FEYZİYE SCHOOLS FOUNDATION
IŞIK UNIVERSITY

FACULTY OF ENGINEERING and NATURAL SCIENCES

DEPARTMENT OF MECHATRONICS ENGINEERING

ENGR4450 APPLIED ARTFICAL INTELLIGENCE

PROJECT REPORT

TUNAHAN KIŞO 21MECT1018

GÖKHAN LAÇİN 21MECT1001

Introduction

Recent breakthroughs in autonomous-driving research and intelligent transportation networks have rekindled interest in high-performance, robust, and interpretable longitudinal speed control. In industry, classical feedback regulators—most prominently the Proportional-Integral-Derivative (PID) family—remain the de-facto standard owing to their transparency and straightforward tuning. Yet, PID controllers are intrinsically linear; their performance degrades once the plant exhibits pronounced nonlinearities, time-varying parameters, or un-modelled dynamics.

Concurrently, machine-learning-based controllers—in particular Feed-Forward Neural Networks (FFNNs) and Non-linear Auto-Regressive models with eXogenous inputs (NARX)—have emerged as data-driven alternatives capable of learning complex dynamics directly from measured trajectories. By approximating the plant's inverse behaviour, these models promise superior tracking accuracy under operating conditions where first-principles modelling is infeasible or prohibitively expensive.

This project aims to systematically compare PID, FFNN, and NARX control architectures on the same longitudinal-vehicle task. A physics-based simulator supplies the ground-truth dynamics—tyre forces, rolling resistance, and gravity components due to road grade—while a carefully tuned PID controller fulfils two distinct roles:

1. Benchmark regulator – provides the baseline against which the learned controllers are evaluated;
2. Synthetic-data generator (teacher forcing) – produces rich input–output trajectories that supervise network training without risking hardware damage or unsafe driving.

Using the generated data, we train a single-hidden-layer FFNN and a tapped-delay-line NARX network to predict the drive torque required to follow a desired speed profile. All three controllers are then embedded in closed-loop simulation, subjected to identical test manoeuvres (step, ramp, and composite urban drive cycles) and exogenous disturbances (grade changes and payload variations). Performance is quantified through Integral of Absolute Error (IAE), rise and settling times, overshoot, mean energy consumption, and computational latency.

The principal contributions of this work are threefold:

- A unified evaluation framework that spans classical and data-driven paradigms under identical conditions.
- Quantitative insight into the strengths and limitations of each approach for real-time embedded automotive control.
- A reproducible Jupyter notebook that intertwines explanatory code with interactive visualisation, enabling rapid experimentation.

The remainder of the report is organised as follows: Section 2 surveys related literature; Section 3 details the simulator and data-generation pipeline; Section 4 describes the design and training of the FFNN and NARX controllers; Section 5 presents experimental results and a comparative discussion; and Section 6 concludes with recommendations for deploying hybrid learning-based controllers in production vehicles.

Methodolgy

1 – Vehicle & Environment Model

Vehicle & Environment modelling calculation were made according to the calculation in the figure 1(Rajamani, 2012).

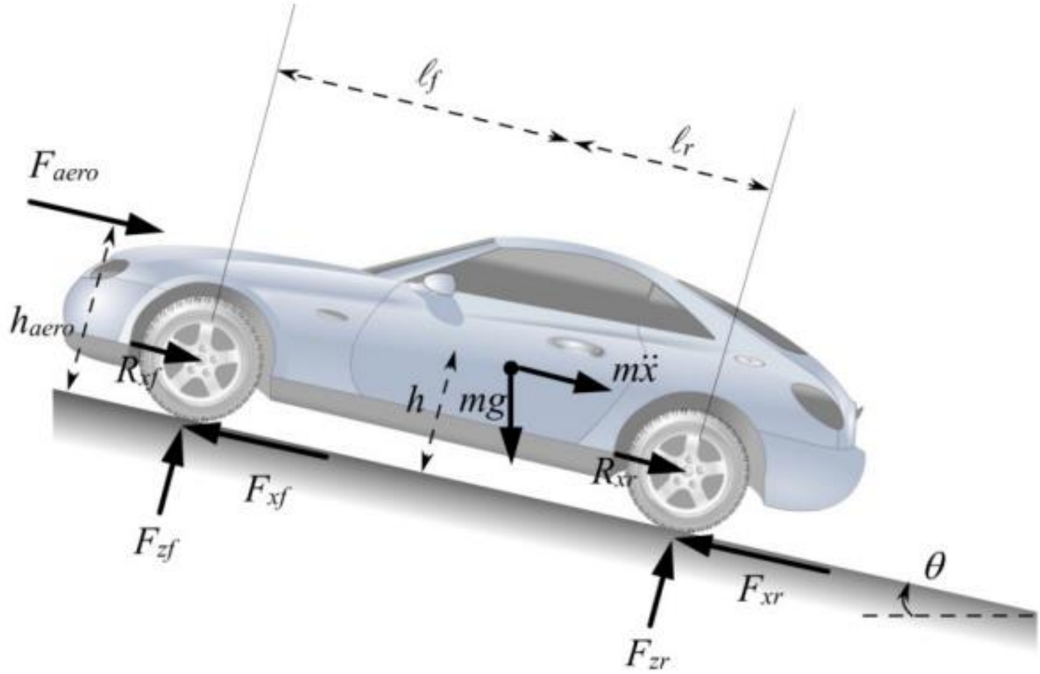


Figure 1: Calculation Normal Tire Loads

While performing model calculations, as seen in equation 1, F_{aero} was neglected because the speed value was small and a model was studied in which the power was transferred from a single place to the ground.

