Machine Learning Engineer Nanodegree

Capstone Proposal

Vikram Sriram
10/1/2019

Proposal

Domain Background and Problem Statement

Fully autonomous cars are slowly becoming a reality and are very important in this day and age. Detecting obstacles, pedestrians, and following the rules of the road is a challenge. With automotive giants competing to develop these cars, we have made remarkable progress.

Reinforcement Learning is an interesting approach to solve problems, but often requires a simulation to be trained and many iterations. There needs to be a source of noise or variability to find the optimum action that maximizes the reward function. Building a simulation can itself be an enormous challenge, because of the many variables that you have to consider. Transferring model from simulation to real life isn't always easy. Instead of using a simulation, I want to find a way to learn from user in real-time.

Earlier I developed a simple line following car using PID. Although it worked well, I want to revisit this challenge and use the power of reinforcement learning alongside PID to see if it yields better results. Based on the states and curated reward function, the behavior of the car can be interesting to observe. The car could perhaps learn more efficient ways to finish the track without the need for explicitly programming complex behavior for different car speed and edge cases.

My motivation:

I initially chose this project because I thought it is a great way to revisit and perhaps learn new technologies in the process of making this car. This project involves the following: 3D design, Embedded programming, Electronic and Digital Circuits, Python programming, and Machine Learning.

Challenges:

Reinforcement Learning in real-time with limited hardware.
Full stable setup for the car and Jetson Nano environment.
The PID version needs to be consistent and reliable for training.
Developing architecture for DQN.
Image processing to detect lane.
Designing hardware to control auto steer and throttle efficiently.

Datasets and Inputs

Since this is a Reinforcement Learning car, the car would ideally learn in real-time. In case the hardware is too slow, I plan to fill the replay buffer with ”good samples” from human and augment that image to train. This would be a better starting point for the model, rather than learning from scratch.

I’m debating on whether to feed the input image data straight into a CNN policy or extract key image features using OpenCV to create a lighter state space, this is also depending on the hardware capabilities.

Solution Statement

The solution will be derived through trial and error. But, I will first attempt is to use a prefilled “good sample” replay buffer and fine tune the hyper parameters. The inputs will also depend on how I choose to interpret the state input. I will experiment with the input state vector and the reward function which can also be influenced by the state vector.

Benchmark Model and Evaluation Metrics:

I will first need to run the car using image processing and PID. This will be my comparison model to benchmark against. Additionally, I also plan to use this to train the agent to possibly simplify my spate space for faster training.

Another benchmark model will be a CNN model from Nvidia’s research paper with a little bit of image pre-processing via OpenCV.
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

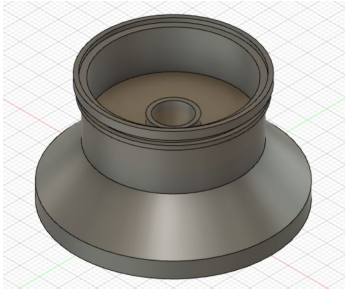
I drew inspiration for this method from this article:
https://towardsdatascience.com/deeppicar-part-5-lane-following-via-deep-learning-d93acdce6110
This article using a Raspberry Pi Board was incredibly helpful for getting started. I plan to implement something similar to this CNN model with similar image preprocessing.

I will use lap time and training time as a metric to compare the performance of these different approaches.

Project Design:

The Car:

The chassis of the car was removed from a model car assembly kit.
http://www.racing-cars.com/pp/Car_Showroom/SupaStox_ATOM.html#
Upon assembly, I discovered that the car ground clearance was very low due to the weight of the boards and stress on suspension, so I had to design larger front and rear wheels for more ground clearance.



The car is a rear wheel drive with a DC Motor and Servo for steering control. I needed a motor control PCB, fortunately I have my Motor Control PCB from my previous car.

