Lab Note 7: Controls

Previously, we have built S1XT33N'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 S1XT33N 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; S1XT33N 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 S1XT33N'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, 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 (all 3 of which are constants) from System ID and hard-code these input values, as they won't change at all as your car runs in open-loop control. As a reminder, here are the equations for our model that we defined in the last lab:

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

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?). As a result, open-loop control does not work very well in practice since the car has no way of adjusting to perturbations and model mismatch present in the real world.

Jolt Calculation

Since we are starting our cars from rest, in order for the cars to move, we must first overcome static friction. However, simply feeding it the open-loop inputs may not be sufficient to allow it to do so. Thus, before our Arduino tries to control our car, we need to provide each of S1XT33N's motors with a large PWM signal for a brief time; we will call this the jolt. 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 and thus different jolt values in order to overcome static friction and start moving. If the motors need different duty cycles to start moving, it is evidence of one or more of the following:

- **Difference in motor parameters.** Your motors might have different armature resistances (different force to generate torque), which in turn limits the amount of armature current flowing through the motor for a given voltage. Higher resistances require more voltage to run.
- **Difference in motor efficiencies.** 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 of S1XT33N

Introduction

To correct for the imperfections of open-loop control, we will build a closed-loop controller that uses feedback to ensure that both wheels still drive at the same velocity. 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.

Let's first set up a series of expectations for our model:

- 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.

$$\delta[i] = d_L[i] - d_R[i] \tag{2}$$

- We will drive our model so that

$$v_L[i] - v_R[i] = 0 \tag{3}$$

  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.

- Because of that,

$$\delta[i+1] - \delta[i] = 0 \tag{4}$$

  This means that between each timestep, no wheel will advance by more ticks than the other.

- Notice the above definition means that $d_L[i]$ does not have to equal to $d_R[i]$ (and thus δ does not have to equal 0, just any constant). In our controls scheme, **we will choose to drive δ to 0** to minimize the amount of steady state correction we have to do later. However, due to model mismatches in the real world, you may find that δ will likely not be zero but some constant instead which can be approximated by Steady-State Error Correction.

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

Deriving the closed-loop model

In System ID, 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).

$$|\lambda| < 1 \tag{5}$$

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

$$\begin{aligned} v_L[i] &= d_L[i+1] - d_L[i] = \theta_L u_L[i] - \beta_L \\ v_R[i] &= d_R[i+1] - d_R[i] = \theta_R u_R[i] - \beta_R \end{aligned} \tag{6}$$

We want to adjust $v_L[i]$ and $v_R[i]$ at each timestep by an amount that's **proportional** to $\delta[i]$. 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 eventually make this difference constant. 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^*, δ tends to zero.

$$\begin{aligned} v_L[i] &= d_L[i+1] - d_L[i] = v^* - f_L\delta[i] \\ v_R[i] &= d_R[i+1] - d_R[i] = v^* + f_R\delta[i] \end{aligned} \tag{7}$$

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 6 for $v[i]$ equal to our desired equation 7 for $v[i]$, we solve for the inputs u_L and u_R:

$$\begin{aligned} u_L[i] &= \frac{1}{\theta_L}(v^* - f_L\delta[i] + \beta_L) \\ u_R[i] &= \frac{1}{\theta_R}(v^* + f_R\delta[i] + \beta_R) \end{aligned} \tag{8}$$

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.

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 δ to zero.

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 7 from each other.

$$\begin{aligned} \delta[i+1] &= d_L[i+1] - d_R[i+1] \\ &= v_L[i] + d_L[i] - (v_R[i] + d_R[i]) \\ &= v^* - f_L\delta[i] + d_L[i] - (v^* + f_R\delta[i] + d_R[i]) \\ &= v^* - f_L\delta[i] + d_L[i] - v^* - f_R\delta[i] - d_R[i] \\ &= -f_L\delta[i] - f_R\delta[i] + (d_L[i] - d_R[i]) \\ &= -f_L\delta[i] - f_R\delta[i] + \delta[i] \\ &= \delta[i](1 - f_L - f_R) \end{aligned} \tag{9}$$

Key points:

- Since we are trying to drive δ to zero, δ is our state variable. All other system variables, such as v_L, v_R, d_L, and d_R, can be obtained from δ, the initial conditions, and the inputs at each timestep i.
- 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

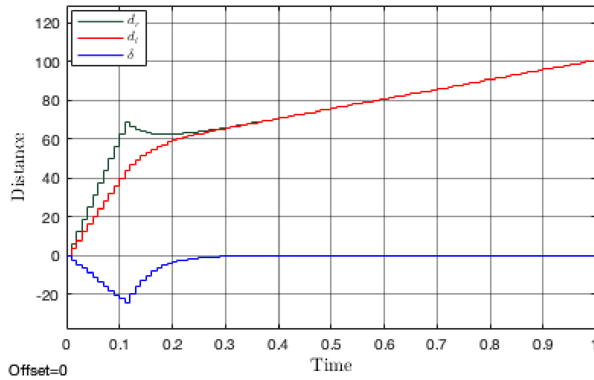


Figure 1: $f_L = 0.1, f_R = 0.1$

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.

