Lab 7: Controls

In the previous labs, we have built SIXT33N's motor control circuitry and developed a linear model for the velocity of each wheel. We are one step away from our goal: to have SIXT33N drive in a straight line! As you probably noticed, simply applying the same PWM input to both wheels as we did during data collection in the last lab is not sufficient; SIXT33N ends up driving in a circle if we do this because of the differences between the two wheels. In this lab, we will see how to use the model we developed in System ID to control SIXT33N's trajectory to be a straight line.

Part 1: Open-Loop Control

Introduction

Before we implement our closed-loop controller, we need to understand how open-loop control works. An open-loop controller is one in which the input is predetermined using your system model ($\theta_{L,R}$, $\beta_{L,R}$), and not adjusted at all during operation. To design an open-loop controller for your car, you would set the PWM duty-cycle value of your left and right wheels (your inputs $u_L[i]$ and $u_R[i]$) such that the velocity of both wheels is your target wheel velocity ($v_L = v_R = v^*$). You can calculate these inputs from the target velocity v^*, $\theta_{L,R}$, and $\beta_{L,R}$ values from the previous lab (System ID). As a reminder, here are the equations for our model that we defined in the last lab:

$$v_L[i] = d_L[i+1] - d_L[i] = \theta_L u_L[i] - \beta_L$$
$$v_R[i] = d_R[i+1] - d_R[i] = \theta_R u_R[i] - \beta_R, \tag{1}$$

where $d_{L,R}[i]$ represents the distance moved by left or right wheel at time step i. To implement this open-loop controller, you will solve for $u_{L,R}[i]$ as a function of your v^*, $\theta_{L,R}$, and $\beta_{L,R}$ values (all 3 of which are constants) and hard-code this input value, as we won't change it at all as your car runs in open-loop control.

This works well if your model is perfect ($\theta_{L,R}$ and $\beta_{L,R}$ exactly match the real system), and your car is never bumped or otherwise disturbed. In other words, with no mismatch between the idealized behavior we model in System ID and no noise or disturbances from the environment, your car will drive straight. Model mismatch inevitably arises because every model is ultimately an approximation (recall how you used least-squares to derive your model coefficients $\theta_{L,R}$ and $\beta_{L,R}$: were your $v[i]$ vs. $u[i]$ plots perfectly linear?). Since we are assuming ideal characteristics, we will not adjust our input as we go to account for environmental factors or model mismatch (i.e., we have no feedback).

Jolt Calculation

As you (should have!) observed in the System ID lab, the individual motors respond differently to changes in the PWM duty cycles (i.e. they have different sensitivities to the PWM input, $\theta_{L,R}$), as well as different velocity offsets ($\beta_{L,R}$). By extension, this means that from rest, the two motors require different PWM duty cycle inputs in order to overcome static friction and start moving. Recall that PWM is used to digitally change the average voltage delivered to a load by varying the duty cycle (the proportion of a given cycle period for which the signal is turned HIGH). If the cycle period is small enough, the on-off switching is imperceptible, but the average voltage delivered to the load changes proportionally with the duty cycle. Hence, changing the duty cycle corresponds to changing the DC voltage supplied to the motor. If the motors need different duty cycles, and therefore voltages, to start moving, it is evidence of one or more of the following:

- **Difference in motor parameters.** Your motors might have different armature resistances, which in turn limits the amount of armature current flowing through the motor for a given voltage. If the resistance is higher, the motor will need a higher voltage to force enough current through the internal motor circuit for the motor to turn.

- **Difference in motor efficiencies.** This could be interpreted as a subsection of the previous item, but the distinction between the two is that we will interpret the first as primarily a consequence of differences in electrical characteristics, and this one as a consequence of differences in mechanical characteristics. The friction within the motor may be higher for one motor than the other, or perhaps a gear is slightly misaligned in one motor, etc.

- **Mass imbalance in the car.** If the mass of the car isn't distributed evenly between the wheels (e.g. your breadboard is off-center and puts more weight on one side), the torque required for the wheels to begin turning will differ.

Most of the cases outlined above are expected, and we are able to correct for them by modifying the jolts we apply to start each wheel when the car starts up.

