AuE-8930 Deep Learning Final Project Proposal

Team Members (Group 4)

- Tanmay Samak
- Chinmay Samak
- Vasanth Seethapathi

Project Title

Data-Driven Discovery of Vehicle Motion Model using Bayesian Deep Learning

Problem Statement

Motion models are crucial for simulation and state estimation purposes and can be either developed using traditional or learning-based methods. On one hand, formal methods such as first-principles modeling are interpretable and explainable but require extensive system identification for parameter tuning and may result in computationally complex models. On the other hand, simple data-driven methods such as end-to-end learning can capture the dynamics to certain extent with low-latency inference, but their interpretability and explainability may be questionable. This work will investigate whether Bayesian deep learning can help bridge the gap between formal and data-driven approaches to develop computationally light-weight surrogate models that are generalizable, interpretable, and explainable. Particularly, we will be using the proposed approach to “learn” a reduced-order (i.e., surrogate) motion model of an Ackerman-steered vehicle given its low-level control inputs such as throttle, brake and steering.

Work Breakdown Structure

Following are the three phases planned for the project:

1. **Phase 1: Data Collection**
    - Shortlist control inputs and states to be measured/estimated
    - Collect dynamical dataset of an Ackerman-steered vehicle
    - Post-process the dataset to get required features and labels
2. **Phase 2: Model Training**
    - Define architecture of Bayesian neural network
    - Define/tune hyperparameters
    - Train and save the model
3. **Phase 3: Model Inference**
    - Forward-simulate the trained model by providing control inputs to it
    - Benchmark the model against ground-truth and comment on its validity

Phase 1: Data Collection

Data Collection Platform:

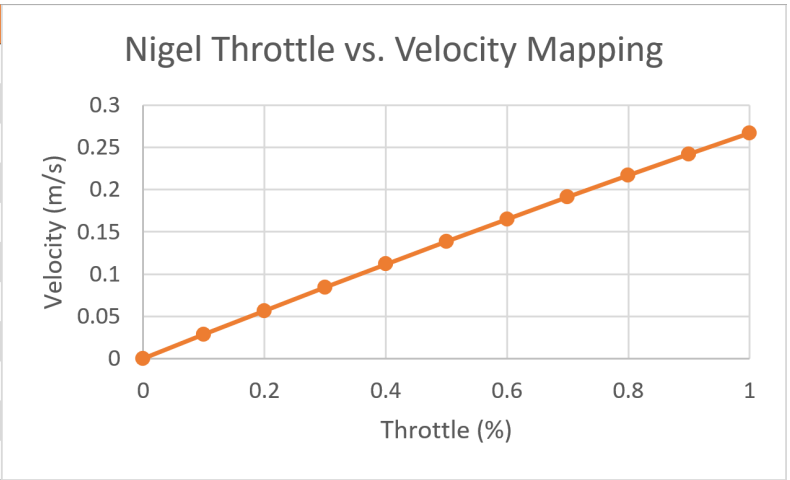
We employed AutoDRIVE Ecosystem for collecting the vehicle dynamics dataset. Particularly, we collected the data of AutoDRIVE's scaled vehicle (called Nigel) using the high-fidelity AutoDRIVE Simulator (from our initial exploration, this task seemd to demand a lot of data collection/filtering across various operating conditions that the vehicle can undergo. Consequently, we exploited this physically accurate and graphically realistic simulation tool to collect time-synchronized data in controlled conditions.)

All the collected dataset is made publicly available in the following GitHub repository: https://github.com/Tinker-Twins/AutoDRIVE-Nigel-Dataset

Following are some of the important vehicle parameters:

- Wheelbase (l): 0.1415 m
- Track width (w): 0.1530 m
- Throttle Limit: 1.0000 norm%
- Steering Limit: 0.5236 rad
- Linear Velocity Limit: 0.2670 m/s
- Angular Velocity Limit: 0.8051 rad/s
- Throttle vs. Velocity Mapping:

Throttle (%)	Velocity (m/s)
0	0
0.1	0.02867
0.2	0.05685
0.3	0.08457
0.4	0.11185
0.5	0.13871
0.6	0.16516
0.7	0.19122
0.8	0.21688
0.9	0.24215
1	0.26703