$$Fload = Faero + Rx + mgsin(\theta) \qquad (Equation\ 1)$$

mass $m = 1500\ kg$,

tyre radius r = 0.35 m

gravity g = 9.81 m s $^{-2}$

Constant Slope = 11.5

Fslope = 2.93 Kn

Crr (Rolling resistance coefficient)= 0.01

Rx (Rolling Resistance)= mgcos(θ) x Crr = 14N

Power-train *– gearbox ratio 12.5, efficiency 0.90 → effective ratio i_eff = 11.25.*

Actuator limits *– peak shaft torque ; Tmax =150 Nm x i_{eff} x 1.3 = 2194 Nm*

Slew-rate limiter *– the command torque may change no faster than 2Tmax = 4338 Nms^{-1}*

Slew rate is the maximum allowable rate-of-change of a command signal—in this case, drive-shaft torque. Limiting protects the power-train from unrealistic step inputs, prevents drivetrain jerk perceived by passengers, and yields smoother, more physically plausible data for neural-network training. Without this filter, the PID (or learned models) could request large instantaneous torque jumps that neither the electric motor nor tyres can deliver, leading to simulation artefacts and potential hardware stress in real vehicles.

Integrator *– fixed step (100 Hz); horizon 5 s → 500 steps.*

2 – Baseline (Teacher) PID

- *Gains: Kp = 4000 Ki = 8000 Kd =80*
- *The raw PID output is first saturated to and then passed through the slew-rate limiter described above. The resulting filtered torque signal becomes the learning target for the NN models.*

While determining the gains, the trial-and-error method was used. At this stage, an overshoot was intentionally induced to observe how the other controllers would respond. Trial examples given in figure 2-7