Part 2: Closed-Loop Control

Introduction

In the System ID lab, you linearized the car system around your operating point (the value of v^* you selected). We will now be deriving the equations which govern the behavior of our system. These equations determine the dynamics of the system that we want to control. **We will implement a state-space controller by selecting two dimensionless positive coefficients, $f_L > 0$ and $f_R > 0$, such that the system's eigenvalue λ is placed in a stable position**. (Recall that for discrete-time systems, eigenvalues must be within the unit circle on the complex plane for the system to be stable; i.e. the magnitude of the eigenvalue must be less than 1).

$$|\lambda| < 1 \tag{2}$$

How do we choose the correct input value for our closed-loop controller? First, we need to identify a control variable (and a value for it) which corresponds to our desired system behavior. Once we do that, we need to determine the relationship between the input and this variable.

We will define our **control variable**, δ, to be the **difference in the *distance traveled*** between the left and right wheels at a given timestep.

$$\delta[i] = d_L[i] - d_R[i] \tag{3}$$

Our goal is for the car to **drive straight.** This means the *velocities*, not necessarily the positions, of the wheels need to be the same.

> Sanity check: If we want $v_L[i] - v_R[i]$ to be zero, what condition does this impose on δ?

We introduced $\delta[i] = d_L[i] - d_R[i]$ as the difference in positions between the two wheels. If both wheels of the car are going at the same velocity, then this difference δ should remain constant, i.e. $\delta[i+1] - \delta[i] = 0$, since no wheel will advance by more ticks than the other. Note: this does not mean that δ needs to be 0! We approximate this constant $\delta[i] \stackrel{n \to \infty}{=} \delta_{ss}$ by using the final value of δ your car exhibits at the end of a run.

> Sanity check: What does it mean for the car's trajectory for when δ is a constant 0? What about a constant nonzero value?

Deriving the closed-loop model

We begin with our open-loop equations from last week:

$$v_L[i] = d_L[i+1] - d_L[i] = \theta_L u_L[i] - \beta_L$$
$$v_R[i] = d_R[i+1] - d_R[i] = \theta_R u_R[i] - \beta_R \tag{4}$$

We want to adjust $v_L[i]$ and $v_R[i]$ at each timestep by an amount that's **proportional** to $\delta[i]$. Let's say we want to drive $\delta[i]$ towards zero. If $\delta[i]$ is positive, the left wheel has traveled more distance than the right wheel, so relatively speaking, we can slow down the left wheel and speed up the right wheel to cancel this difference (i.e. drive it zero) in the next few timesteps. Therefore, instead of v^* (our target velocity), our desired velocities are now $v^* - f_L \delta[i]$ and $v^* + f_R \delta[i]$. As v_L and v_R tends to v^*, $\delta[i]$ tends to zero.

$$v_L[i] = d_L[i+1] - d_L[i] = v^* - f_L \delta[i]$$
$$v_R[i] = d_R[i+1] - d_R[i] = v^* + f_R \delta[i] \tag{5}$$

In order to bring $d_L[i+1]$ and $d_R[i+1]$ closer to each other, at timestep i we set $v_L[i]$ and $v_R[i]$ to the desired values $v^* - f_L\delta[i]$ and $v^* + f_R\delta[i]$, respectively. Setting the open-loop equation 4 for $v[i]$ equal to our desired equation 5 for $v[i]$, we solve for the inputs u_L and u_R:

$$u_L[i] = \frac{1}{\theta_L}(v^* - f_L\delta[i] + \beta_L)$$

$$u_R[i] = \frac{1}{\theta_R}(v^* + f_R\delta[i] + \beta_R)$$

(6)

These are the inputs required to equalize $d_L[i+1]$ and $d_R[i+1]$! But how do we know what to set the feedback gain values f_L and f_R to? To figure this out, we will need to find the eigenvalue(s) of the system so we can better understand its behavior in closed-loop feedback.

Let's begin by reviewing some terms related to state-space control.

- System variable: Any variable dependent on/ responding to the input or initial conditions of a system.

- State variables: The *smallest* set of linearly independent system variables such that the values of the members of this set, together with the known functions of these variables that model the system behavior, *completely determine* the value of *all* system variables for all $t \geq t_0$.