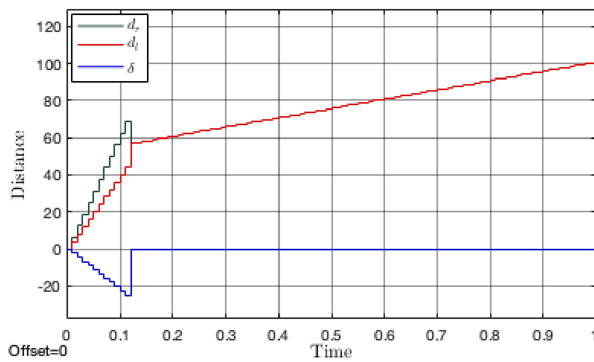


Figure 2: $f_L = 0.5, f_R = 0.5$

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.

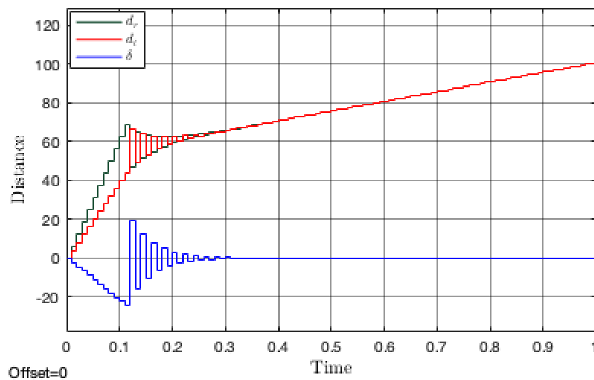


Figure 3: $f_L = 0.9, f_R = 0.9$

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.

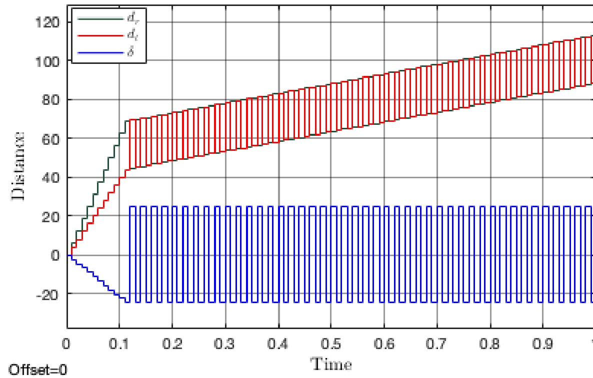


Figure 4: $f_L = 1, f_R = 1$

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.

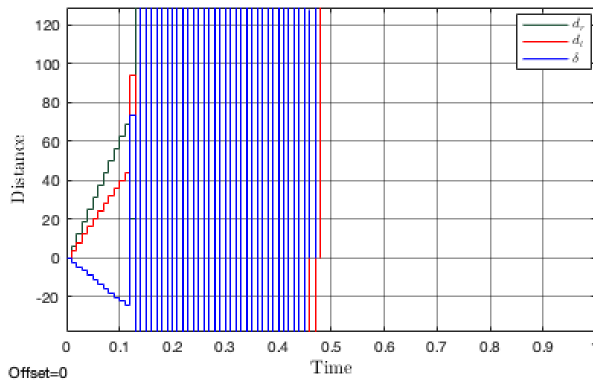


Figure 5: $f_L = 2, f_R = 2$

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.

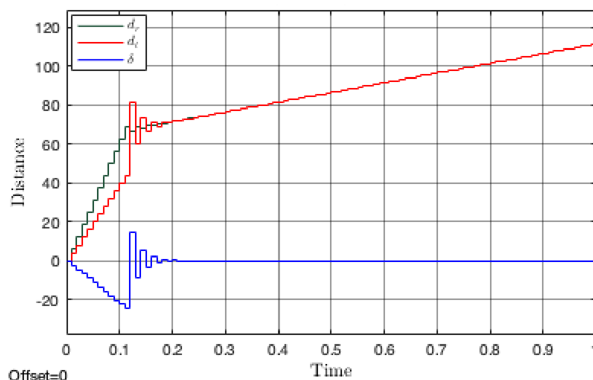


Figure 6: $f_L = 1.5, f_R = 0.1$

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 $\delta[i] \overset{i\to\infty}{=} \delta_{ss}$ by using the value of δ your car converges to 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.

$$\begin{aligned} v_L[i] &= d_L[i+1] - d_L[i] = \theta_L u_L[i] - \beta_L^* \\ v_R[i] &= d_R[i+1] - d_R[i] = \theta_R u_R[i] - \beta_R^* \end{aligned} \tag{10}$$

where $u_L[i]$ and $u_R[i]$ (Equation 7) are the inputs we calculate from our closed-loop model.