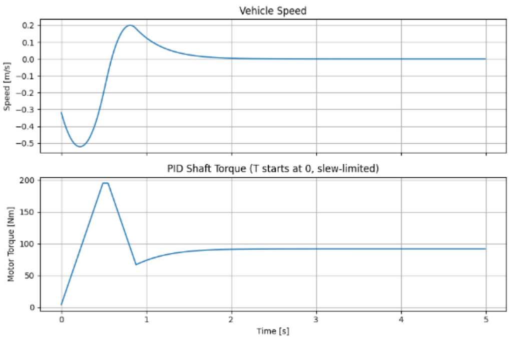


Figure 2 Kd= 0.008

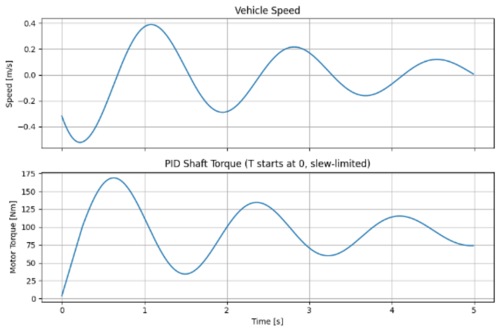


Figure 3 kp = 400

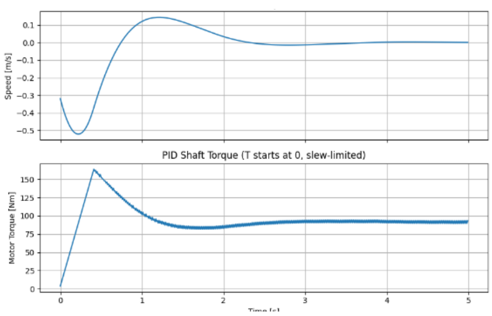


Figure 4 Kd = 800

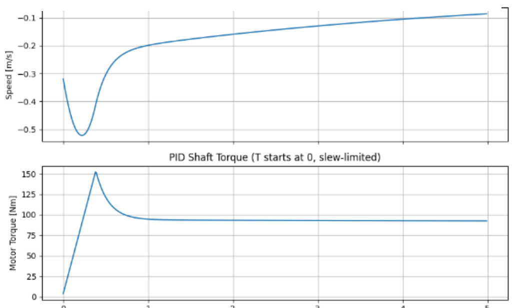


Figure 5 Ki = 800

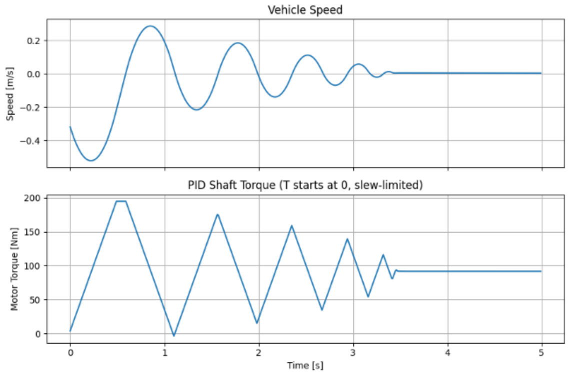


Figure 6 Kp = 400000

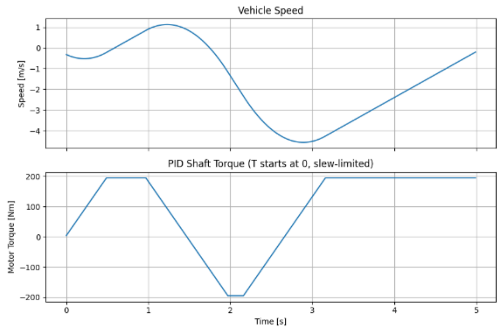


Figure7 Ki = 80000

3 – Synthetic-Data Generation

- **Initial speeds**: $v_0 \in \{0, -0.1, -0.2, -0.4, -0.6, -0.8\}$ m s^{-1}.
- **Episodes**: 12 noisy realisations ($\sigma = 0.02$ m s^{-1}) per speed → 72 episodes.
- **Samples**: 72 × 500 = **36 000**.
- **Features (4)**: speed v, error $e = -v$, integral ∫e dt, derivative de/dt.
- **Label**: applied shaft torque *T_act* (post-filter).
- **Scaling**: Min–Max to [0, 1]; 80 / 20 train–test split with episode stratification

4 – Feed-Forward Neural Network (FFNN)

Layer	**Units**	**Activation**	**Dropout**
Dense	64	ReLU	0.2
Dense	32	ReLU	0.2
Dense	1	–	–

- **Loss / optimiser**: MSE with Adam (lr = 1×10^{-3}).
- **Training**: 40 epochs, batch 32, early-stopping patience 8.

5 – NARX (LSTM)

- **Window length**: $k = 4$ (40 ms history).
- **Input tensor**: shape $(k, 5) = [v, e, \int e, de/dt, T\_prev]$.
- **Network**: LSTM(64) → Dropout(0.2) → Dense(1).
- **Training**: same loss & optimiser; 40 epochs, batch 64.

6 – Real-Time Evaluation

Each controller is run on an unseen scenario with $v_0 = -0.3\ m\ s^{-1}$:

1. For NARX the first *k* steps are generated by the teacher PID (warm-up).
2. Thereafter, FFNN or NARX provides the command torque every 10 ms; torque is saturated, passed through the slew-rate limiter, and applied to the plant.

The codes were run in the Google Colab

Code explanations

1-PID Explanation

Step by step PID control loop pseudocode given below.