Notations:

Following is a brief summary of the notation used henceforth:

- **Vehicle Controls:** The low-level control inputs of the vehicle are throttle (τ) and steering (δ). Note that negative throttle indicates reverse driving and zero throttle indicates braking condition.
- **Vehicle States:** The most prominent vehicle states being considered are the positional coordinates along X-axis (x) and Y-axis (y) as well as the orientation about Z-axis, i.e. yaw angle (θ). Apart from these, the rate variables including longitudinal forward velocity (v) and angular velocity (ω) of the vehicle may also be considered.

Dataset Collection Process:

The data collection was partially automated through an `open_loop_controller.py` script, which is capable of selecting a maneuver (skidpad, fishhook, slalom) and its direction (ccw, cw), and controlling the vehicle actuators within the prescribed limits ($\tau \in [-1, 1]$ and $\delta \in [-0.5236, 0.5236]$) in an open-loop control setting.

The eight maneuvers were performed partially automatically (τ was set constant) and partially manually (δ was controlled using human-in-the-loop teleoperation framework offered by AutoDRIVE Simulator). The input method used for steering was a standard computer mouse, which allowed continuous variation of steering proportionate to the click-and-drag distance across screen.

Dataset Structure:

The recorded dataset contains an exhaustive set of features, which can be used not only for this project but also for other applications such as perception, state estimation, vehicle dynamics, motion planning, control, etc. We intentionally recorded realistically plausible sensor data from the simulator (i.e. data that can be sensed/estimated in real world as well; no unfair advantage of using simulation) to ensure the sim2real capability of our approach going forward. Following is a header describing the dataset structure:

DATA	timestamp	throttle	steering	leftTicks	rightTicks	posX	posY	posZ	roll	pitch	yaw	speed	angX	angY	angZ	accX	accY	accZ	cam0	cam1
UNIT	yyyy_MM_dd_HH_mm_ss_fff	norm%	rad	count	count	m	m	m	rad	rad	rad	m/s	rad/s	rad/s	rad/s	m/s^2	m/s^2	m/s^2	img_path	img_path

Dataset Preprocessing and Analysis:

The dataset for AutoDRIVE's scaled vehicle, Nigel, has been gathered from the high-fidelity AutoDRIVE Simulator for four distinct benchmark maneuvers:

