

# Use Deep Learning to Clone Driving Behavior

审阅


代码审阅 1


HISTORY

## Meets Specifications

Dear Student,

I am really impressed with the amount of effort you've put into the project. You deserve applaud for your hardwork!

 Finally, Congratulations on completing this project. You are one step closer to finishing your Nanodegree.

Wishing you good luck for all future projects 

## Some general suggestions

### Use of assertions and Logging:

- Consider using [Python assertions](#) for sanity testing - assertions are great for catching bugs. This is especially true of a dynamically type-checked language like Python where a wrong variable type or shape can cause errors at runtime
- Logging is important for long-running applications. Logging done right produces a report that can be analyzed to debug errors and find crucial information. There could be different levels of logging or logging tags that can be used to filter messages most relevant to someone. Messages can be written to

the terminal using `print()` or saved to file, for example using the [Logger module](#). Sometimes it's worthwhile to catch and log exceptions during a long-running operation so that the operation itself is not aborted.

## Debugging:

- Check out this guide on [debugging in python](#)

## Reproducibility:

- Reproducibility is perhaps the biggest issue in machine learning right now. With so many moving parts present in the code (data, hyperparameters, etc) it is imperative that the instructions and code make it easy for anyone to get exactly the same results (just imagine debugging an ML pipeline where the data changes every time and so you cannot get the same result twice).
- Also consider using random seeds to make your data more reproducible.

## Optimization and Profiling:

- Monitoring progress and debugging with [Tensorboard](#): This tool can log detailed information about the model, data, hyperparameters, and more. Tensorboard can be used with Pytorch as well.

## Required Files

The submission includes a `model.py` file, `drive.py`, `model.h5` a writeup report and `video.mp4`.

All files are included in the submission zip

- ✓ `model.py`
- ✓ `drive.py`
- ✓ `model.h5`
- ✓ `writeup`
- ✓ `video`

Suggestion:

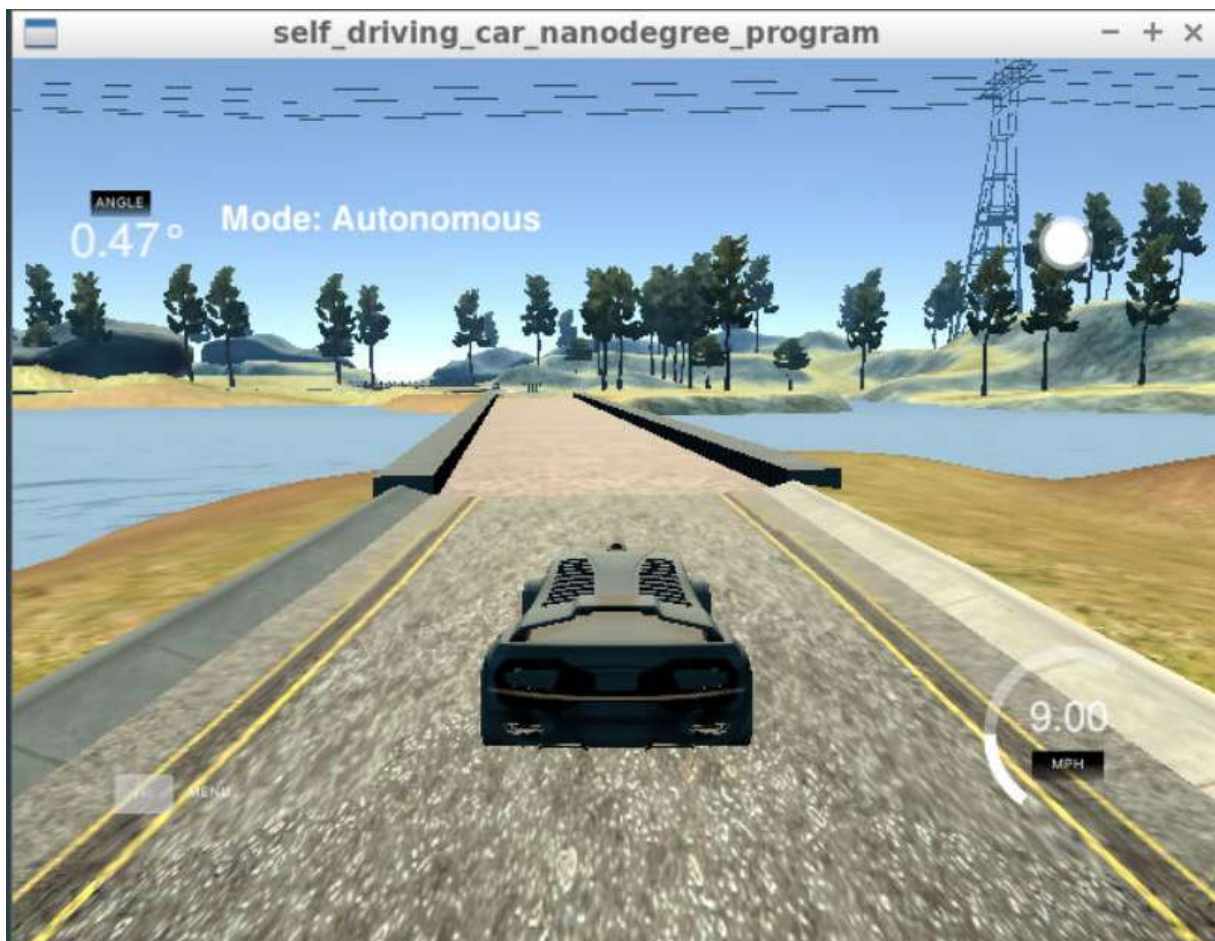
You can export your conda environment into `environment.yaml` file so that you can recreate your conda environment later while practicing on your own system. Use the following command -

```
conda env export -f environment.yaml
```