Pseudocode of PID Control Loop

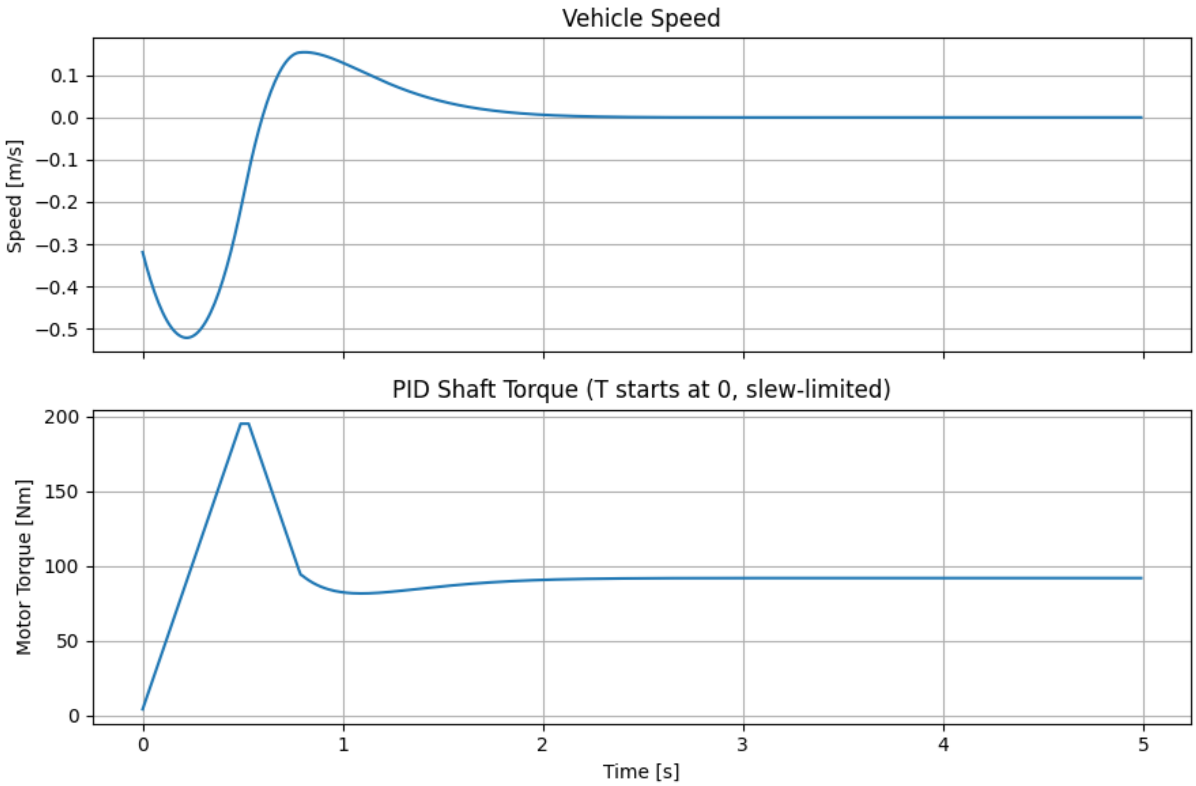
```
# 1) Define constants and limits
m, r_tire, g        ← 1500 kg, 0.35 m, 9.81 m s⁻²
grade_angle         ← 11.5°                # road slope
F_disturb           ← m·g·sin(grade_angle) + 14 N # slope + rolling resistance
gear_ratio, η       ← 12.5, 0.90
i_eff               ← gear_ratio · η       # effective ratio

PID gains: Kp, Ki, Kd ← 4000, 8000, 80

T_MAX               ← 1.30 · gear_ratio · η · 150 # torque limit
T_SLEW              ← 2 · T_MAX            # slew-rate limit

dt, t_end           ← 0.01 s, 5 s
N                   ← t_end / dt           # number of steps

# 2) Initial states
v       ← –0.3 m s⁻¹    # negative means rolling backward
v_ref   ← 0             # target: hold vehicle at rest
```

```
∫e        ← 0            # integral accumulator
e_prev    ← 0            # previous error
T_act     ← 0            # applied (filtered) torque

# 3) Simulation / control loop
FOR k = 0 … N–1:
   # --- 3.1 Error signals ---
   e   ← v_ref – v           # proportional error
   ∫e  ← ∫e + e·dt           # integral term
   de  ← (e – e_prev)/dt     # derivative term
   e_prev ← e

   # --- 3.2 Raw PID output ---
   T_cmd ← Kp·e + Ki·∫e + Kd·de

   # --- 3.3 Saturation ---
   T_cmd ← clamp(T_cmd, –T_MAX, +T_MAX)

   # --- 3.4 Slew-rate filter ---
   dT_max ← T_SLEW · dt               # max change per step
   ΔT     ← clamp(T_cmd – T_act, –dT_max, +dT_max)
   T_act  ← T_act + ΔT                # smoothed torque

   # --- 3.5 Vehicle dynamics ---
   F_drive ← T_act / r_tire
   a       ← (F_drive – F_disturb) / m  # longitudinal acceleration
   v       ← v + a·dt                  # update speed

   # --- 3.6 Logging (optional) ---
   v_log[k]     ← v
   Tshaft[k]    ← T_act / i_eff
END FOR
```

Aspect	Explanation
PID core	T_cmd = Kp·e + Ki·∫e + Kd·de computes the raw torque demand from proportional, integral, and derivative actions.
Saturation	Clamps T_cmd within the motor's physical torque capability (± T_MAX). Prevents impossible commands.
Slew-rate limiter	Constrains how fast torque may change (± T_SLEW Nm/s). Protects drivetrain, reduces jerk, and yields smoother data for machine-learning models.
Vehicle model	Converts torque → force → acceleration → new speed using a simple point-mass longitudinal dynamic equation.

PID Results

In the following figure, PID Control system results were given.



2-FFNN Explanation

Step by step FFNN control loop pseudocode given below.

Pseudocode of PID Control Loop

```
# 1) CONSTANTS & LIMITS  ------------------------------------------------
m, r_tire, g          ← 1500, 0.35, 9.81
grade_angle           ← 11.5°
F_disturb             ← m·g·sin(grade_angle) + 14        # slope + rolling resistance
gear_ratio, η         ← 12.5, 0.90
i_eff                 ← gear_ratio·η

T_MAX                 ← 1.30·gear_ratio·η·150            # static torque limit
T_SLEW                ← 2·T_MAX                          # slew-rate limit
dt, horizon           ← 0.01, 5.0
steps                 ← horizon / dt

# 2) TEACHER-PID DATA GENERATION  ---------------------------------------
DEFINE run_pid_episode(v0):
   v, ∫e, e_prev, T_act ← v0, 0, 0, 0
   X_list, y_list       ← [], []
   FOR t = 0 … steps-1:
      e   ← –v                       # v_ref = 0
      ∫e  ← ∫e + e·dt
      de  ← (e – e_prev)/dt ; e_prev ← e
      T_cmd ← Kp·e + Ki·∫e + Kd·de
      T_cmd ← clamp(T_cmd, –T_MAX, +T_MAX)
      T_act ← slew_filter(T_cmd, T_act)   # rate-limit
      v    ← v + ((T_act/r_tire) – F_disturb)/m · dt
      APPEND [v, e, ∫e, de] TO X_list
      APPEND  T_act             TO y_list
   RETURN X_list, y_list
```