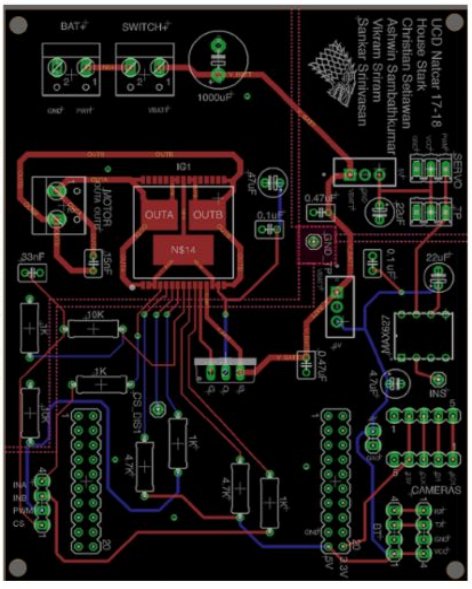


Figure 2. PCB Design Schematic

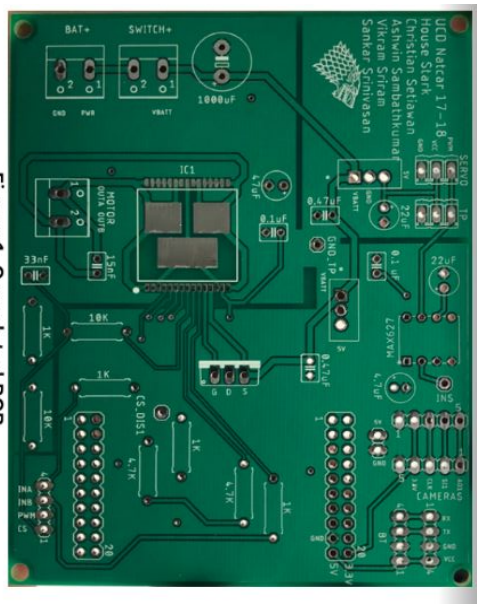
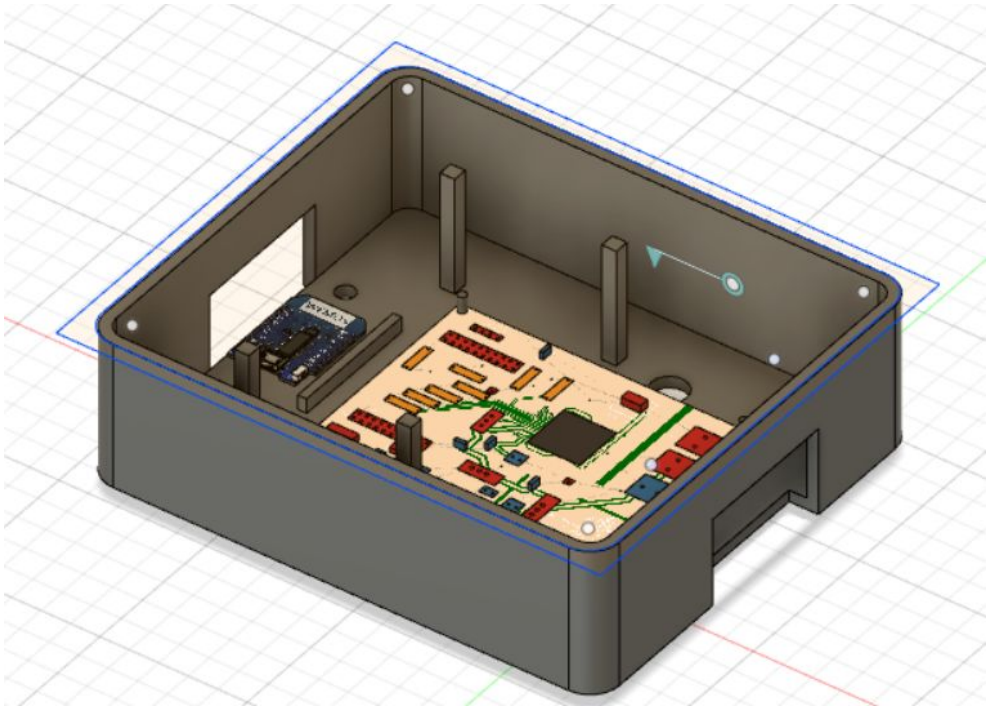
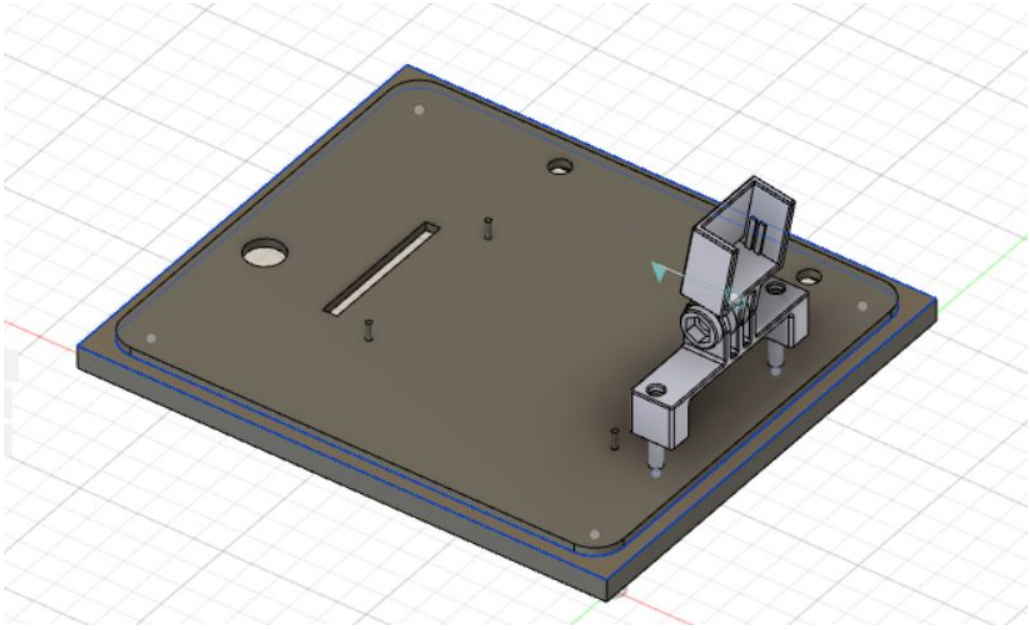