## Quality of Code

The model provided can be used to successfully operate the simulation.

I tested your `model.h5` file in the simulator and model managed to operate the car successfully ! 👍



The code in `model.py` uses a Python generator, if needed, to generate data for training rather than storing the training data in memory. The `model.py` code is clearly organized and comments are included where needed.

```
yield sklearn.utils.shuffle(X_train, y_train)
```

Good job keeping your code clean and organized.

`model.py` uses a Python generator appropriately to generate data for training.

Note - If you wish to experiment with other libraries, I recommend you to check out PyTorch. It comes with a built-in data loading module called `DataLoader` that makes loading data a breeze. It also supports unique features like automatic batching.

You can check out more features of the module [here](#)

## Model Architecture and Training Strategy

The neural network uses convolution layers with appropriate filter sizes. Layers exist to introduce nonlinearity into the model. The data is normalized in the model.

- Lambda layer has been used to normalize the input data.

```
Lambda(lambda x: (x / 255.0) - 0.5)
```

- You've invoked and used the `Conv2D` module in Keras to build convolutional layers with appropriate filter sizes.
- Non-linearity has been added by means of `ReLU` activation function.
- In the final stages, the output from convolution layers is flattened and fed to the `Dense` layers to perform regression and predict the steering angle.

```
model.add(Flatten())
```

Train/validation/test splits have been used, and the model uses dropout layers or other methods to reduce overfitting.

- Using a validation set is extremely crucial as it allows us to monitor the model's generalisation abilities. It aids us in choosing the right set of hyperparameters and monitor overfitting.

```
train_samples, validation_samples = train_test_split(samples, test_size=0.2)
```

- Dropout layers help in preventing overfitting by randomly shutting down nodes during the training process. You can check out the original paper of this regularisation mechanism [here](#)

```
model.add(Dropout(0.2))
```