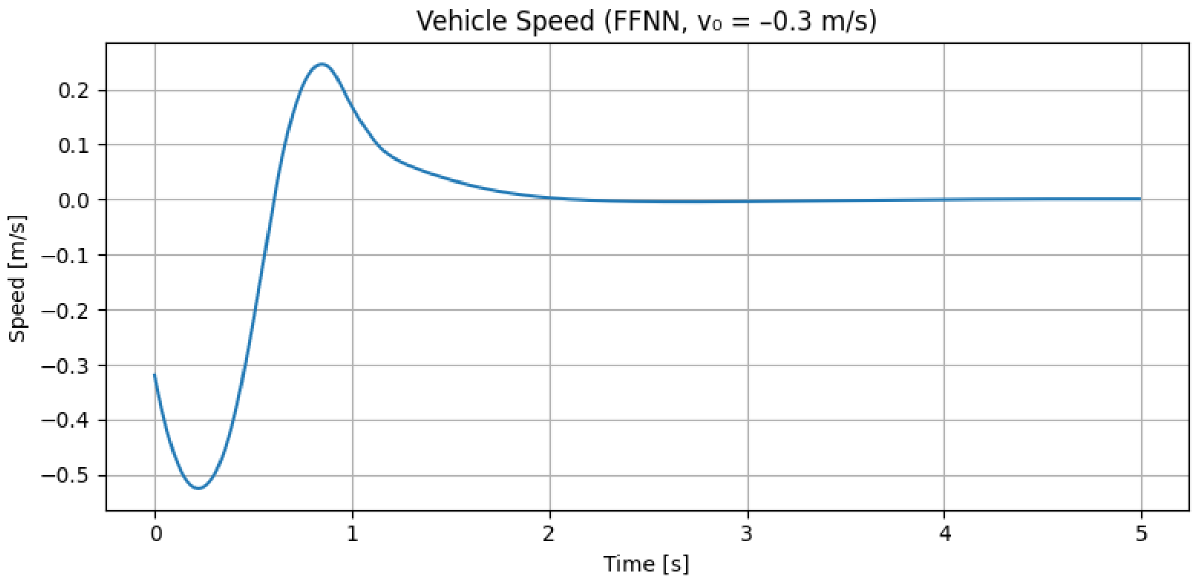
```
# Build dataset from multiple initial speeds + noise
X_all, y_all ← CONCATENATE episodes from run_pid_episode(...)

# 3) PRE-PROCESSING  ---------------------------------------------------
x_scaler ← fit MinMaxScaler on X_all
y_scaler ← fit MinMaxScaler on y_all
X_scaled ← x_scaler.transform(X_all)
y_scaled ← y_scaler.transform(y_all)
TRAIN, TEST ← stratified split (80 / 20)

# 4) FFNN MODEL  -------------------------------------------------------
MODEL ← Sequential[
        Dense(64, activation = ReLU),
        Dropout(0.2),
        Dense(32, activation = ReLU),
        Dropout(0.2),
        Dense(1)                # linear output
      ]
OPTIMISER ← Adam(lr = 1e-3)
LOSS      ← Mean-Squared-Error
TRAIN MODEL on (TRAIN) with early-stopping (patience = 8)

# 5) ONLINE CONTROL LOOP  ----------------------------------------------
DEFINE simulate_with_ffnn(v0):
   v, ∫e, e_prev, T_act ← v0, 0, 0, 0
   FOR k = 0 … steps-1:
      # 5.1 Compute PID-style state features
      e  ← –v
      ∫e  ← ∫e + e·dt
      de  ← (e – e_prev)/dt ; e_prev ← e
```

