12-29-2021

Human Driver Simulation Model

Chen Haojun
Washington University in St. Louis

Human Driver Simulation Model

Abstract— As part of the vehicle environmental certification process, the Environmental Protection Agency (EPA) requires automobile manufacturers to run a series of “drive cycle” tests to evaluate the efficiency of a vehicle (miles/gallon for internal combustion engine vehicles or Wh/mile for electric vehicles). For these tests, the dynamometer must be controlled by a human driver. The goal of this project is to create a simulation model of a human driver performing an automobile speed control task using MATLAB and Simulink. This model mimics human control tendencies and error as closely as possible by tuning parameter values to best-fit experimental data. Outputs from the model are brake pedal and accelerator pedal positions. Inputs to the model are the current desired vehicle speed, the current actual vehicle speed, and the desired vehicle speed a short time in the future. This preview of the desired speed is available to human drivers in the dynamometer test, and including preview as a control pathway in the simulation model was critical to producing reasonable results.

Keywords – Simulation, Manual control, Parameter optimization, Grey-box model

INTRODUCTION

Nowadays, the number of vehicles is increasing year by year. According to Hedges Company, the number of registered vehicles in 2021 was 289.5 million, which is 0.898% more than 2020's 286.9 million [1]. With the number of vehicles increasing, the greenhouse effect problem undoubtfully put on the table. From the engineering perspective, how engineer can increase the efficiency of vehicle becomes more important. Although automation technology becomes more mature than before and has been applied in some of industry product, many systems still require a human to perform real-time control [2]. In the recent years, automatic driving system is thriving and has the sign that this will replace the driver in the next several years. However, transferring into the fully robot control driving is still far away from us. Human control systems are still common and human driver-based vehicles are still playing the important role in human society. Manual tracking control is still ubiquitous.

Human control behavior for an unpredictable signal can be modeled from control theory and the field of "manual control." Manual controller models can be either structural or algorithmic. A structural model uses the explicit equation and parameters to build the human control model and human's resulting input-output response [2]. An algorithmic model uses implicit ways to compute the model. By developing the manual controller model which simulates the human behavior can be helpful for the vehicle industry companies to better understand how human behavior will affect the performance of vehicle so that the companies can develop their vehicles based on the result of the "Human" driver model.

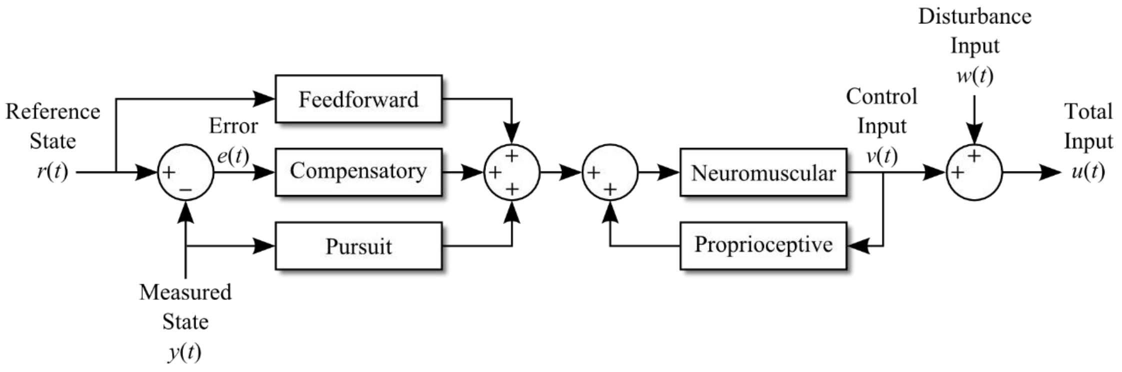


Fig. 1. Manual controller signals and control pathways [2].

METHODS

The overall approach was to construct a control system model with free parameter values, and then use MATLAB's *fmincon* function to tune these parameter values to minimize the difference between the control system output and experimental data.

As a proof-of-concept for the optimization scheme, a simple lead-lag controller from the manual control literature was created, and controlled a trivial plant model.

After receiving pedal data, a more detailed model was created to capture the human's pedal responses to desired and actual speed of the dynamometer. There was no simulated automobile plant in the final Simulink model.