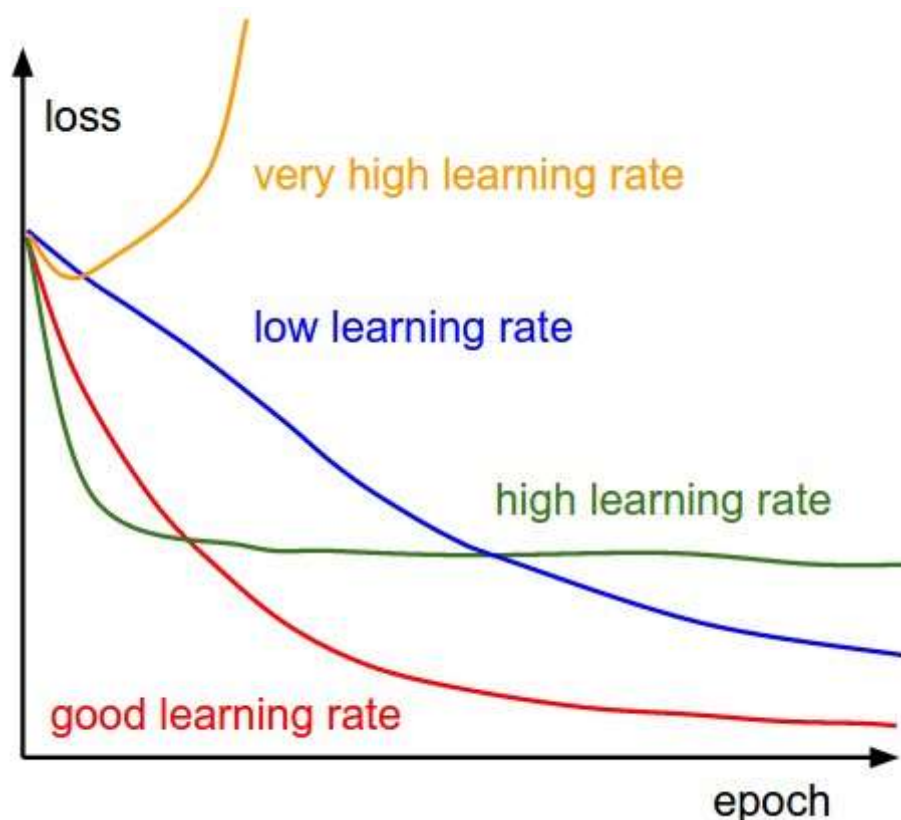
There are some aspects of Dropout that are usually not talked about anywhere. This [blog](#) does a great job of summarising those details.

Learning rate parameters are chosen with explanation, or an Adam optimizer is used.

### 3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (`model.py` line 107).

Here's a curve of loss vs epochs for different learning rates. You can see that for high learning rates the loss might decrease initially but stagnates or rapidly goes up again as the number of epochs increases.



Coming to the topic of Hyperparameter tuning, there is no universal answer to what works well. Therefore, it's best to experiment with a range of different values to check which hyperparameters result in the best model.

Check out this wonderful guide on [HyperParameter Optimization for Deep Neural Networks](#)

Additional Reading:

Sebastian Ruder's article comparing different optimization algorithms - <https://ruder.io/optimizing-gradient-descent/>



Training data has been chosen to induce the desired behavior in the simulation (i.e. keeping the car on the track).

#### 4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road. I driving for nearly 7 lays and 5 laps in a counter-clockwise direction. The data size is around 1Gb.

I used the three cameras'images and angles as training data and the angles are corrected by 20 degrees. Then I filp the images and negtive the angles to augmentate the training dataset.

The submission uses appropriate training data, takes sensible algorithmic decisions in `model.py` to ensure that the car stays on track during the autonomous mode.

## Architecture and Training Documentation

The README thoroughly discusses the approach taken for deriving and designing a model architecture fit for solving the given problem.

#### 1. Solution Design Approach

My first step was to try to use LeNet model with a lamdba layer for normalization and a cropping layer to crop input image to appropriate size mentioned in the lecture notes. The car will easily drift out of the center of lane. It also works bad when the roadside is empty and so on.

Then I use the Nvidia's model architecture. This time the problem is the overfitting as the training loss decreases linearly while the validation loss fluctuates up and down. So I tried to add some dropout layers to reduce the overfitting problem.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting. Also I collect more data for training.