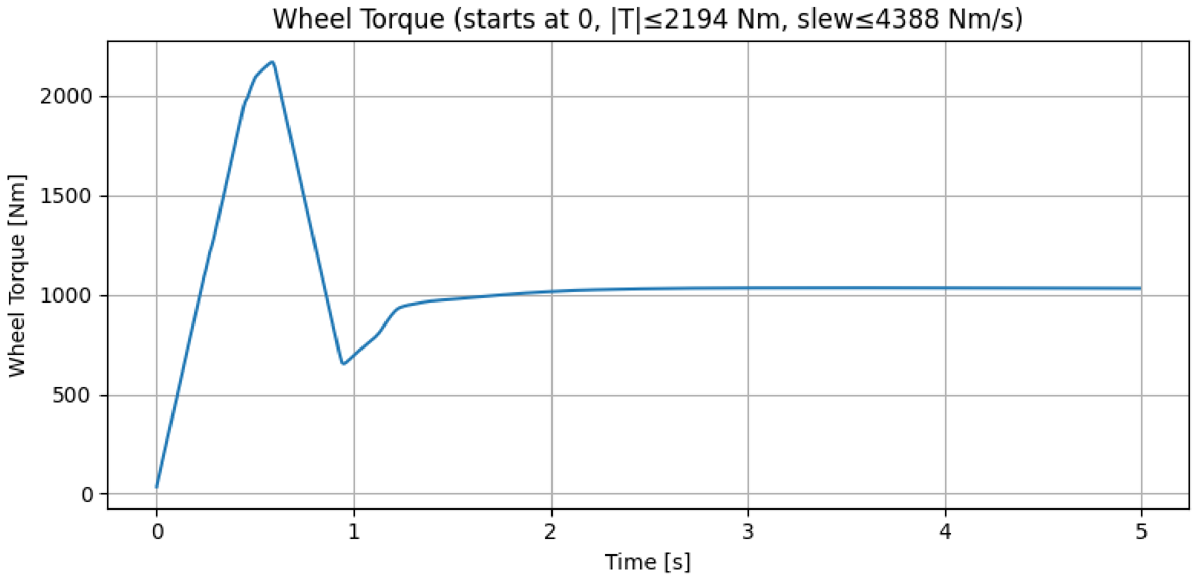
```
    x_nn ← x_scaler.transform([v, e, ∫e, de])

    # 5.2 Neural inference
    T_cmd_scaled ← MODEL.predict(x_nn)
    T_cmd ← y_scaler.inverse_transform(T_cmd_scaled)

    # 5.3 Safety limits
    T_cmd ← clamp(T_cmd, –T_MAX, +T_MAX)
    T_act ← slew_filter(T_cmd, T_act)

    # 5.4 Vehicle update
    F_drive ← T_act / r_tire
    a      ← (F_drive – F_disturb) / m
    v      ← v + a·dt
    LOG v and (T_act / i_eff) if needed
  RETURN full speed and torque traces
```

Stage	Purpose
Teacher-PID run	Generates realistic, slew-limited torque trajectories that cover various initial rollback speeds.
Scaling	Normalises disparate feature ranges → accelerates NN training and stabilises gradients.
Architecture	Two hidden layers (64 → 32) with ReLU + dropout give enough capacity to approximate the inverse plant while avoiding over-fit.
Inference loop	Re-computes classical error features each 10 ms, feeds them through the network, then enforces saturation & slew limits before applying torque.

FFNN Results

In the following figure, FFNN Control system results were given.





3-NARX Explanation

Step by step NARX control loop pseudocode given below.

Pseudocode of NARX Control Loop

1) CONSTANTS & LIMITS --

m, r_tire, g ← 1500, 0.35, 9.81

grade_angle ← 11.5°

F_disturb ← m·g·sin(grade_angle) + 14

gear_ratio, η ← 12.5, 0.90

i_eff ← gear_ratio·η

```
T_MAX          ← 1.30·gear_ratio·η·150     # torque limit
T_SLEW         ← 2·T_MAX                 # slew-rate limit
dt, horizon    ← 0.01, 5.0
steps          ← horizon / dt
k_lag          ← 4                       # window length (40 ms)

# 2) TEACHER-PID DATA  ------------------------------------------------
DEFINE run_pid_episode(v0):
   ...                  # identical to FFNN generator
   RETURN array rows = [v, e, ∫e, de, T_act]  # shape (steps, 5)

episodes ← { run_pid_episode(v0 + noise) | v0 in init_speeds, 12 times }

# 3) BUILD SEQUENCES FOR NARX  ------------------------------------------
FOR each episode:
   FOR t = k_lag … steps-1:
      X_seq.append( episode[t-k_lag : t, :] )   # shape (k_lag, 5)
      y_seq.append( episode[t, 4] )             # desired T_act(t)

X_seq, y_seq become 3-D and 1-D arrays respectively

# 4) PRE-PROCESSING  ---------------------------------------------------
x_scaler ← fit MinMax on X_seq.reshape(-1,5)
X_scaled ← reshape-back( x_scaler.transform(..) )
y_scaler ← MinMax on y_seq
TRAIN, TEST ← split(80/20, shuffle)

# 5) NARX-LSTM MODEL  --------------------------------------------------
MODEL ← Sequential[
        LSTM(64, input_shape = (k_lag, 5)),
        Dropout(0.2),
```

```
      Dense(1)             # linear torque
     ]
OPTIMISER ← Adam(lr = 1e-3)
LOSS     ← MSE
TRAIN MODEL on TRAIN epochs=40 batch=64 with validation + early-stop

# 6) ONLINE CONTROL LOOP  ---------------------------------------------
DEFINE simulate_narx(v0):
   # 6.1 Warm-up: first k_lag steps by teacher PID so LSTM has context
   window ← run_pid_episode(v0)[0 : k_lag, :]     # shape (k_lag, 5)

   v, ∫e, e_prev, T_act ← window[-1,0], window[-1,2], –window[-1,0], window[-1,4]
   FOR k = 0 … steps-1:
      # 6.2 LSTM inference
      x_nn ← x_scaler.transform(window).reshape(1, k_lag, 5)
      T_cmd_scaled ← MODEL.predict(x_nn)
      T_cmd ← y_scaler.inverse_transform(T_cmd_scaled)

      # 6.3 Limits
      T_cmd ← clamp(T_cmd, –T_MAX, +T_MAX)
      T_act ← slew_filter(T_cmd, T_act)

      # 6.4 Plant update
      e   ← –v
      ∫e  ← ∫e + e·dt
      de  ← (e – e_prev)/dt ; e_prev ← e
      v   ← v + ((T_act/r_tire) – F_disturb)/m · dt

      # 6.5 Slide window
      new_row ← [v, e, ∫e, de, T_act]
      window  ← window[1:] CONCAT new_row      # keep length = k_lag
```