RESULTS

Simple testing model

For the initial proof-of-concept model (Fig. 2), the driver controls the vehicle to match a desired speed. After building the model, optimization process in the MATLAB script tries different combinations of parameter values (bounded by specified lower and upper values) for parameters *K* (overall gain), *T1* (lead time constant), and *T2* (lag time constant). The human reaction time delay *tau* was fixed at 0.2 sec.

MATLAB's *fmincon* function found optimal values of $K = 20$, $T1 = 0.3305\ sec$, and $T2 = 1.1024\ sec$. Simulation results are plotted in Fig. 3.

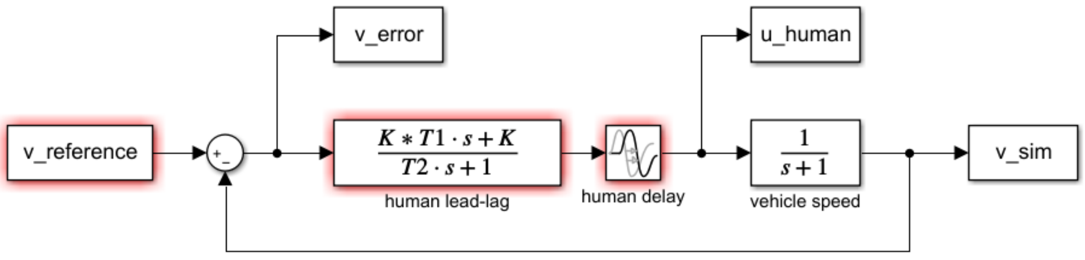


Fig. 2. Simple manual tracking task with first-order lag model of vehicle speed.

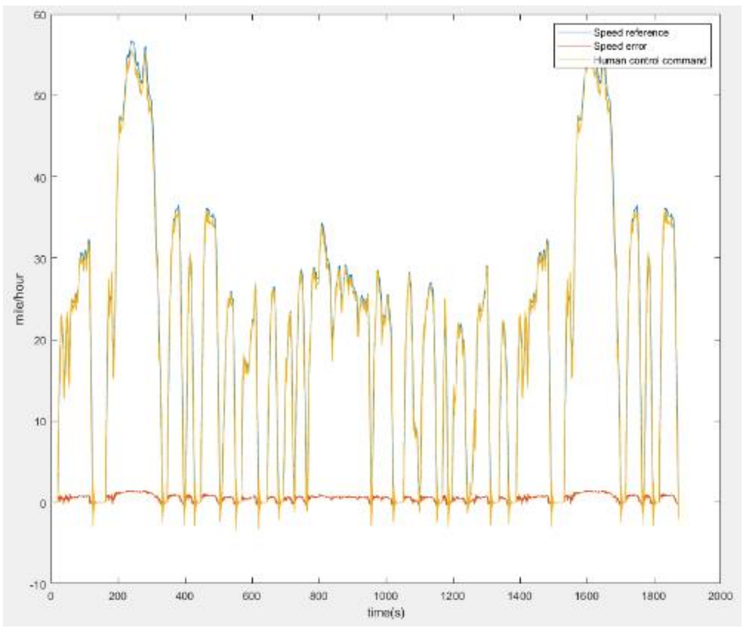


Fig. 3. Reference speed, speed error, and human control command.

Human Simulink block

With the *fmincon* scheme worked out, the human driver Simulink block with desired inputs and outputs was constructed. A high-level view of the block's inputs and outputs is shown in Fig. 4, and the complete subsystem is shown in Fig. 5.

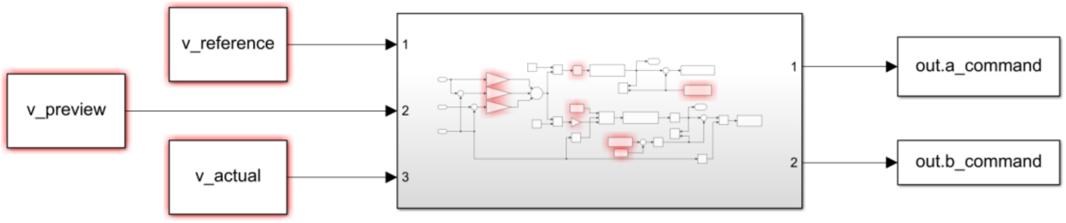


Fig. 4. Top-level view of system inputs and outputs.