- Skidpad
- Fishhook
- Slalom
- Eight

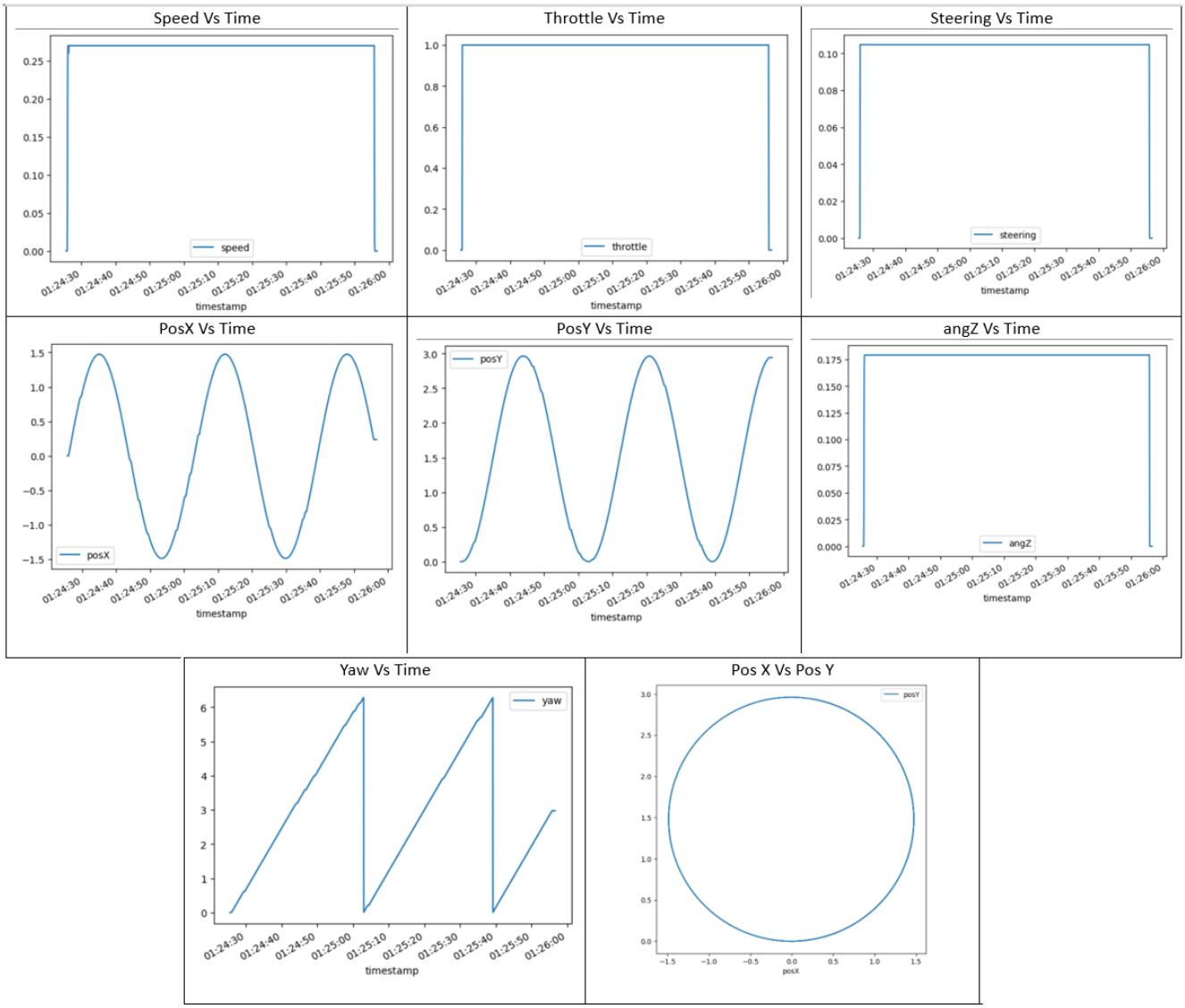
Upon preprocessing the raw data acquired from the AutoDRIVE Simulator for the various maneuvers executed within the simulation environment, the data integrity was analyzed/verified and visualizations were generated. The following sections briefly explain each of the benchmark maneuvers and the data visualization in each case.

1. Skidpad Maneuver

The skidpad test is vehicle dynamics test that is designed to measure a vehicle's lateral grip and handling capabilities. It involves driving a vehicle in a constant-radius circle around a circular track or pad.

The throttle of the vehicle was set to a constant value, $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}\,\mathrm{norm}\%$, and so was the steering angle, $\delta \in \{0.1047, 0.2094, 0.3142, 0.4189, 0.5236\}$ rad. This made the vehicle go in circles of different radii $\left(R = \frac{l}{\tan(\delta)}\right)$ based on the combination of sampled values.

Following figure depicts the visualization of low-level control inputs (τ, δ) as well as vehicle states $(x, y, \theta, v, \omega)$ for one sample run of skidpad maneuver:



