

AUTONOMOUS CAR STARTING CODE PLANNING

Rudra Dey

Student# 1010124866

rudra.dey@mail.utoronto.ca

Pravin Kalaivannan

Student# 1010141295

pravin.kalaivannan@mail.utoronto.ca

Aadavan Vasudevan

Student# 1010101514

aadavan.vasudevan@mail.utoronto.ca

Abishan Baheerathan

Student# 1010218756

abishan.baheerathan@mail.utoronto.ca

ABSTRACT

This document presents our team's procedure to create the baseline model code for an autonomous self-driving car. ...

1 SETTING UP THE ENIRONMENT

1.1 INSTALLING CARLA

1.2 INSTALLING ANACONDA

2 DATA COLLECTION

2.1 CREATING THE DATA COLLECTION SCRIPT

In CARLA, there are already cars which have an autonomous functionality. To create the dataset, we collect data from the RGB Camera Sensor attached to the car at 10 frames per second (FPS) as the car travels around a designated map. Additionally, we collect data for the steering values and turn signals of the car allowing for use as ground truth labels for training the model. This data collection was done on diffrent maps with different weather conditions to ensure a diverse dataset.

Procedure:

1. Spawn in a vehicle
2. Attach an RGB Camera Sensor (front-facing) to the dash of the car.
3. Collect image data (10 fps) along with steering values and turn signals at that frame.

2.2 COLLECTING DATA IN STABLE CONDITIONS

The data was collected on a map without traffic on the road with the car travelling a constant speed. The camera will always be mounted in the same position and angle at the front of the car, as seen in a dash-cam. The FPS will be set to a constant 10 FPS. We will use the CARLA's autopilot to gain the steering values necessary to be used as ground truth labels. To get a diverse set of data, we will collect data in different maps and weather conditions.

2.3 CLEANING THE DATA AND SAVING DATASET

To ensure the data is clean, we will remove any images that are all black (indicated by the small file size)

3 CREATING BASELINE MODEL

This baseline model will predict steering angles from grayscale images for an autonomous vehicle simulator (which already has a autopilot built in).

The following libraries will be used:

1. numpy: For numerical operations on arrays
2. matplotlib: For visualization of data
3. scikit-learn: For machine learning components
 - (a) Ridge: A regularized linear regression model
 - (b) MSE: Mean Square Error for evaluation metrics
 - (c) train_test_split: For data splitting

The data will be loaded using two numpy arrays: images.npy and angles.npy. The images.npy will contain N grayscale images of resolutions 160x120, and the angles will have the corresponding images steering angles (of the car) in a range from [-1,1]. This makes it so the image data is represented as 3D arrays of samples x height x width as the dimension. The steering angles are continuous (regression problem).

The images will be flattened from their 2D representation of 160x120 matrix to a 1D representation of 19200 vectors. Where each image becomes a single row in a feature matrix. Additionally, we will split do a 80-20 train-test split.

The baseline model will be a ridge regression model. The ridge regression model is basically, the same as a linear regression model except, it regularizes large weights. It adds an L_2 penalty to discourage large weights, this helps with overfitting. The same as linear regression as we try to fit: $\hat{y} = Xw + b$, but now we try to minimize the loss function: $LOSS = MSE + \alpha ||\omega||_2^2$. The model will learn the weights corresponding to each pixel's contribution to steering.

We will compute two metrics, the mean square error (MSE) and the R^2 score (proportion of variance).

This Ridge Regression Algorithm is a linear regression technique which introduces a L_2 penalty which discourages the model from using very large weights, and thus improving generalization to unseen data. This model was chosen due to its simplicity and fast training time but also allowed quick adjustment to an α value which allowed us to adjust the regularization effect on the performance. By not using a plain linear regression model, we were able to prevent major overfitting, and generalize better.

3.1 BUILDING THE MODEL

3.2 EVALUATING THE MODEL

4 TRAINING NEURAL NETOWRK

*Quantitative Results: Training Loss (MSE) and Validation Loss

4.1 MODEL TRAINING SCRIPT

4.2 LOGGING

4.3 OUTPUTTING RESULTS

5 EVALUATING WITH INFERENCE

*Qualitative Results: Using model for actual driving in CARLA

5.1 RUNNING MODEL IN CARLA USING LIVE CAMERA FEED