- State space: The n-dimensional space whose axes are the state variables (n is the number of state variables)

To study the dynamics, we will first define our state variables. If the goal is to have the car go straight, then our goal should be to maintain equal velocity for the left and right wheels. However, we are measuring the distance travelled by the left and right wheels, so we will implement a controller that drives the difference $\delta[i]$ down to zero. Let's pick δ as our state variable.

> Does the definition of state variables apply to $\delta[i]$? In other words, is the following statement true: Knowing $\delta[i]$ at each time step as well as the initial conditions, we can uniquely determine the value of all other system variables.

Let's now find $\delta[i+1]$ in terms of $\delta[i]$. We can do so by subtracting the two sides of the system of equations from equation 5 from each other.

$$\begin{aligned}
\delta[i+1] &= d_L[i+1] - d_R[i+1] \\
&= v_L[i] + d_L[i] - (v_R[i] + d_R[i]) \\
&= v^* - f_L\delta[i] + d_L[i] - (v^* + f_R\delta[i] + d_R[i]) \\
&= v^* - f_L\delta[i] + d_L[i] - v^* - f_R\delta[i] - d_R[i] \\
&= -f_L\delta[i] - f_R\delta[i] + (d_L[i] - d_R[i]) \\
&= -f_L\delta[i] - f_R\delta[i] + \delta[i] \\
&= \delta[i](1 - f_L - f_R)
\end{aligned}$$

(7)

Key points:

- Since we are trying to drive δ to zero, δ is our state variable.

- Since we have only one state variable, our state space is one-dimensional.

- Since $\delta[i+1] = \delta[i](1 - f_L - f_R)$, the coefficient $1 - f_L - f_R$ fully describes the evolution of our state variable in time. So, $1 - f_L - f_R$ is our system eigenvalue!

Feedback Gain (f-Value) Tuning

We can get different system dynamic behaviors over time for different values of $1 - f_L - f_R$. Below are some example plots of system behavior (assuming no model mismatch) with various f-values. The rapid ramp-up at the beginning of each plot from $t = 0$ to $t \approx 0.1$ is from the jolts, which are set to be unequal so that the controller has something to correct in these ideal simulations. Note that in the following plots, the distance travelled by one of the wheels can decrease, but since our car cannot go backwards, this is not possible with our cars. The purpose of the plots is simply to demonstrate the system's behavior for various values of the system eigenvalue.

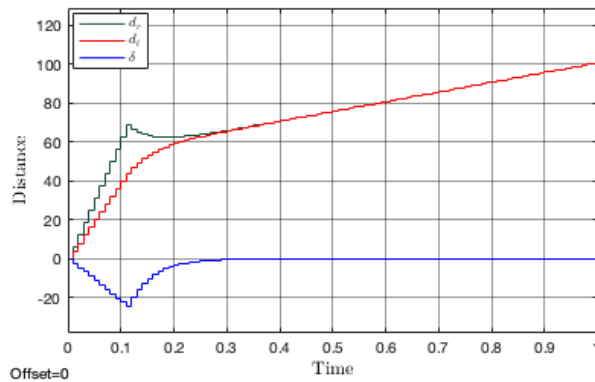## Examples of System Behavior with Various f-Values

Figure 1: $f_L = 0.1, f_R = 0.1$

With $f_L = f_R = 0.1$, our system eigenvalue is 0.8, so, since our system is one-dimensional, δ is multiplied by 0.8 at each timestep, slowly driving δ to zero. Since our f-values are equal, the d_l and d_r lines approach each other at equal rates.
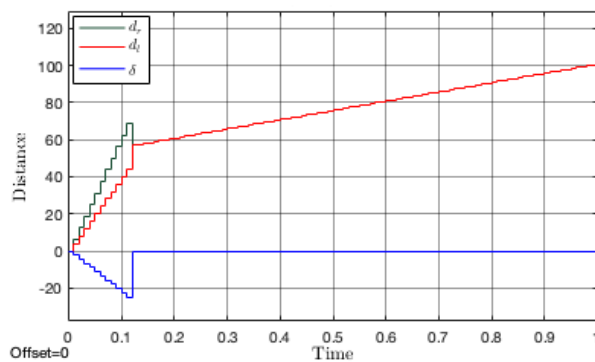