2. Fishhook Maneuver

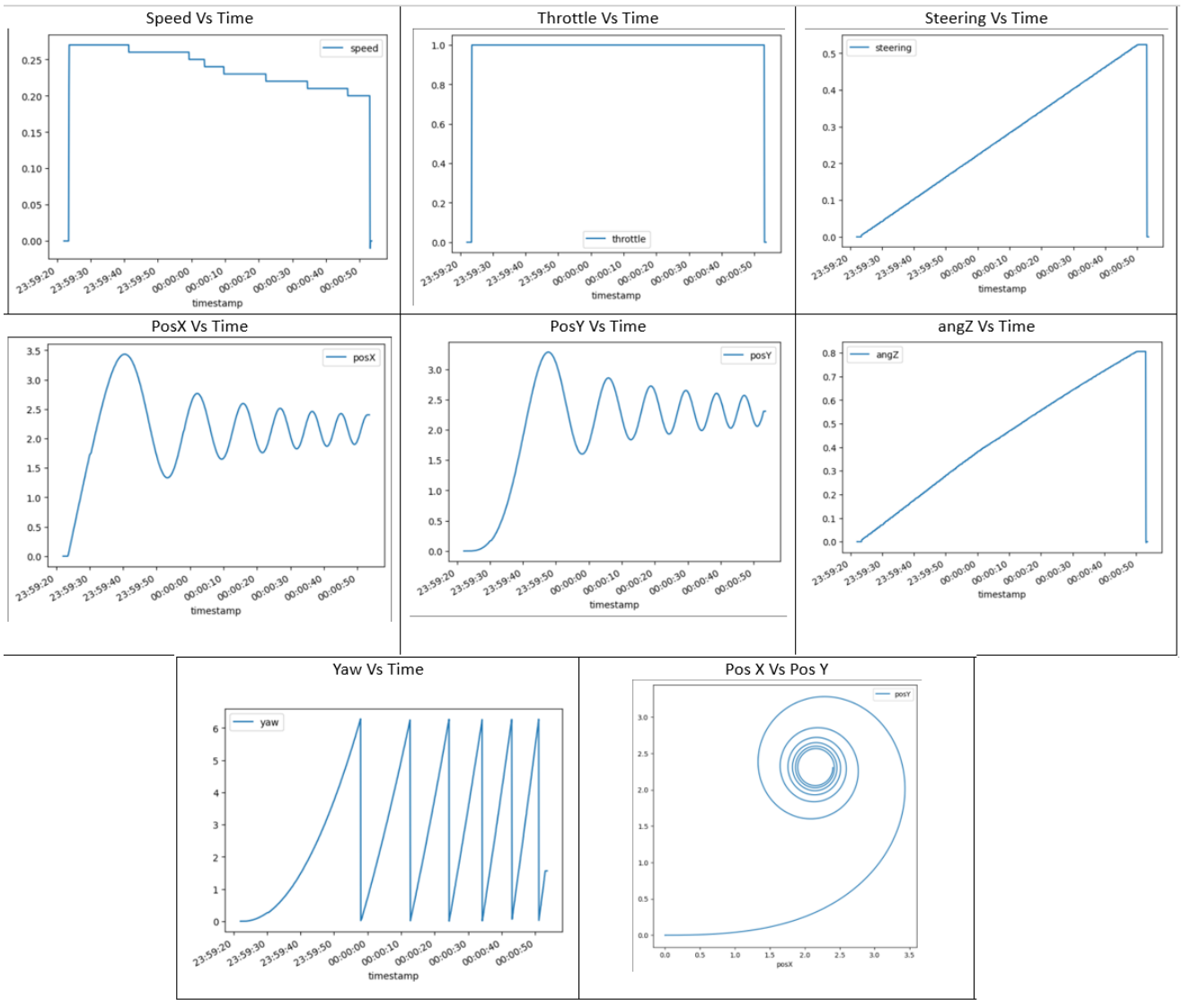
The fishhook test has a steering input defined in terms of angle against time. The test measures the steering gradient and sensitivity by making the vehicle go in spiral trajectory in one of the two following ways:

- Keeping the velocity constant and increasing the steering angle gradually.
- Keeping the steering angle constant and increasing the velocity gradually.

We adopted the former method.

The throttle of the vehicle was set to a constant value, $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}\,\mathrm{norm}\%$, while the steering angle was continuously incremented with a ramp input, $\delta = \min\left(k_{fishhook} * \mathrm{ramp}(t), \delta_{max}\right)$ rad based on the time (t) elapsed since initialization (90 s). This made the vehicle go in spirals of different radii.