Figure 1. Completed PCB

Next step was to create an enclosure that houses and protects the electronics. For this, I designed a simple box with holes for wires and airflow. I decided to place the Jetson Nano AI/vision controller board on top of the card, secured by stubs.





Raspi-Cam Camera Mount Source:
https://github.com/dudasdavid/Jetson-nano-case

The generated PWM signals need to come from a system with hardware realtime PWM signal implementation available. For this, a regular Arduino Nano was used. I wanted a sort of black box where I can feed the raw action output which will be steering (0-180 Degrees) and throttle (0 to 100%). This was done to keep things simple for training.

For training, I needed a way to wireless send inputs to the Jetson Nano. I developed a small remote controller on a breadboard with 2 Joystick analog inputs for steering and throttle. The controller was powered by Attiny85. I chose this MCU because it was very low power consumption and it has 2 10-bit ADCs. I chose RF 433Mhz for wireless communication between the Arduino Nano and Attiny85 controller. With a finely tuned antennae (loaded coil design), this module had good range.

This controller can also serve as a way to bring the car back on track to resume training and as an emergency control device if the car veers off road into obstacles.

Jetson Nano:

The Jetson Nano is running light-weight Ubuntu OS with all the essential ML tools installed. The input to the vision controller is a Pi Camera connected via CSI cable to the main board.

Power:

The power for the DC Motor, Servo, and Arduino Nano comes from a 7.2 NiCd battery that is connected to a DC linear voltage regulator that is on the Motor Control PCB.

The Jetson Nano is powered by another 7.2 NiCd battery connected to a buck converter that is capable of supplying 3A. This is critical because the Jetson will consume 2A just without any peripherals, so 3A is just the right amount for some GPIO outputs, USB peripherals, and the Pi Camera attached directly onto the board.