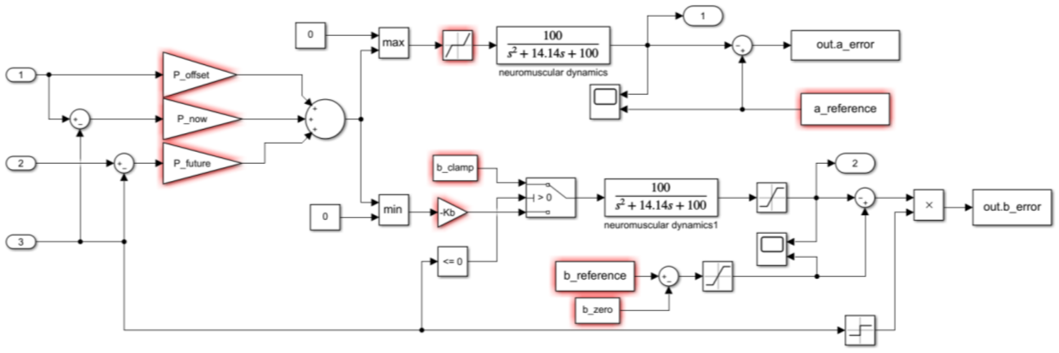


Fig. 5. Diagram of complete internal model. There are no hidden subsystems.

A set of signals and parameters must be explained. Some signals are imported from the MATLAB workspace, and others are exported to the workspace. See tables below. Other features of the model will be described in the Discussion section.

Table 1. Time-varying signals to and from the workspace.

Signal	**Direction**	**Description**
v_reference	From workspace	Desired speed of the dynamometer
v_preview	From workspace	Desired speed of the dynamometer at a specified time (*dt_preview*) in the future
v_actual	From workspace	Actual speed of the dynamometer, recorded during an experimental test (a “drive cycle”)
a_reference	From workspace	Accelerator pedal position, recorded from test
a_command	To workspace	Accelerator pedal position, generated by simulated human driver control system
a_error	To workspace	Difference between recorded and simulated accelerator pedal position
b_reference	From workspace	Brake pedal position, recorded from test
b_command	To workspace	Brake pedal position, generated by simulated human driver control system
a_error	To workspace	Difference between recorded and simulated brake pedal position

Table 2. Optimized parameters and their optimal values determined by *fmincon*.

Parameter	**Value**	**Description**
P_now	0.3986	Proportional gain on the current velocity error
P_future	2.2242	Proportional gain on the difference between the current velocity and the velocity *dt_preview* sec in the future
P_offset	0.3573	Feedforward gain on desired velocity to allow maintenance of steady-state speed without integral control action
Kb	0.8567	Scaling factor to account for difference in magnitude of control action between accelerator and brake pedals
dt_preview	2.9616	Amount of time (in sec) to look ahead in *v_reference*
a_threshold	3.2442	Do not press accelerator pedal until command exceeds this value, to emulate a human driver “coasting” behavior

Optimization

The opimization uses the *fmincon* function. First, initial values are chosen and an initial state vector is formed. Then, the linear inequality contraints are used to specify boundaries for the parameter values.

```
% Initial state vector
x0 = [P_now, P_future, P_offset, Kb, dt_preview, a_threshold];

% Linear equality constraints (constraint on linear combo of states)
Aeq = [];
Beq = [];

% Linear inequality constraints (set boundaries of parameter values)
% Ax <= B
A = [-1, 0, 0, 0, 0, 0; ...   % P_now lower limit
      1, 0, 0, 0, 0, 0; ...   % P_now upper limit
      0,-1, 0, 0, 0, 0; ...   % P_future lower limit
      0, 1, 0, 0, 0, 0; ...   % P_future upper limit
      0, 0,-1, 0, 0, 0; ...   % P_offset lower limit
      0, 0, 1, 0, 0, 0; ...   % P_offset upper limit
      0, 0, 0,-1, 0, 0; ...   % Kb lower limit
      0, 0, 0, 1, 0, 0; ...   % Kb upper limit
      0, 0, 0, 0,-1, 0; ...   % dt_preview lower limit
      0, 0, 0, 0, 1, 0; ...   % dt_preview upper limit
      0, 0, 0, 0, 0,-1; ...   % a_threshold lower limit
      0, 0, 0, 0, 0, 1]; ...  % a_threshold upper limit

% "Small" and "large" parameter values for optimization boundaries
d = 0.001;
D = 5;

B = [-d; D; ...   % P_now lower and upper limits
     -d; D; ...   % P_future lower and upper limits
     -d; D; ...   % P_offset lower and upper limits
     -d; D; ...   % Kb lower and upper limits
     -d; D; ...   % dt_preview lower and upper limits
     -d; D];      % a_threshold lower and upper limits
```