Following figure depicts the visualization of low-level control inputs (τ, δ) as well as vehicle states $(x, y, \theta, v, \omega)$ for one sample run of fishhook maneuver:

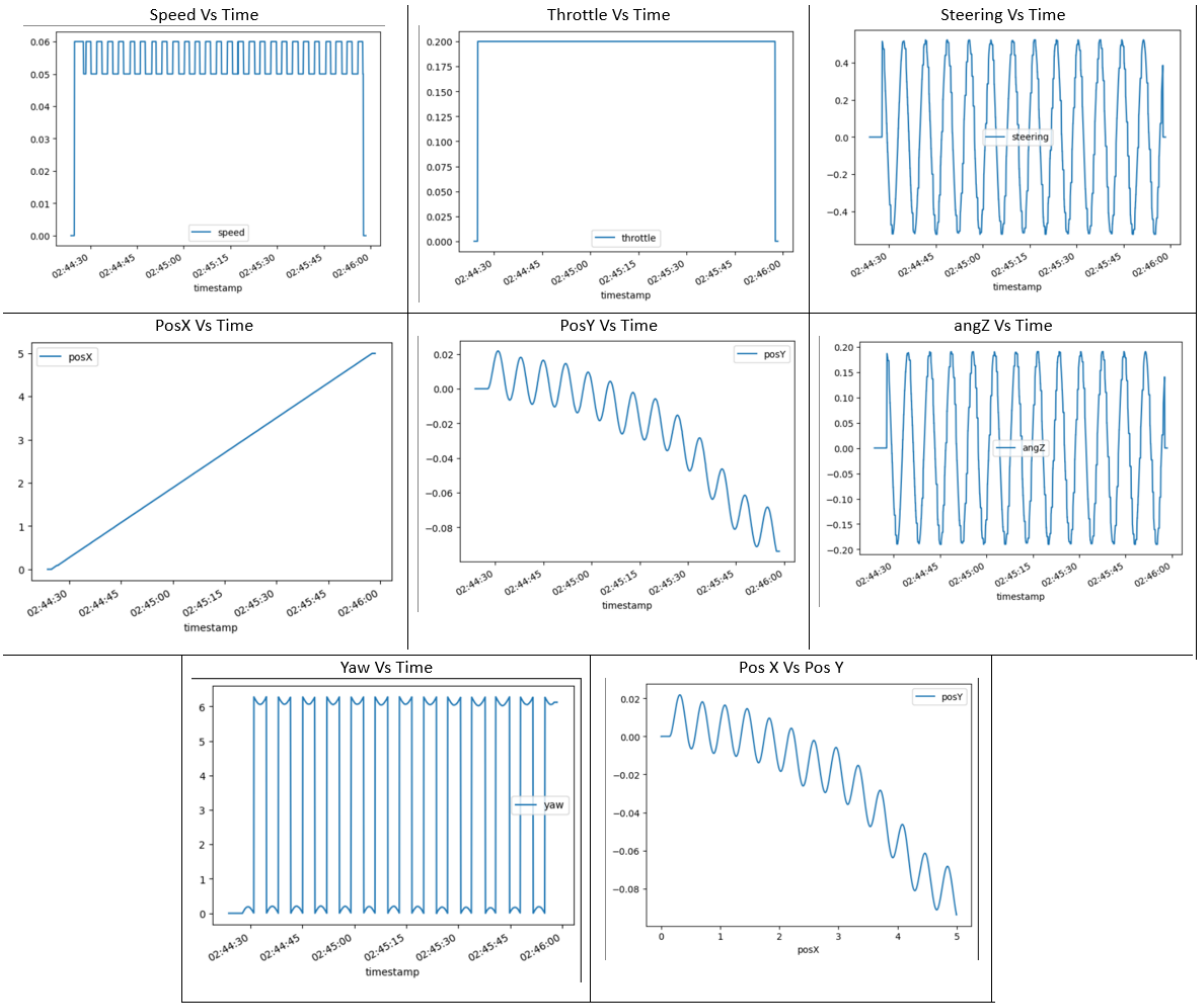


3. Slalom Maneuver

Slalom test is a driving exercise that involves maneuvering a vehicle through a series of cones or other obstacles set up in a zigzag pattern. The objective of the slalom test is to assess a driver's ability to handle the vehicle in a tight space and maintain control while making quick turns and changes of direction. It measures the vehicle's roll and yaw stability.

