UC DAVIS
UNIVERSITY OF CALIFORNIA

Department of Mechanical and
Aerospace Engineering

Autonomous Electric Vehicle

Recently, you were contracted to create a bond graph model of the drive train of a prototype electric vehicle. The company has now asked you to not only simulate the vehicle, but also to implement some vehicle autonomy. The vehicle schematic can be found in Figure 1, and the bond graph model can be found in Figure 2.

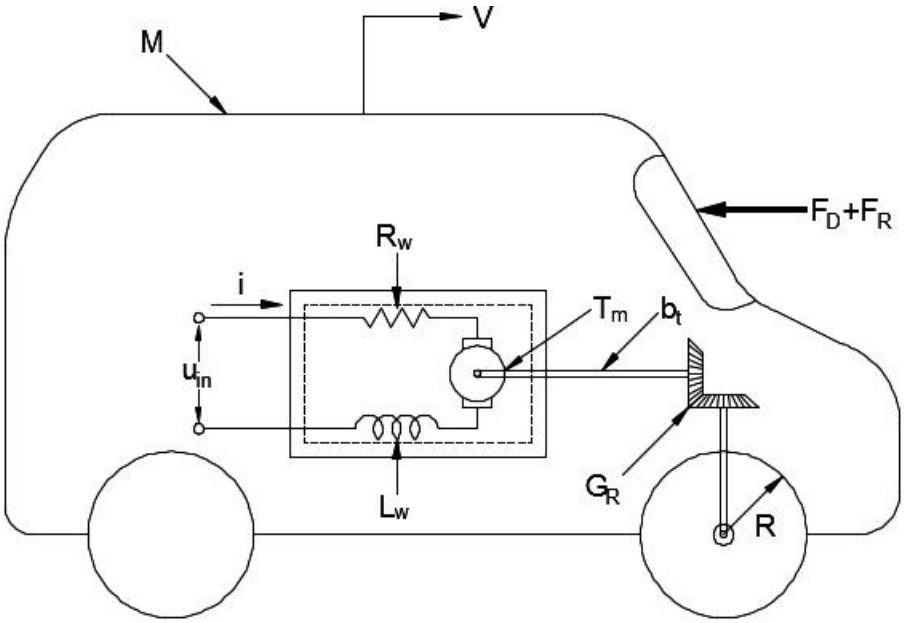

Figure 1: Vehicle Schematic. Note that motor inertia and drive shaft compliance are neglected.

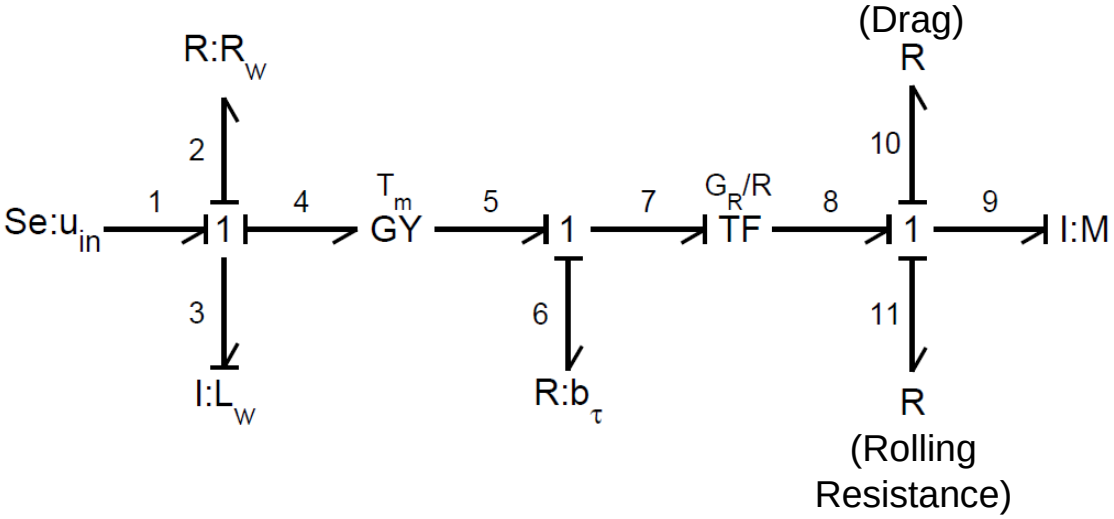

Figure 2: Vehicle Bond Graph Model.