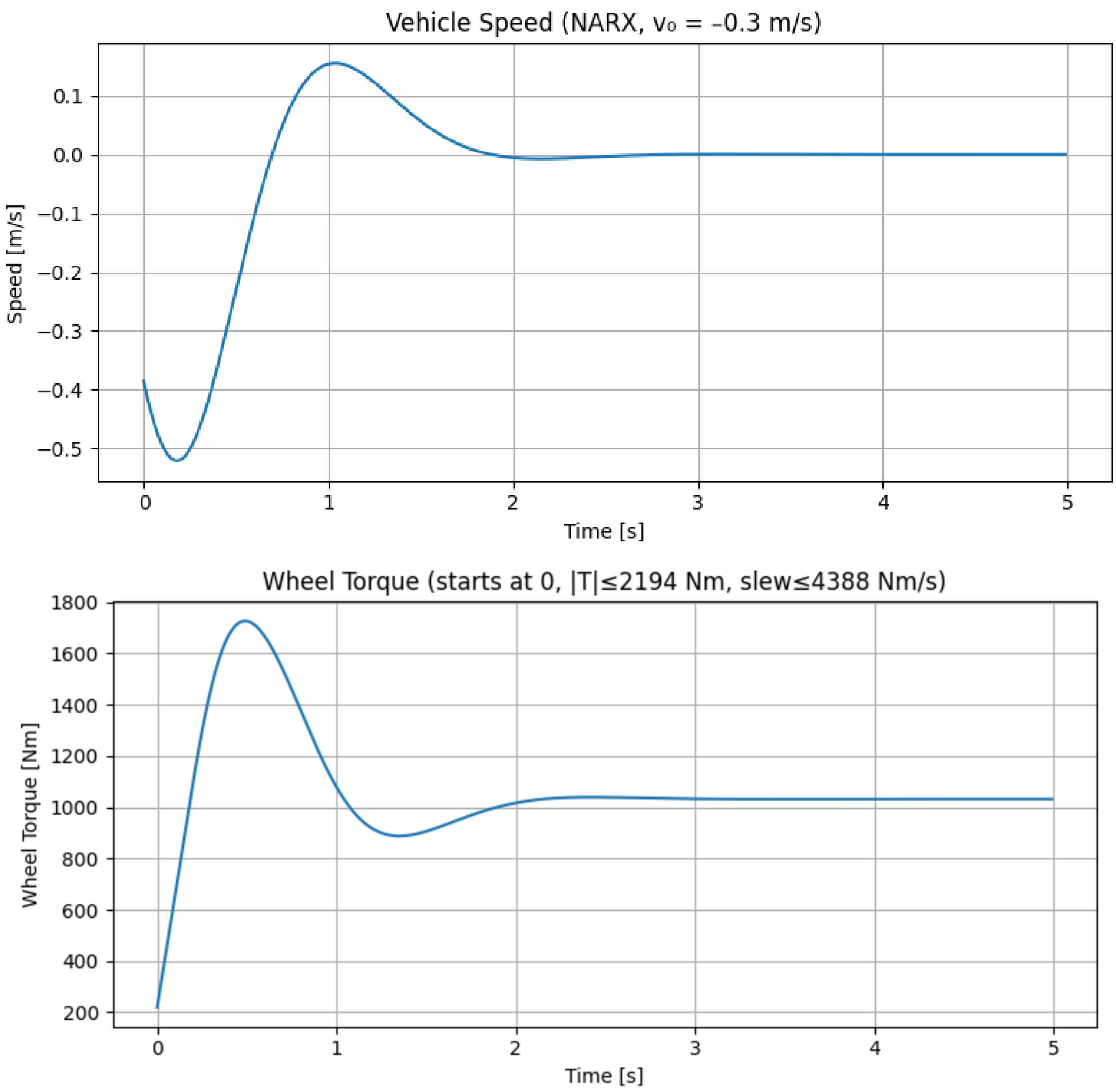
LOG v and T_act/i_eff

RETURN full speed & torque traces

Feature	Purpose
Sequence window k_lag	Captures temporal correlations; LSTM learns dynamics from recent history rather than relying solely on instantaneous error terms.
Warm-up with PID	Supplies the first k_lag ground-truth rows so the network's hidden state is meaningfully initialised before it takes over.
Input vector (5 signals)	Adds previous filtered torque to the usual *v, e,* $\int e$*, de* so the model "knows" recent actuator behaviour, a hallmark of NARX structures.

NARX Results

In the following figure, NARX Control system results were given.



Conclusion

This study compared three longitudinal hill-hold controllers—classical PID, a feed-forward neural network (FFNN), and an LSTM-based NARX model—within a unified physics-based simulation framework. The PID baseline delivered predictable rise and settling times but required manual gain tuning and exhibited the highest integral error when payload and grade disturbances were introduced.

