Jalopy

Python plays Euro Truck Simulator 2[1]

Project overview

I am building a self-driving algorithm for the video game Euro Truck Simulator 2. I'm naming the algorithm *Jalopy*.

Competitive analysis

Many have tried to build similar self-driving algorithms for this game; many have succeeded.

The popular open-source package for the game *europilot* by *marsauto* involves using a convolutional neural network (cNN) to direct the truck. From what I have seen from the GitHub repository, *europilot* takes several screenshots of the game's state each second, and then runs in either one of two ways. If the user has activated 'training mode,' they will drive the truck themselves as the game quietly records their key-pressed values as a new row entry in a .csv file alongside the screenshot. This creates a 'virtual joystick' for the program to begin learning how the user drives.

On the other hand, if the user is in 'testing mode,' the virtual joystick will output the relevant commands to the truck, such as steering, shifting gears as necessary, accelerating, and braking. It uses the training data accumulated from testing mode to generate its inferences on how to best direct the vehicle.

I envision *Jalopy* will be like *europilot* and many other packages with its usage of OpenCV to track game state. However, I am not certain yet if I will use a neural network in my implementation. The drawbacks of the cNN is that I will need to devote several hours (potentially even days) to train my algorithm, and the resulting pack of images would be several gigabytes large on my small SSD. I detail my approach below in the algorithimic plan section.

Structural plan

Jalopy will be organized into three main components:

1. Processing game screenshot
2. Detecting lanes and calculating steering angle
3. Sending key-pressed output to the game in real-time

[1] Euro Truck Simulator 2 (ETS2), published by SCS Software, is a popular 2012 driving simulator video game for Microsoft Windows, Mac OS, and Linux. Players drive and deliver cargo across Europe.

These objectives can be achieved through a few Python files within a *jalopy* folder, which would probably be named according to their relevant function: *screenCapture*.py, lanes.*py*, *steering.py*, *sendkey*.*py*, etc.

Other folders involved may include *assets/*, which will contain game images, including the rotating 3D gif of the truck on the start screen, as well as any typographic elements.

Because I can't include Euro Truck Simulator 2 in this project itself, I plan on making an *examples/* folder with gifs of *Jalopy* in-action for all to view.

I'll probably also need a *setup.py* file to install the relevant Python modules onto other people's computers, if they would like to install *Jalopy* onto their systems.

Algorithmic plan

Lane detection and steering is currently the largest hurdle conceptually for me. I could attempt implementing the neural network I often see used in these sorts of self-driving programs, but I think that there is a simpler solution.

If I can detect and draw the lanes, I can find crude slopes of those lines. These two slopes roughly represent the angle of the road, or instance:

- If both lanes have negative slopes, then the road is turning left.
- If the left line has a positive slope but the right line has a negative slope, the road is straight.
- If both lanes have positive slopes, then the road is turning right.

Once I can refine my slope-detection model, I can then send WASD commands to the truck to accelerate, steer, brake, etc.

In terms of the game settings, I plan on running the truck at a low speed[2] to maintain control of the truck and prevent bad things like roll-overs.

Timeline plan

By TP 2:

- The truck can successfully navigate expressways and rural roads.

[2] Say, < 40 kph

- The truck can detect cars in front of itself and brake accordingly.
- Basic user interface to control basic movement functionality i.e. stop, start, sliders for lane tolerance/ speed.

By TP 3 (hopefully):

- The truck can stop at stop signs and red lights.
- The truck now uses mouse-based steering controls for finer manipulation.
- The truck can recognize highways from city and rural roads.
    - The truck can overtake cars on the highway (questionable feasibility)

Version control plan

I'm using GitHub to host my work at *github.com/eh8/jalopy*

Module list

Primary

- `OpenCV`
    - For imaging and the bulk of preprocessing work
- `ImageGrab (from pillow)`
    - (otherwise known as Python Imaging Library)
    - To take screenshots of the game and place into *Jalopy*

Auxiliary (primarily installed through Anaconda)

- `numpy`
    - For array handling and math calculations
- `matplotlib`
    - Allows me to finetune my masking
- `pygame`
    - For eventual GUI
- `pprint`
    - Pretty printing the OpenCV arrays for debugging purposes

TP2 Update

I spent a considerable amount of my TP2 time trying to finetune my slope steering heuristic with limited success.

While not fully implemented, I would like to amend the speed control section of my user interface to just be a multiplier. As it turns out, trying to gauge a vehicles speed using only a camera feed is hard.

TP3 Update

Everything in this program is meant to be as fast as possible—fast image processing, fast lane detection, fast logic to steer, etc. For

this reason, I realized some of the initial considerations I had in my storyboard would be prohibitively slow or just infeasible given my computer specs.

I simplified the user interface and added clear feedback as to what the truck is doing. I made it easier for the user to see what was going on.

I ended up using `PyQt5` for the user interface component of the program and developed a unique lane-finding heuristic that works well on gentle roads and highways.

The image filtering system has now been updated with separate Sobel detection, RGB, and HLS filters to find lanes. A polynomial function tracks lanes with decent accuracy across weather and times.

The steering functionality is controlled by a derivative check in horizontal truck position, as well as a brute analysis of the truck's estimated deviation from the lane.