Figure 2: $f_L = 0.5, f_R = 0.5$

With $f_L = f_R = 0.5$, our system eigenvalue is zero, and since our system is one-dimensional, this means it takes one timestep to send δ to zero.
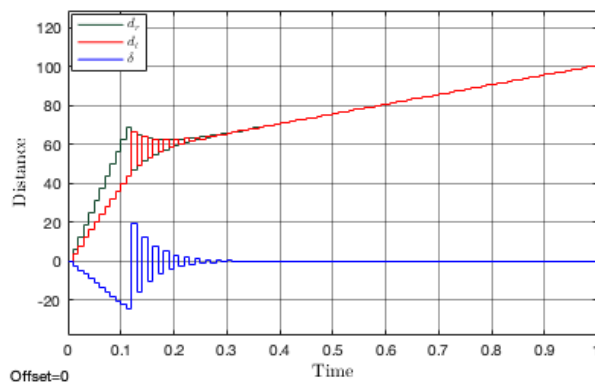

Figure 3: $f_L = 0.9, f_R = 0.9$

With $f_L = f_R = 0.9$, our system eigenvalue is -0.8, and since our system is one-dimensional, δ is multiplied by -0.8 at each timestep. Since the sign of δ changes at each timestep, you see oscillatory behavior.
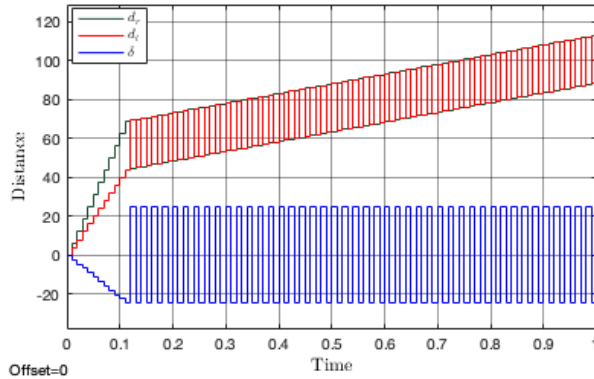
Figure 4: $f_L = 1, f_R = 1$

With $f_L = f_R = 1$, our system eigenvalue is -1. This is the marginally-stable case: δ's magnitude does not increase or decrease, but its sign changes at every timestep. Picture this behavior in a car: if δ switches sign at every timestep, the car is veering from side to side in a sinusoidal pattern while going forward.
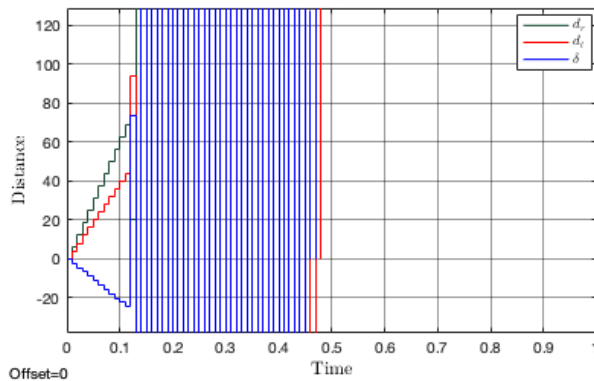

Figure 5: $f_L = 2, f_R = 2$

With $f_L = f_R = 2$, our system eigenvalue is -3. This case is oscillatory, like the previous case, but this case is unstable, since δ is multiplied by -3 at each timestep and so it grows without bound.
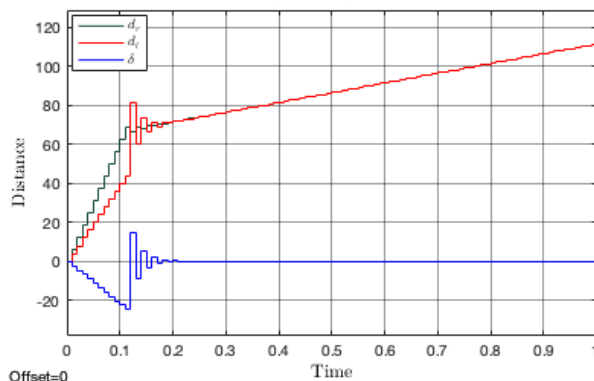