The prototype uses a fixed mechanical transmission. The battery pack is treated as an ideal, controllable voltage source (u_{in}). The vehicle uses the parameters listed below:

Parameter	Description	Value
R_w	Armature Winding Resistance	0.3 Ω
L_w	Armature Winding Inductance	0.015 Henry
T_m	Transduction Coefficient	1.718 Weber
M	Vehicle Mass	2200 kg
b_τ	Drive Shaft Friction	0.05 Nms/rad
R	Wheel Radius	0.2 m
G_R	Gear Ratio	5: 1
C_R	Rolling Resistance Coefficient	0.006
g	Acceleration due to gravity	9.81 $\mathrm{m/s^2}$
C_D	Drag Coefficient	0.32
ρ	Air Density	1.21 $\mathrm{kg/m^3}$
A_f	Vehicle Frontal Area	2.05 $\mathrm{m^2}$

The vehicle experiences two nonlinear resistances: aerodynamic drag and rolling resistance. Their constitutive laws are as follows (v is the flow corresponding to either resistive element):

$$F_{Drag} = \frac{1}{2}\rho A_f C_D v|v|, \qquad F_{Roll} = MgC_R \mathrm{sgn}(v)$$

Using the notation on the bond graph shown in Figure 2, that means we have

$$e_{10} = \frac{1}{2}\rho A_f C_D f_{10}|f_{10}|, \qquad e_{11} = MgC_R \mathrm{sgn}(f_{11})$$

All other elements are linear.

Overview

This lab will consist of three parts. In Part 1, you will the derive equations of motion and implement a model of an electric vehicle in Matlab. In Part 2, you will design a velocity controller for the vehicle and assess the controller performance. In Part 3, you will simulate the vehicle for a realistic drive profile.

Part 1: Modeling & Implementation

Your first task is to take the bond graph model of the prototype vehicle (Figure 2) and develop the state equations of the system. You will have two state variables: p_L for the motor flux linkage and p_M for the vehicle momentum. u_{in} is the voltage input to the system.

Once you've implemented your state equations, check that your model is working by simulating a step input to u_{in} with a magnitude of 100 volts (i.e. choose a start time T_1 and set $u_{in} = 100$ for $t \geq T_1$). Simulate for 4 seconds after the start of the step input with a time step of 0.01 seconds and initial conditions of zero. Your vehicle should reach a steady speed of about 2.3 m/s.

Hint: $\text{sgn}(\cdot)$ is the signum function, which returns 1 if the input is positive, -1 if the input is negative, and 0 if the input is zero. Although Matlab has a command for the $\text{sgn}(\cdot)$ function, the ODE solver will not be able to simulate your system quickly when using that command. Instead, try approximating it with the following relationship: $\text{sgn}(v) \cong v/(|v| + n)$, where n is some small number. Choosing a smaller n will more closely approximate $\text{sgn}(v)$, but require more computational effort when you simulate the system.

Part 2: Controller Design

Your next task is to implement a velocity controller using proportional-integral (PI) control. On-board sensors and computers will generate a desired velocity (v_{ref}) for the vehicle. Your controller will generate an input voltage u_{in} based on the error between v_{ref} and the actual vehicle velocity (p_M/M) as well as the integral of that error:

$$u_{in} = K_p\left(v_{ref} - \frac{p_M}{M}\right) + K_i \int_0^t \left(v_{ref} - \frac{p_M}{M}\right) dt$$

K_p and K_i are the proportional and integral gains, respectively. The integral of velocity is simply distance, so we can rewrite the control law as follows:

$$u_{in} = K_p\left(v_{ref} - \frac{p_M}{M}\right) + K_i\left(d_{ref} - d\right)$$

In order to implement this controller, you will need to obtain d_{ref} and d, which can be achieved by numerically integrating $\dot{d}_{ref}$ and $\dot{d}$ with the rest of your state variables (i.e., expand your state space to include $\dot{d}_{ref} = v_{ref}$ and $\dot{d}_{actual} = p_M/M$ in your list of integrated variables). Remember to include initial conditions of zero for these states. Then, define u_{in} per the equation above (you will be told how to define v_{ref} shortly).