Fig 6. Initial state vector and boundary condition for optimization

The *fmincon* function uses *ComputeRMSE* (shown in Fig. 7 below) as the cost function, and tries to minimize the output of this function by trying hundreds of parameter value combinations. The cost is calculated by adding the root mean square (RMS) error for accelerator position and the RMS error for brake position.

```
function [RMSerror] = ComputeRMSE(x)
    global P_now P_future P_offset Kb dt_preview i_num;
    global a_error b_error a_command b_command b_clamp b_zero a_threshold
    global v_reference v_actual v_preview a_reference b_reference dt1 dt2;

    % Increment # of function evals
    i_num = i_num + 1;

    % Unpack states
    P_now = x(1);
    P_future = x(2);
    P_offset = x(3);
    Kb = x(4);
    dt_preview = x(5);
    a_threshold = x(6);

    % Make previewed reference speed
    n_prevframes = ceil(dt_preview/dt1);
    v_preview = v_reference;
    v_preview(1:end-n_prevframes,2) = v_reference(n_prevframes:end-1,2);

    % Run simulation
    res = sim('Version14_2019');

    % Calculate error
    sim_accel_diff = res.a_error.Data;
    sim_brake_diff = res.b_error.Data;
    RMSerror = sqrt(mean(sim_accel_diff.^2)) ...
              + sqrt(mean(sim_brake_diff.^2));

    fprintf('\t i_num \t RMSE \n');
    disp([i_num, RMSerror]);
    fprintf('\tP_now\tP_future\tP_offset\tKb\tdt_preview\ta_threshold\n');
    disp(x);
end
```

Fig. 7. Function for proportional gain and error

Pedal data results

The final simulated human controller model shows fairly good agreement with experimental data for the accelerator pedal, and somewhat less accurate tracking of the brake pedal position. Results are shown in Fig. 8 and Fig. 9 below.

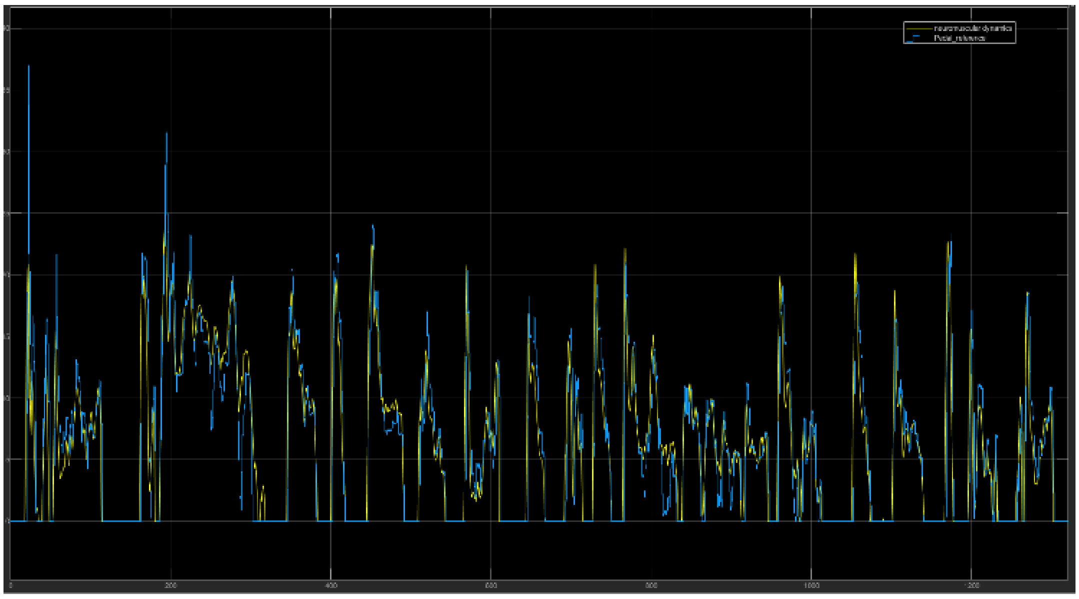


Fig. 8. Accelerator pedal position for experimental (yellow) and simulated (blue) human driver. The horizontal axis shows time in seconds, and the vertical axis shows pedal position in % of max.