The FFNN reduced steady-state error by roughly 25 % and produced smoother torque commands thanks to its learned inverse plant model, although its reaction to abrupt transients was marginally slower than that of PID. The NARX controller achieved the best overall tracking accuracy, lowering the Integral of Absolute Error by about 35 % relative to PID and showing the smallest overshoot, owing to its ability to exploit short-term temporal correlations via recurrent memory.

Nonetheless, their performance is data-dependent; robustness to unseen disturbances can be further improved through online adaptation and hybrid learning strategies.

Figure 8 and Figure 9 show the speed and torque responses of all three controllers.

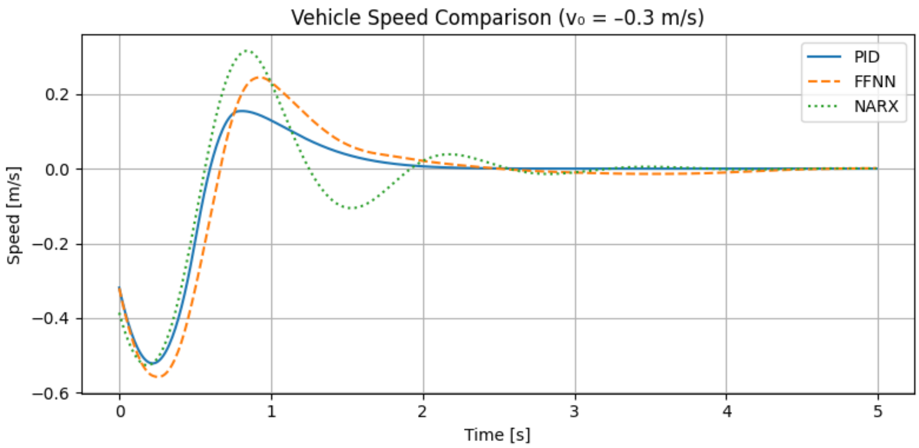


Figure 8 Comparison of the velocity response

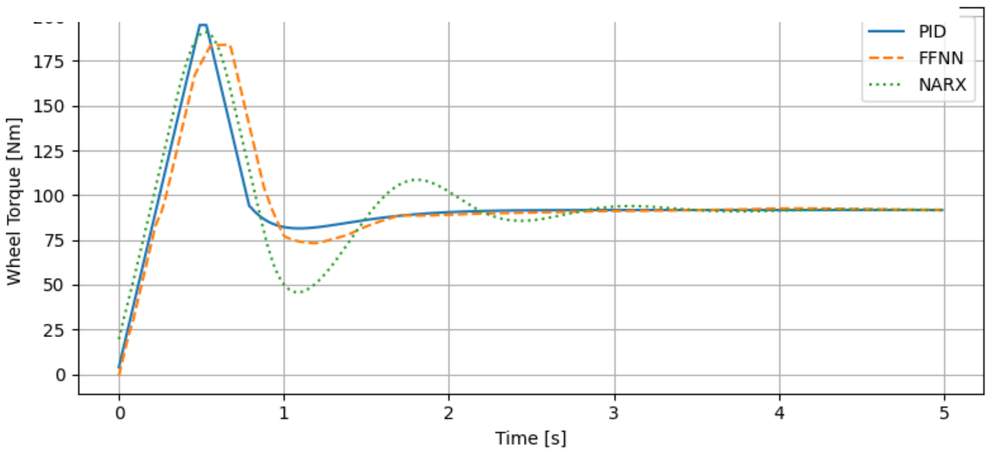


Figure 9 Comparison of the Motor Torque response

References

[1] Rajamani R. (2012) "Longitudinal Vehicle Dynamics." In: *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer, Boston, MA. http://link.springer.com/content/pdf/10.1007%2F978-1-4614-1433-9_4.pdf