Your first job is to find values for K_p and K_i so that the vehicle responds to a command velocity quickly, accurately, and without much overshoot. To do this, you will use a trial-and-error approach while simulating a unit step input. First, in your function file, choose a start time T_1 and then set $v_{ref} = 1$ for $t \geq T_1$. Set your simulation time span to have a finish time of 2 seconds after the start of the step function, and use a time step of 0.01 seconds.

Choose some small values for K_p and K_i and simulate the system. If the response is too slow or if there is large steady-state error, try increasing the gains (or if it's too fast and has too much overshoot, lower them). Simulate again and repeat until you find a pair of gains that meet the following specifications:

1) rise time less than 0.5s,
2) settling time less than 2.0s, and
3) overshoot of less than 10%.

As illustrated in Figure 3, the rise time is the time to takes to go from 10% to 90% of the steady state value. Settling time is the time until output response stays within 2% error of the input (note that this means the vehicle velocity must go to 1 m/s). Overshoot percent is the peak value of the

response divided by the steady-state value, minus one (or zero, if there is no peak above the input command). You can use Matlab's data cursor ( ) to measure these values directly from your plots.

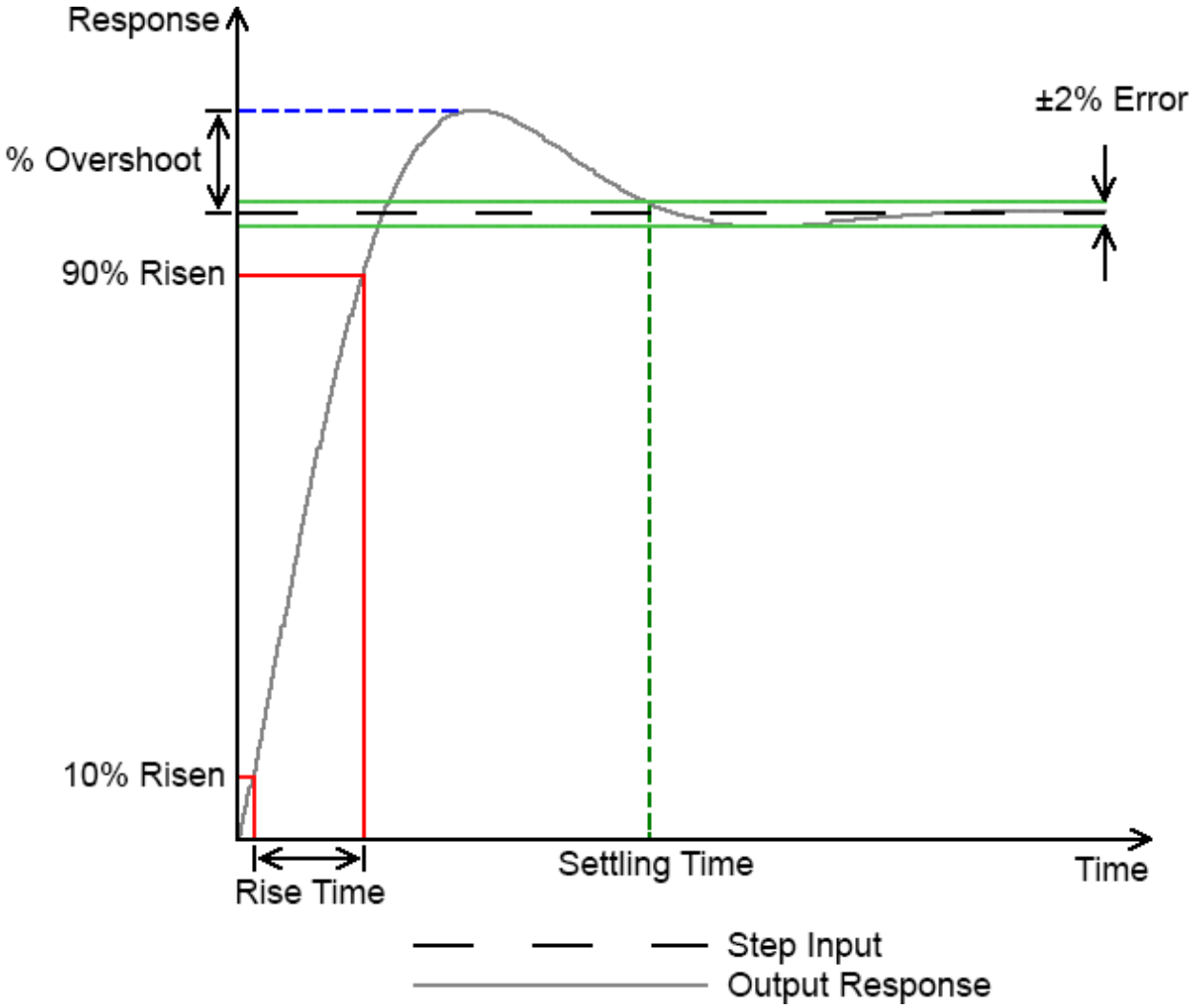

Figure 3: Illustration of rise time, settling time, and overshoot.