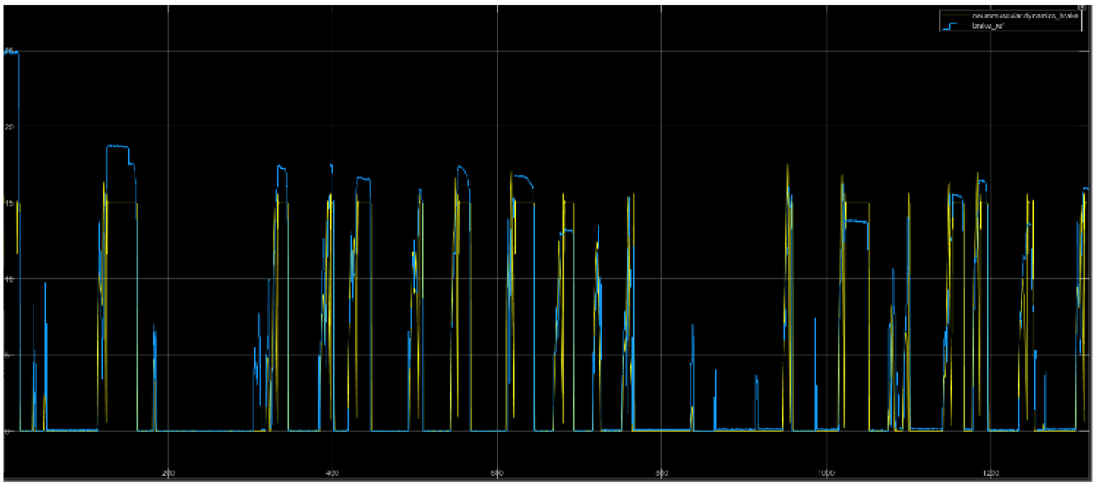
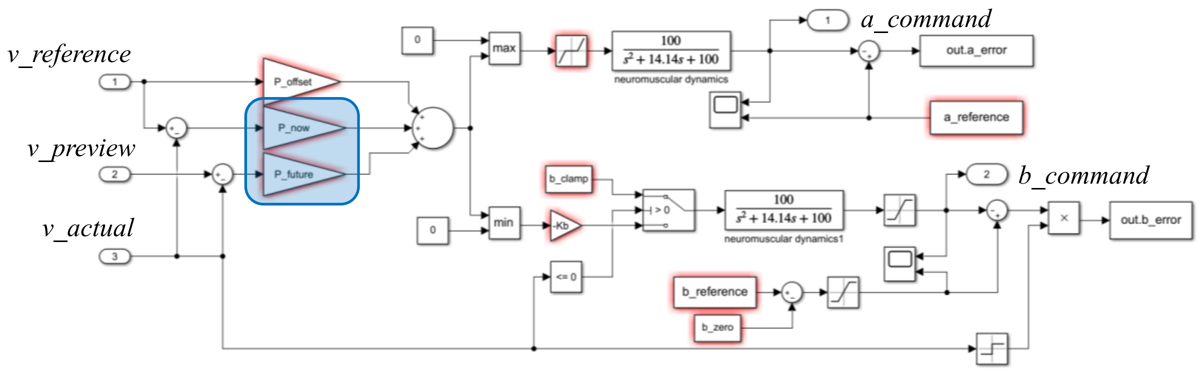


Fig. 9. Brake pedal position for experimental (yellow) and simulated (blue) human driver. The horizontal axis shows time in seconds, and the vertical axis shows pedal position in % of max.

