly, we should add only a small amount of recurrent layers. On the other hand, adding a large number of recurrent layers could allow for more advanced behavior like reestablishing line of sight (with the follower car in theory remembering the path of its leader before loss of line of sight). This may very well be unfeasible, though, with the model running on just a Raspberry Pi. Regardless, after more research and further into our project’s development I think pursuing recurrent layers will be greatly helpful to the model’s accuracy.

As a final minor point, it should be noted that because we’ve confirmed the existing python code on the Raspberry Pis is only Python 2 compatible we’re left with two options: port the existing code and picar library to Python 3, or else try to use the Pytorch version compatible with Python 2.7 on Linux. I think the former is more promising, firstly because I do not think porting the code will actually be so problematic, and more importantly because performance is of the essence and I doubt the Python 2.7 version of Pytorch has received much attention.