Hint: The proportional gain K_p primarily affects the speed of the response, while the integral gain K_i primarily affects steady-state error.

Part 3: Testing

The final step is to simulate how the vehicle performs over a realistic driving scenario and assess its performance. Use the K_p and K_i that you designed in Part 2 for this step.

The California Air Resource Board has gathered experimental data from medium- and light-duty vehicles and used it to create a velocity profile for testing vehicle emissions and energy consumption. You will use this velocity profile, known as the "LA92 Dynamometer Driving Schedule", to test your vehicle.

You are provided with a Matlab function, LA92Oracle.m, that contains the LA92 velocity profile. The LA92Oracle function accepts a time input and outputs the reference velocity at that time. You can use it in your function m-file to set v_{ref} as follows:

```
vref = LA92Oracle(t);
```

Simulate the vehicle model for the first 300 seconds of the LA92 velocity profile: Set the simulate timespan to start at 0, end at 300, and have a time step of 0.01 seconds. Note: this simulation might take some time to run.

You can load the LA92 velocity profile after you run your simulation by using the following code in your master file:

```
vref=zeros(length(t),1);
for i=1:length(t)
     vref(i) = LA92Oracle(t(i));
end
```

Generate two plots: One with the actual vehicle velocity and another with both the actual velocity and the LA92 reference velocity. If the controller you designed is well, then the it should very closely follow the reference velocity. At first glance, the two might be indistinguishable. You will have to zoom in on regions of the plot to identify the differences between the actual velocity and the reference velocity.

Lab Report

For your lab report, provide:

Part 1

- State equations for the vehicle model in symbolic form.
- A plot of the vehicle velocity as a response to a 100 volt input to u_{in}.

Part 2

- A plot of the velocity response to a 1 m/s input to v_{ref} for the $K_p$and K_i you designed.
- Your controller gains K_p and K_i and the rise time, settle time, and overshoot with respect to a 1 m/s input to v_{ref}.
- Describe how you designed your controller and how the gains K_p and K_i and how they affect the response of the vehicle:
    - Describe how you found the gains K_p and K_i.
    - How K_p and K_i each affect the speed and accuracy of the response.
    - What sort of response do you get by setting either of the gains to zero?

Part 3

- A plot of the simulated vehicle velocity on the first 300 seconds of the LA92 drive cycle.

- A plot of the simulated vehicle velocity *and* the reference velocity, this time showing only 32s to 54s on the x-axis and 4m/s to 8 m/s on the y-axis. Use the command `axis([32,54,4,8])` to set the axes of the plot.
- Describe how the vehicle/controller performed on the LA92 test:
  - How much does the actual velocity lag behind the command velocity?
  - Is there any significant overshoot/undershoot or oscillations in the response?
  - Does the performance of the controller change with vehicle speed?

  Note: You can use the `xlim`, `ylim`, or `axis` commands, or the zoom tool (), to get a closer look at how the actual velocity and reference velocity compare.
- Discuss vehicle energy efficiency:
  - What is the average accelerating efficiency of the vehicle over the simulated LA92 cycle, in terms of power in vs. power out? The vehicle velocity on the LA92 cycle is never negative, so that means when the output power is positive, then the vehicle is accelerating. So, find the average input power while the *output power* is greater than zero and compare it to the average output power while the output power is greater than zero.

    Hint: Power equals effort times flow. Although you have the input and output flows available from your state variables ($i_{in} = p_L/L_w$, $v = p_M/M$), you may need to define extra output variables to obtain the input effort (u_{in}) and output effort ($\dot{p}_M$).