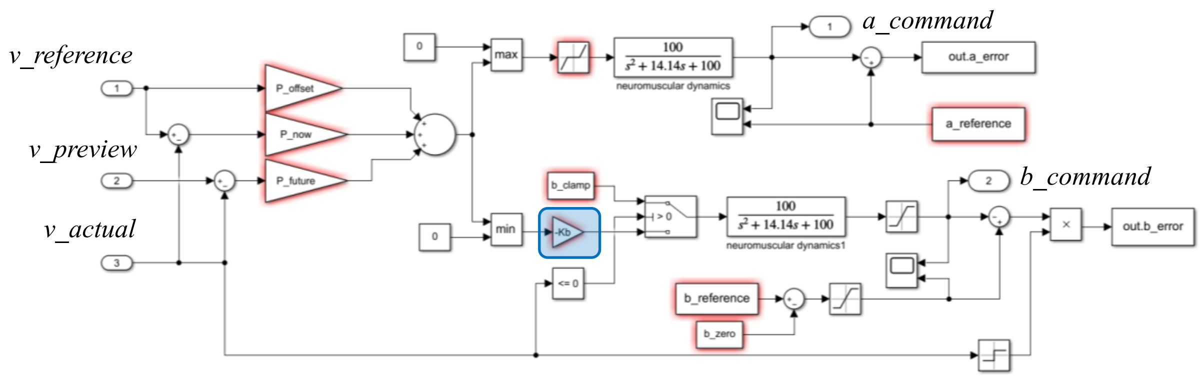
DISCUSSION

Some control situations have been studied extensively in the literature, and have well-defined and verified models. There are several factors that make the tracking situation in this project ***significantly different*** from the standard lab tracking experiments from the literature, and did not allow easy application of a model from the literature.



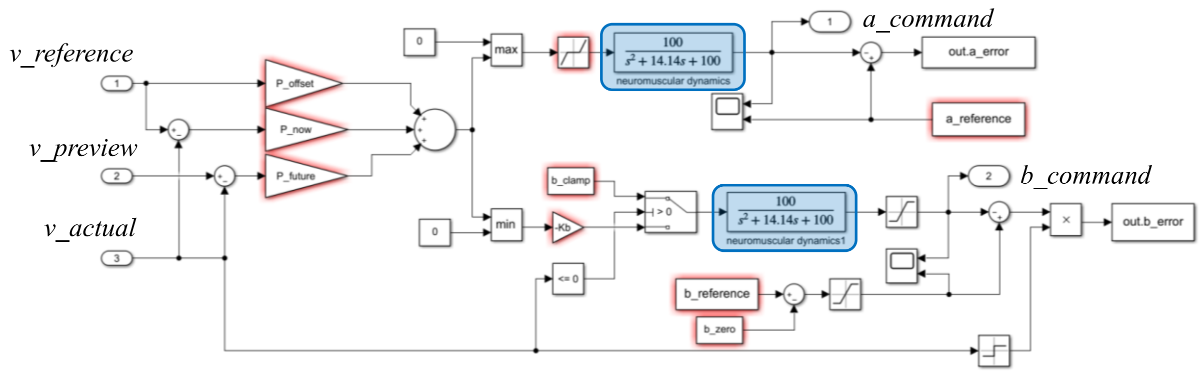
1. Reaction time delay and preview

It is well-established that humans have an unavoidable reaction time delay. This is critical in compensatory and pursuit tasks with an unpredictable reference signal and no preview of its future trajectory. In this tracking task, the human driver is able to see the reference signal up to a few seconds in advance. For this reason, the human's reaction time is not a factor. It was further found that including preview as a control pathway was critical to success of the simulation model. In other words, the human was presented with previewed information, and it appears that *they <u>did</u> use this information*. (Just because information is available does not always mean that it is used.)



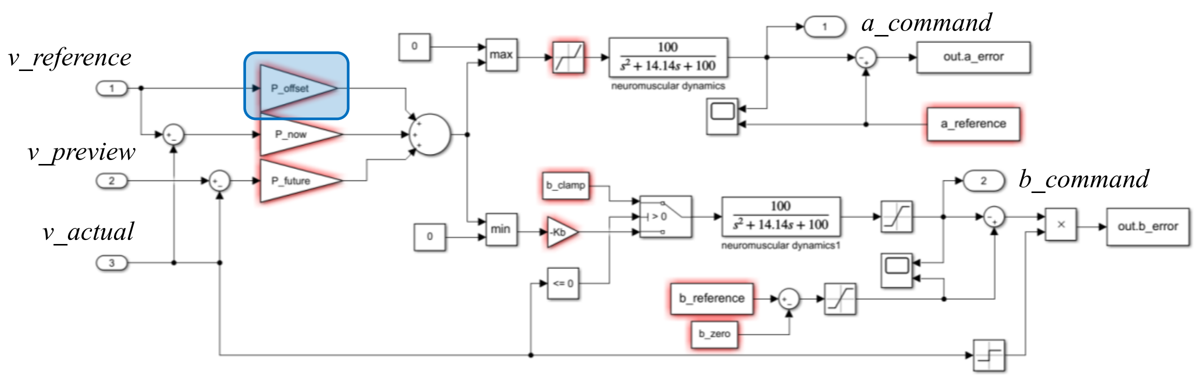
2. Different scale of accelerator and brake pedal commands

An initial command magnitude is formed from summing the three gains on the left. Then the command is compared to zero, and split into two branches: one for positive commands (requiring accelerator input) and one for negative commands (requiring brake input). The accelerator and brake pedal inputs do not necessarily cause the same magnitude of effect on the plant (the vehicle). Therefore, a gain was applied to the brake side of the command to capture this.