The throttle of the vehicle was set to a constant value, $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}\,\mathrm{norm}\%$, while the steering angle was controlled with a limit-scaled time-shifted sinusoidal input, $\delta = \delta_{lim} * \sin\left(\frac{\pi}{2} + t\right)$ rad based on the time (t) elapsed since initialization (90 s) and prescribed steering actuation limit $\delta_{lim} \in \{0.1047, 0.2094, 0.3142, 0.4189, 0.5236\}$. This made the vehicle go in sinusoidal trajectories of different amplitude.

Following figure depicts the visualization of low-level control inputs (τ, δ) as well as vehicle states $(x, y, \theta, v, \omega)$ for one sample run of slalom maneuver:

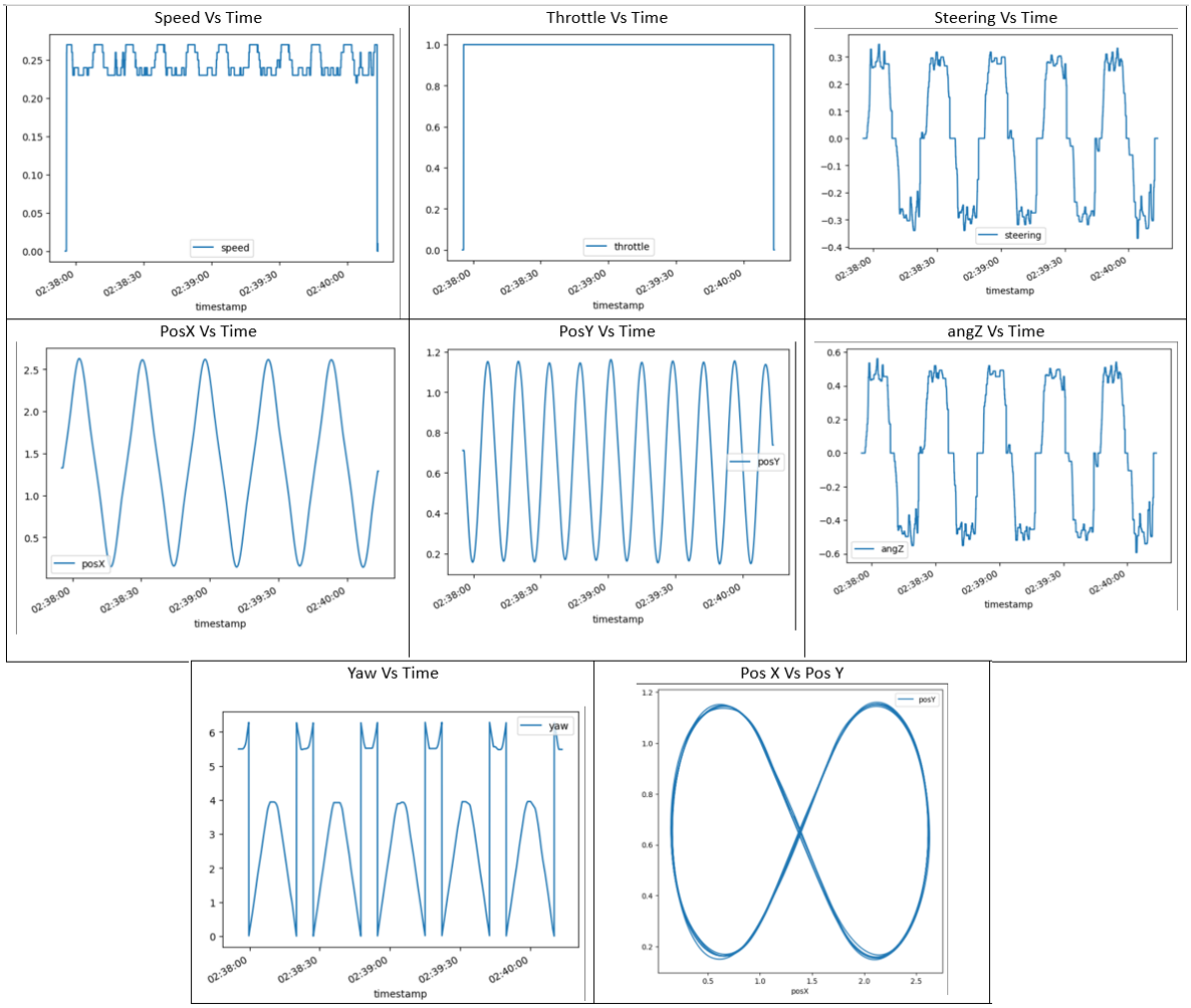


4. Eight Maneuver:

The 8-shape maneuver, also known as the figure-8 maneuver or 8-loop maneuver, is a common driving exercise used to improve a driver's handling and control of a vehicle. The maneuver consists of driving in a figure-eight pattern, which requires the driver to make continuous turns in both directions, as well as speed and direction changes.

As described earlier, the eight maneuvers were performed partially automatically (τ was set constant, $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}\ \mathrm{norm}\%$) and partially manually (δ was controlled using the human-in-the-loop teleoperation framework offered by AutoDRIVE Simulator). The input method used for steering was a standard computer mouse, which allowed continuous variation of steering proportionate to the click-and-drag distance across screen.

Following figure depicts the visualization of low-level control inputs (τ, δ) as well as vehicle states $(x, y, \theta, v, \omega)$ for one sample run of eight maneuver:



Project Progress and Next Steps

Phase 1: Data Collection

The dataset for AutoDRIVE's scaled vehicle, Nigel, has been gathered from the high-fidelity AutoDRIVE Simulator for four distinct benchmark maneuvers: skidpad, fishhook, slalom and eight. **This marks the completion of Phase 1 of the project.** Based on the model's outcomes, there may be a necessity to collect additional data for one or more maneuvers.