Figure 6: $f_L = 1.5, f_R = 0.1$

With $f_L = 1.5, f_R = 0.1$, our system eigenvalue is -0.6, so while the behavior is oscillatory, δ is still driven to zero and the system is stable. However, one of the wheel controllers (the left one, in this case) is working much harder than the other (i.e. the left wheel's velocity changes significantly while the right wheel's velocity remains the same). This means that the left wheel will need to be able to attain a larger range of velocities than the right wheel.

Sanity check question: Now that you know how the system responds to various eigenvalues, you can choose your feedback gains such that $1 - f_L - f_R$ is the eigenvalue that you want. Do you want them to be balanced (roughly the same magnitude) or unbalanced?

Steady-State Error Correction

Ideally, our system should be able to drive δ to 0 over an infinite timespan. However, due to mismatch between the physical system and the model, some steady-state error will be present. We will approximate this value $\delta[i] \stackrel{n\to\infty}{=} \delta_{ss}$ by using the final value of δ your car exhibits at the end of a run after your controller has been implemented.

> **Sanity check question:** What would the path of a controlled car with nonzero steady-state error look like? How is this different from the path of a controlled car with zero steady-state error?

Let's assume there is only a mismatch between β in our model and reality; i.e. β^* is the real model and we mistakenly are using β.

$$v_L[i] = d_L[i+1] - d_L[i] = \theta_L u_L[i] - \beta_L^*$$
$$v_R[i] = d_R[i+1] - d_R[i] = \theta_R u_R[i] - \beta_R^* \tag{8}$$

where $u_L[i]$ and $u_R[i]$ (Equation 5) are the inputs we calculate from our closed-loop model.

$$v_L[i] = d_L[i+1] - d_L[i] = \theta_L[\frac{1}{\theta_L}(v^* - f_L\delta[i] + \beta_L)] - \beta_L^*$$
$$v_R[i] = d_R[i+1] - d_R[i] = \theta_R[\frac{1}{\theta_R}(v^* + f_R\delta[i] + \beta_R)] - \beta_R^* \tag{9}$$

Solving for $\delta[i+1]$ in terms of $\delta[i]$ in the same way as before, we get:

$$\delta[i+1] = (1 - f_L - f_R)\delta[i] - [\beta_L(\frac{\beta_L^*}{\beta_L} - 1) - \beta_R(\frac{\beta_R^*}{\beta_R} - 1)] \tag{10}$$

Just to get a sense of the magnitude of the second term $(-[\beta_L(\frac{\beta_L^*}{\beta_L} - 1) - \beta_R(\frac{\beta_R^*}{\beta_R} - 1)])$ in the previous equation, let's say there is a 10 percent mismatch (i.e. $\frac{\beta^* - \beta}{\beta} = 0.1$) between our model and reality, and that $\beta_L = \beta_R + 10$. The second term will become -1.

The magnitude of the error is not that large, but let's see how it will translate to the steady-state error. First, let's simplify the model by letting $\lambda = 1 - f_L - f_R$ and $\varepsilon = -[\beta_L(\frac{\beta_L^*}{\beta_L} - 1) - \beta_R(\frac{\beta_R^*}{\beta_R} - 1)]$:

$$\delta[i+1] = \lambda\delta[i] + \varepsilon \tag{11}$$

You can show that $\delta_{ss} = \frac{\varepsilon}{1-\lambda}$ if $|\lambda| < 1$. You can reach this conclusion if you iteratively expand $\delta[i+1]$, and write $\delta[i]$ in terms of $\delta[0]$. Let's say in our designed system, $\lambda = 0.9$. We can see that in this case, the small ε can be amplified by a factor of 10 in steady state. This conclusion was derived based on the assumption that only β has a mismatch. As practice, derive $\delta[i+1]$ in terms of $\delta[i]$, when there is a mismatch in the θ values instead.