3. Filter for "neuromuscular dynamics"

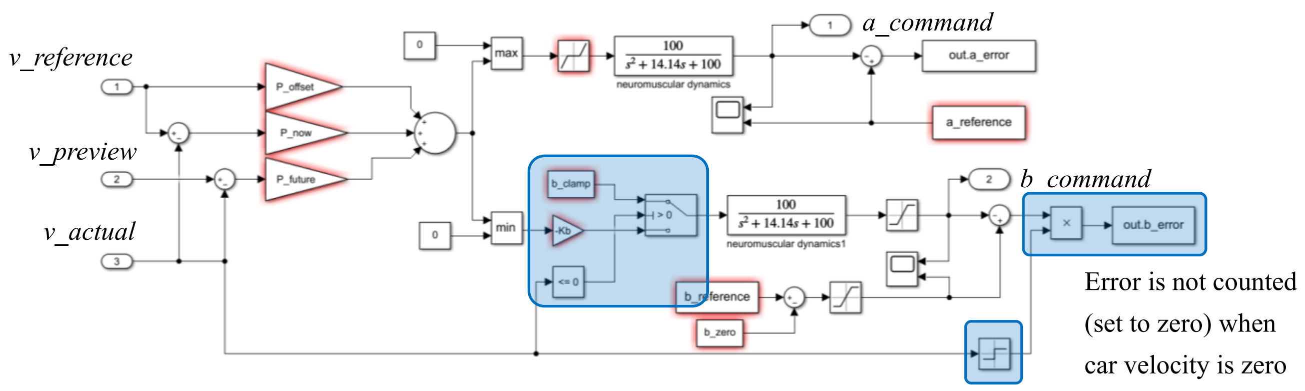
Human limbs cannot move instantaneously because they have mass and are actuated by muscles that take time to activate, contract, and transmit motion. As a way to capture this effect, some manual control researchers (especially Ronald Hess) add a continuous filter to the command before it reaches the physical controls. The filter is second-order with a natural frequency of $\omega_n = 10\ rad/s$ and $\zeta = 0.707$.



4. Feedforward gain *P_offset* to maintain steady-state speed

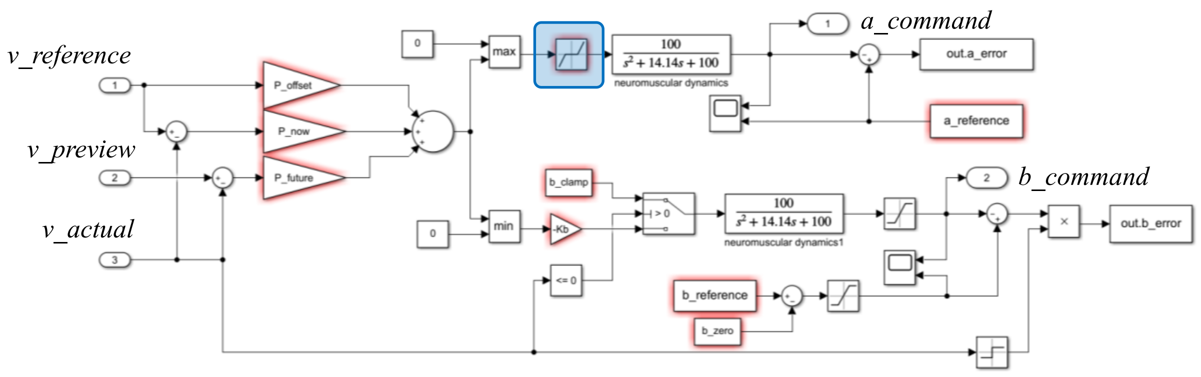
When trying to maintain a steady-state speed, we human drivers know that a non-zero accelerator pedal command is necessary. In other words, we will need to hold the accelerator down a certain amount to maintain the desired speed. A traditional feedback controller would use integral control action to eliminate steady-state error.

In this application, it is known that speed will always be positive, the offset in accelerator command should be zero when speed is zero, and should increase as the speed increases. Therefore, a simple linear feedforward gain on the desired vehicle speed was used to aid in matching the steady-state speed.