$$\begin{aligned} v_L[i] &= d_L[i+1] - d_L[i] = \theta_L[\frac{1}{\theta_L}(v^* - f_L\delta[i] + \beta_L)] - \beta_L^* \\ v_R[i] &= d_R[i+1] - d_R[i] = \theta_R[\frac{1}{\theta_R}(v^* + f_R\delta[i] + \beta_R)] - \beta_R^* \end{aligned} \tag{11}$$

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

$$\delta[i+1] = (1 - f_L - f_R)\delta[i] - [\beta_L(\frac{\beta_L^*}{\beta_L} - 1) - \beta_R(\frac{\beta_R^*}{\beta_R} - 1)] \tag{12}$$

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 $\varepsilon = -[\beta_L(\frac{\beta_L^*}{\beta_L} - 1) - \beta_R(\frac{\beta_R^*}{\beta_R} - 1)]$:

$$\delta[i+1] = \lambda\delta[i] + \varepsilon \tag{13}$$

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.

Part 3: Turning

Closed-Loop Control

We can get a clue about turning by considering how we implemented closed-loop control. We perform feedback control to minimize the difference between the two wheels ($\delta[i] = d_L[i] - d_R[i]$). When perturbations cause one wheel to get ahead of the other, the result is a non-zero δ, causing feedback control to turn the car to correct the error. Our feedback control policy is:

$$\begin{aligned} u_L[i] &= u_L^{OL} - \frac{f_L}{\omega_L}\delta[i] \\ u_R[i] &= u_R^{OL} + \frac{f_R}{\omega_R}\delta[i] \end{aligned} \tag{14}$$

Let's say the right wheel has moved further than the left ($d_R[i] > d_L[i]$), resulting in a negative $\delta[i]$. A negative $\delta[i]$ results in a reduction of $u_R[i]$, and an increase of $u_L[i]$. But what does this mean physically? If the right wheel is getting ahead of the left one, the left wheel will need to move a little faster and the right wheel a little slower for the wheels to even back out. Thus, feedback control corrects errors by turning to compensate for them. Thus, $\delta \neq 0$ can be used to turn the car. What if we artificially modified our $\delta[i]$ values to make the car think it was turning?

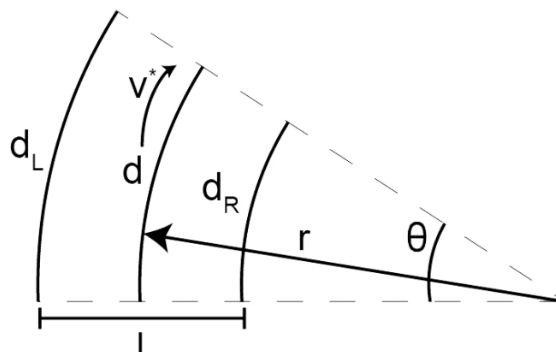
Turning via Reference Tracking

To turn, we will manually add some δ_{ref} to our δ value at each step of the control scheme. By doing so, we are tricking our control scheme into thinking that our car has turned by some amount to generate that extra δ_{ref} of error, causing it to turn in the opposite direction to compensate. When making a turn, the magnitude of δ grows because otherwise, the car is driving straighter. Therefore, we need to update δ_{ref} at each timestep to ensure that the car continues to turn.

Without loss of generality, we'll analyze a right turn, corresponding to adding a negative δ_{ref} value (the car thinks it's turned left, so it tries to turn right). Our goal is to generate this δ_{ref}. We would like the car to turn with a specified radius r and linear velocity v^*. The controller's unit for distance is encoder ticks, but each tick is approximately 1 cm of wheel circumference. Additionally, we want our car to turn gradually (rather than making a pivot turn), so δ_{ref} will be a function of the controller's time-step.
We define the following variables:

- i [time] - timestep
- r [cm] - turn radius of the center of the car; 1 cm $\approx$ 1 encoder tick
- d [tick] - distance traveled by the center of the car
- l [cm] - distance between the centers of the wheels; 1 cm $\approx$ 1 encoder tick
- ω [rad/time] - angular velocity
- θ [rad] - angle turned

Inspect the following diagram. Based on this geometry, we will derive an expression for $\delta_{ref}[i]$ in terms of r, v^*, l, i in the pre-lab/lab notebook.



Notes written by Mia Mirkovic, Meera Lester (2019)
Edited by Kourosh Hakhamaneshi (2020). Version 2.0, 2020
Edited by Steven Lu, Yi-Hsuan Shih (2021). Version 3.0, 2021.
Edited by Steven Lu, Megan Zeng (2022). Version 4.0, 2022.
Edited by Megan Zeng (2023). Version 5.0, 2023.
Edited by Junha Kim, Ryan Ma (2023). Version 5.1, 2023, 2024.