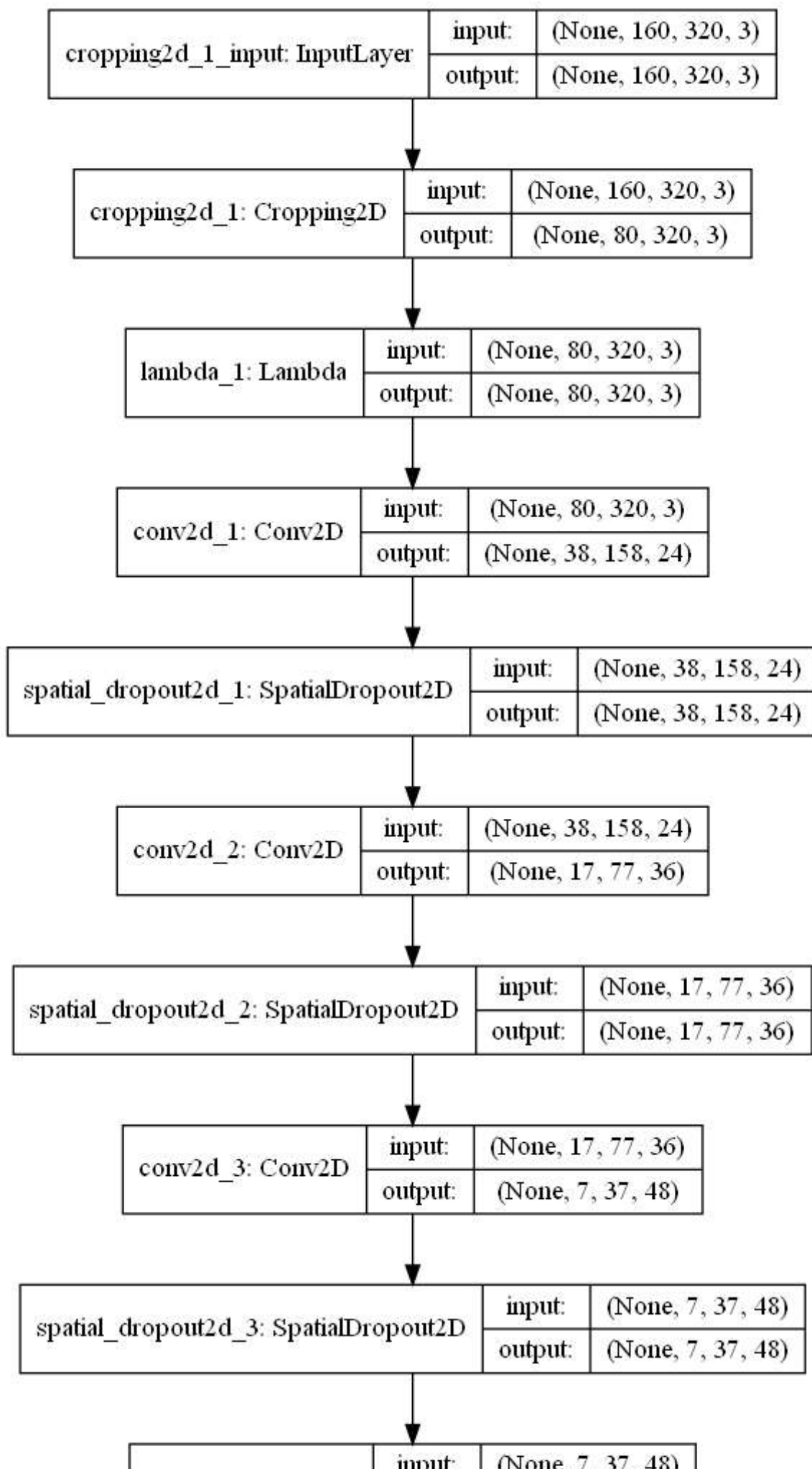
Finally the model is well trained and the the car can keep on the center of road all the time without leaving the road.

Good job with the write-up. You've provided a succinct description of your approach.

The steps involved in building and training a proper Neural Network are crucial.

Check out Andrej Karpathy's famous blog on the topic - ["A Recipe for Training Neural Networks"](#)

The README provides sufficient details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.



conv2d_4: Conv2D	input:	(None, 7, 37, 16)
	output:	(None, 5, 35, 64)



spatial_dropout2d_4: SpatialDropout2D	input:	(None, 5, 35, 64)
	output:	(None, 5, 35, 64)



conv2d_5: Conv2D	input:	(None, 5, 35, 64)
	output:	(None, 3, 33, 64)



flatten_1: Flatten	input:	(None, 3, 33, 64)
	output:	(None, 6336)



dense_1: Dense	input:	(None, 6336)
	output:	(None, 100)



dropout_1: Dropout	input:	(None, 100)
	output:	(None, 100)



dense_2: Dense	input:	(None, 100)
	output:	(None, 50)



dropout_2: Dropout	input:	(None, 50)
	output:	(None, 50)



dense_3: Dense	input:	(None, 50)
	output:	(None, 10)