Phase 2: Model Training

We have already started exploring the concepts of Bayesian Neural Networks (BNNs).

BNNs, given an input, predict output with certainty. In other words, they learn the parameters of a probability distribution function (PDF) $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ from which their parameters (weights and biases) are sampled $w_i,\ b_i \sim \mathcal{N}(\mu_i, \sigma_i)$.

Consider a BNN Model: $F_\theta()$ provided with an input vector: $\boldsymbol{x}$, and producing an output: $\boldsymbol{y}$, trained on a dataset: $\boldsymbol{D} = (x_i,\ y_i)_{i=1}^{n}$ with a certain loss function: $\mathcal{L}()$. Let the BNN have weights: $\boldsymbol{w}$, bias: b and activation function: $g()$.

Then, the objective of the BNN is to learn $\{\mu_i,\ \sigma_i\}$ such that its parameters when sampled from a normal distribution parameterized by these values $w_i,\ b_i \sim \mathcal{N}(\mu_i, \sigma_i)$ will produce the desired output(s).

In theory, the training of BNN can be thought of as maximizing the log likelihood of predictions over a given dataset in conjunction with the KL-divergence term:

$$\mu^*,\ \Sigma^* = \underset{\mu,\ \Sigma}{\operatorname{argmax}} \sum_{(x_i,\ y_i)\ \in\ D} \log[p(y_i|x_i,\ \theta)] - KL[p(\theta),\ p(\theta_0)]$$

where, $\theta \sim \mathcal{N}(\mu, \Sigma)$ and $\theta_0 \sim \mathcal{N}(0, \mathrm{I})$

However, practically, this is achieved by minimizing a loss function which quantifies the "error" in predictions w.r.t. data-truth, in conjunction with the KL-divergence term:

$$\mu^*,\ \Sigma^* = \underset{\mu,\ \Sigma}{\operatorname{argmin}} \sum_{(x_i,\ y_i)\ \in\ D} \mathcal{L}(F_\theta(x_i),\ y_i) + KL[p(\theta),\ p(\theta_0)]$$

The prediction step can be theoretically understood as integrating the predictions of the BNN over the entire PDF, given the optimal parameters $\theta^* \sim \mathcal{N}(\mu^*, \Sigma^*)$:

$$p(\hat{y} \mid \hat{x}, D) = \int p(\hat{y} \mid \hat{x}, \theta^*)\ p(\theta^*|D)\ d\theta$$