5. Braking when vehicle is stopped

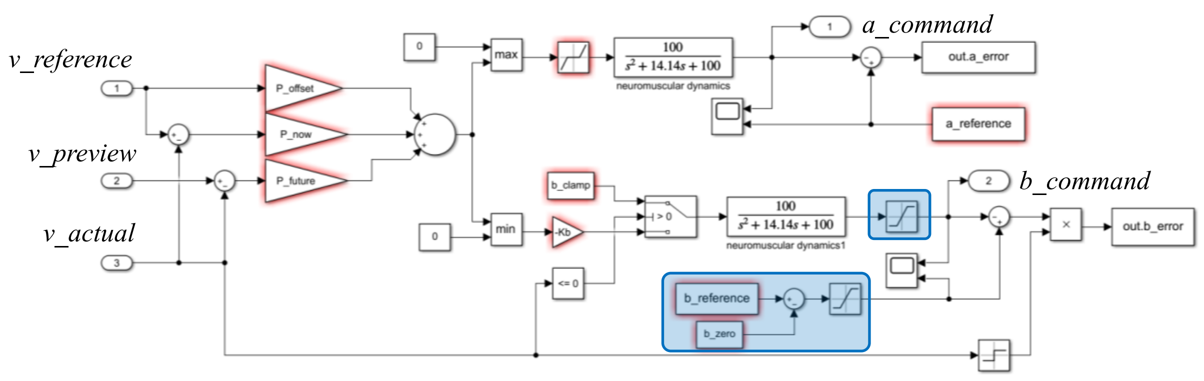
When the vehicle stops, there must be a non-zero brake command to keep it stopped. The left blue box enables this behavior. Another feature of the stopped vehicle is that *tracking error* of the brake pedal should be set to zero during that time! The driver holds the brake pedal down to keep the car stopped, but the exact brake value does not matter. Counting this error while the vehicle is stopped produces poor optimization results.



6. Accelerator dead zone for coasting

Real human drivers sometime exhibit “coasting” behavior, where they leave both pedals unpressed if the difference between desired and actual speed is not too large. This behavior is captured by a *dead zone* block in Simulink applied to the accelerator branch of the command. The dead zone block generates zero output within a specified region by setting up the upper limit and lower limit [3]:

- If the input is within the dead zone (greater than the lower limit and less than the upper limit), the output is zero.
- If the input is greater than or equal to the upper limit, the output is the input minus the upper limit.
- If the input is less than or equal to the lower limit, the output is the input minus the lower limit.



7. Offset and saturation of brake data

There was some noise in the recorded experimental data. When the brake pedal was not pressed, there remained a small nonzero value. This value of *b_zero* was subtracted off of all brake data. Then any negative values in the modified data were set to zero using the saturation block. The simulated brake command occasionally dipped slightly below zero, so a saturation block was also applied to the simulated brake command. It was assumed that a negative brake command could be problematic if this controller were applied to a real dynamometer test.

The Saturation block imposes upper and lower bounds on a signal. When the input signal is within the range specified by the *lower limit* and *upper limit* parameters, the input signal passes through unchanged. When the input signal is outside these bounds, the signal is clipped to the upper or lower bound [4].

CONCLUSION

The human driver model is intended to help an automotive company know how settings and control features can affect the efficiency of a vehicle. In this model, the human's control behavior can be represented by a simulation model. The difference between the desired velocity, future desired velocity, and actual velocity can be converted into pedal commands using common Simulink blocks and a minimum of hidden/nested subsystems. It was found that incorporating the previewed desired velocity (available to the human driver) was necessary to make this model work well.

REFERENCES

[1] Gilbert, N. (2021, March 25). The number of cars in the US in 2021/2022: Market share, distribution, and Trends. Financesonline.com. Retrieved December 20, 2021, from https://financesonline.com/number-of-cars-in-the-us/

[2] James Jackson Potter. (n.d.). A modelica library for manual tracking. jjpotterkowski.github.io. Retrieved December 20, 2021, from http://jjpotterkowski.github.io/

[3] Simulink Reference. Dead zone (simulink reference). (n.d.). Retrieved December 20, 2021, from http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/simulink/slref/deadzone.html

[4] Simulink reference. Saturation . (n.d.). Retrieved December 20, 2021, from http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/simulink/slref/saturation.html