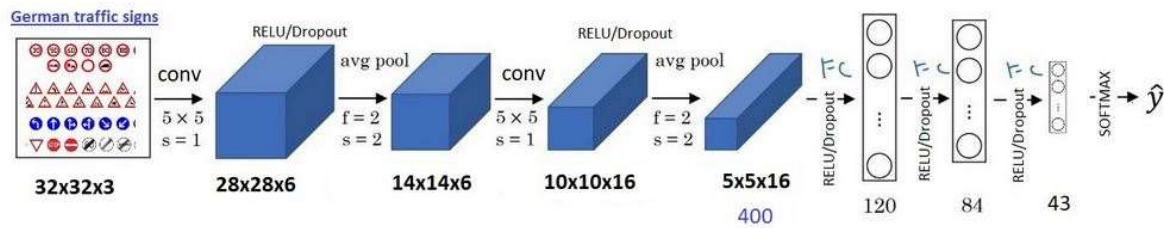
dense_4: Dense	input:	(None, 10)
	output:	(None, 1)



Nicely done! You've provided a lucid description of the model architecture. 🍑

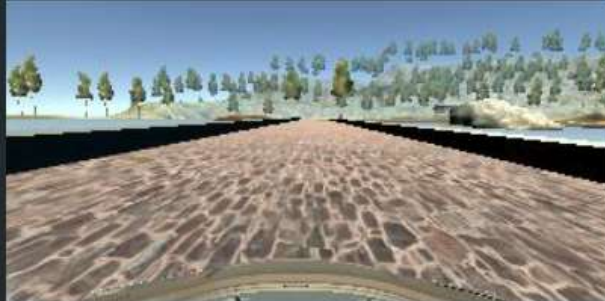
Good job using dropout between dense layers as it avoids overfitting and improves model performance on unseen data.

💡 Suggestion - You could've also included a diagram representing the model architecture. An example is shown below:

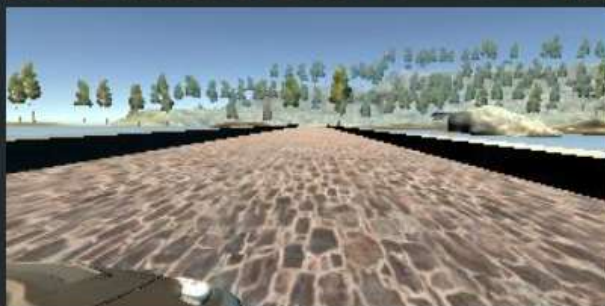
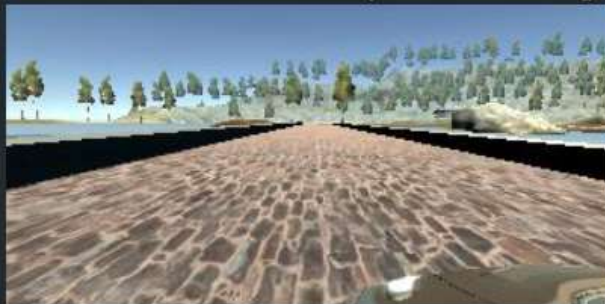


The README describes how the model was trained and what the characteristics of the dataset are. Information such as how the dataset was generated and examples of images from the dataset must be included.

To capture good driving behavior, I first recorded 7 laps on track one using center lane driving. Here is an example image of center lane driving:



I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to predict the steering angles.



Then I repeated this process on the counter-clockwise direction and collect 6 laps.

To augment the data set, I also flipped images and angles thinking that this would ... For example, here is an image that has then been flipped:





After the collection process, I had 25822 number of origin data (image, angle) and the same number of flipped data.

I finally randomly shuffled the data set and put 80% of the data into a validation set. All the data was shuffled randomly.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. I set the epoch number as 10. I used an adam optimizer so that manually training the learning rate wasn't necessary. The training and validation loss is shown in the following image. The validation is still little fluctuation but the trend looks like reasonable.

While it may seem like a daunting task, writing clear README's/WRITEUP's is an extremely rewarding skill. They provide several advantages such as:

- 1) Making it easy for potential recruiters to get an overview of your projects.
- 2) Before interviews, you can quickly review your README's/WRITEUP's to recall the core structure of your project so that you can talk about it in detail during the interview process.
- 3) When building similar projects in the future, clear descriptions of your past projects may help you take better algorithmic decisions.

## Simulation

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

The model steers the car successfully around the track. Nicely done! 👍

If you wish to learn more about Self Driving Cars, I would highly recommend you to go through the following blog series by Jonathan Hui - [https://medium.com/@jonathan\\_hui/self-driving-car-series-b8a356f7f2ac](https://medium.com/@jonathan_hui/self-driving-car-series-b8a356f7f2ac)

📄 下载项目

[返回 PATH](#)

**给这次审阅打分**

开始

---