However, practically this is just a forward pass of the neural network $\hat{y} = F_{\theta^*}(\hat{x})$ averaged over multiple instances:

$$\hat{y} = \frac{1}{K} \sum_{k=1}^{K} F_{\theta_k^*}(\hat{x})$$

During the back-propagation step, BNNs face a challenge of intractability since derivatives of the parameters being learnt (i.e. derivatives of distributions) must be calculated:

$$p(\hat{y} \mid \hat{x}, D) = \int p(\hat{y} \mid \hat{x}, \theta^*)\, p(\theta^* | D)\, d\theta$$

A work-around solution to this problem is the local re-parameterisation trick, which "moves" the parameters to be learnt (μ, σ in case of a Gaussian distribution), out of the distribution function for any weight w.

We know that $\theta = (\mu, \sigma)$. Now let $\epsilon \sim \mathcal{N}(0, 1)$, i.e. a variable that follows standard normal distribution. We can then express the weights as a linear combination of the normal distribution parameters (μ, σ) over which it was being sampled from: $f(\epsilon) = w = \mu + \sigma \cdot \epsilon$

We can now compute derivatives of weights and biases w.r.t. this surrogate function $f(\epsilon)$ and update the parameters to eventually get the optimal parameters $\theta^* = (\mu^*, \sigma^*)$:

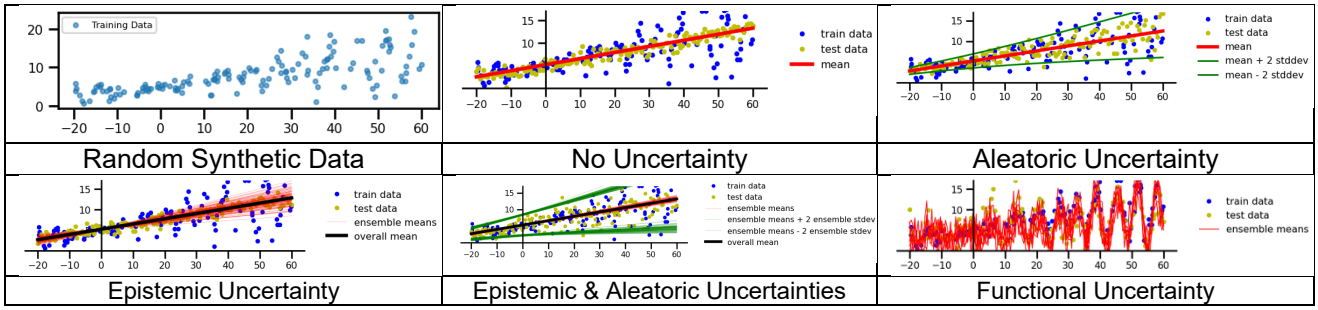
$$\Delta\mu = \frac{\partial f}{\partial w} + \frac{\partial f}{\partial \mu}$$

$$\Delta\sigma = \frac{\partial f}{\partial w}\frac{\epsilon}{\sigma} + \frac{\partial f}{\partial \sigma}$$

$$\mu := \mu - \alpha\Delta\mu$$

$$\sigma := \sigma - \alpha\Delta\sigma$$

We have also started trying out some toy examples using BNNs. Following are some preliminary results:



Random Synthetic Data | No Uncertainty | Aleatoric Uncertainty

Epistemic Uncertainty | Epistemic & Aleatoric Uncertainties | Functional Uncertainty

For this project, the high level architecture of the BNN is planned to take in low level control inputs and predict the vehicle states. Apart from the generic application of forward-simulating the model to predict states, this approach will be very useful in state estimation of autonomous vehicles and some its novel applications are as follows:

1. Ability to replace a hardware sensor by software model, which can reduce the production cost of the vehicle.
2. State-estimation in coverage denied areas or in sensor failure cases.
3. Redundant variable measurement for more robust state-estimation.
4. Predicting the anomalies in the sensor data stream using the model.

Phase 3: Model Inference

This phase of the project will be performed after getting satisfactory results from Phase 2.