    Second hint: If you define the input power in Matlab as `Pin` and the output power as `Pout`, the input and output power while accelerating can be obtained be obtained by writing `Pin_acc = Pin(Pout>0)` and `Pout_acc = Pout(Pout>0)`. Then, compare the mean of `Pin_acc` to the mean of `Pout_acc` to get the average efficiency of the vehicle while accelerating.
  - Find the ratio of meters traveled per Joule of energy consumed over the complete LA92 cycle (meters per Joule could be converted to a more typical measure of fuel economy like equivalent miles per gallon of gasoline, but you are not required perform that conversion).

    Hint: Distance traveled is the integral of the vehicle velocity with respect to time – you should already have this variable in your state vector, from when you implemented PI control. Energy consumed is the integral of the input power with respect to time – you will need to expand your state space to include the input power to obtain this variable. You want to compare the final values for distance and energy for this question.

    Second hint: One Joule is a very small amount of energy when it comes to something the size of a car. Expect to get a very small number for this calculation.
- A printout of your code.

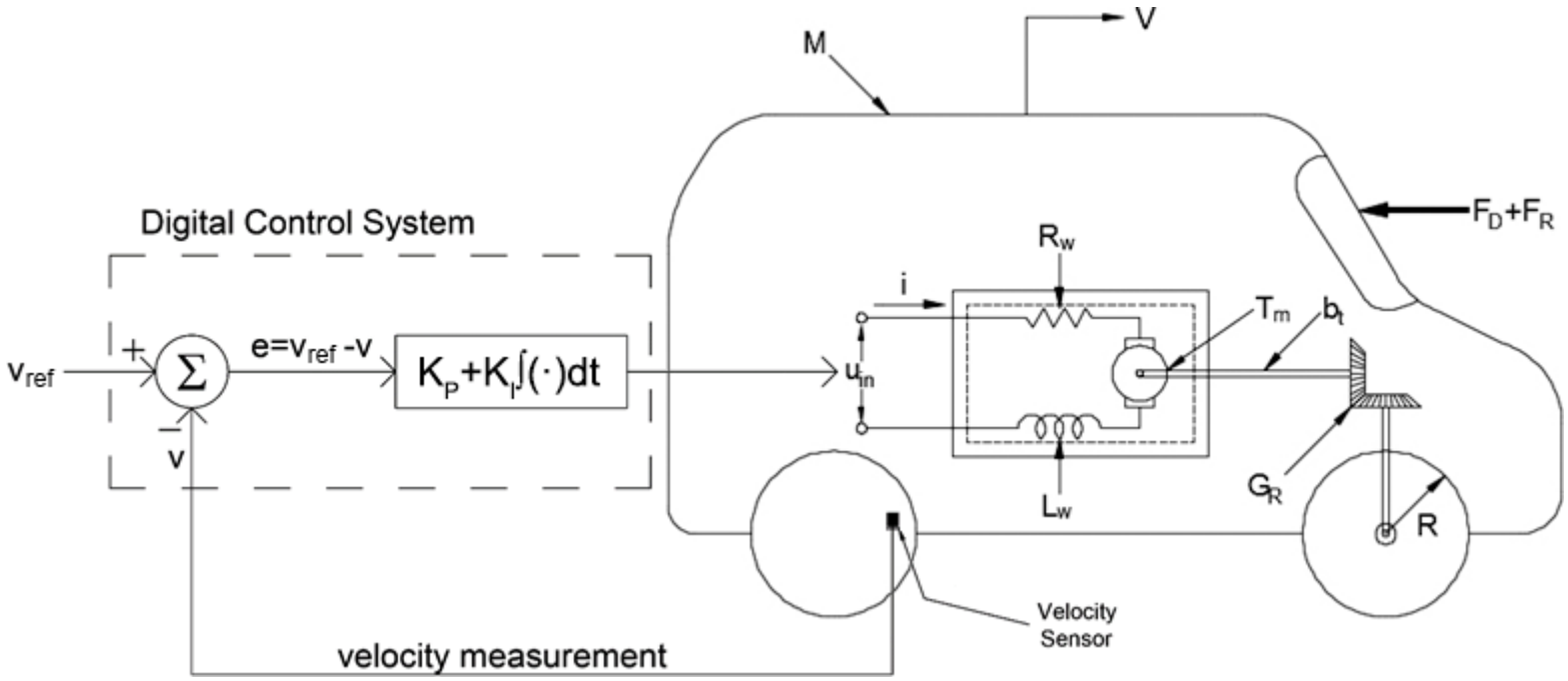

Figure 4: Schematic with Control System.