Lab 6: System Identification

In this lab, we will develop a linear model for the car system and collect data to estimate the free parameters. The specific goals of this phase are as follows:

- Understand the S1XT33N system model
- Collect data
- Use least squares to fit the data to our model and estimate the free parameters

Deriving the Car Model

As we continue developing S1XT33N, we'd like to better understand the car model that we will be using to develop a control scheme in a future lab. In order to model the velocity of each wheel, we must have a way to measure it; this is where the encoder comes in. As a wheel on the car turns, there is an encoder that is able to detect how fast the wheel is turning over a specific time interval.

As much as we would like our system modeling the velocity of the wheels to be linear, in reality our system (as with most real-life systems) is not. Modeling it accurately would involve a highly complicated model, which we do not want to implement in this class. So, instead of designing a complex nonlinear model, we will approximate the system with a linear model to work for small perturbations around an equilibrium point. You will locate this operating point in part 3 of the lab.

Linear models are particularly useful because their solutions, stability, and controllers can be studied with linear algebra. It is common practice to approximate a nonlinear model with a linear one that is valid near a desired operating point because only limited control methods are applicable to nonlinear systems.

The following model applies separately to each wheel (and associated motor) of the car:

$$v_L[n] = d_L[n+1] - d_L[n] = \theta_L u_L[n] - \beta_L \tag{1}$$

$$v_R[n] = d_R[n+1] - d_R[n] = \theta_R u_R[n] - \beta_R \tag{2}$$

Meet the variables at play in this model:

- n - The current timestep of the model. Since we model the car as a discrete system, this will advance by 1 for every new sample in the system.
- $d[n]$ - The total number of ticks advanced by a given encoder (the values may differ for the left and right motors– think about when this would be the case).
- $v[n]$ - The discrete-time velocity (in units of ticks/timestep) of the wheel, measured by finding the difference between two subsequent tick counts ($d[n+1]-d[n]$).
- $u[n]$ - The input to the system. The motors that apply force to the wheels are driven by an input voltage signal. This voltage is delivered via a technique known as pulse width modulation (PWM), where the average value of the voltage (which is what the motor is responsive to) is controlled by changing the duty cycle of the voltage waveform. The duty cycle, or percentage of the square wave's period for which the square wave is HIGH, is mapped to the range $[0,255]$. Thus, $u[n]$ takes a value in $[0,255]$ representing the duty cycle. The duty cycle is equal to the value of $\frac{u[n]}{255}$. For example, when $u[n]=128$, the duty cycle is roughly 50%, so the Arduino delivers a PWM wave signal that spends half of its period HIGH and the other half LOW to the motor circuits we built in the last lab. This circuit amplifies the voltage range of the PWM wave from [0V, 5V] from the Arduino output to [0V, 9V] across the motor, and delivers half the maximum power to the motor, with its average voltage across it being $\frac{9V}{2}=4.5V$. When $u[n]=0$, the duty cycle is 0%, and the motor controller delivers 0 V.
- $\theta_{L,R}$ - Relates change in input u to change in velocity: $\theta = \frac{\Delta v[n]}{\Delta u[n]} = \frac{v[n+1]-v[n]}{u[n+1]-u[n]}$. **Its units are ticks/(timestep $\cdot$ duty cycle).** Since our model is linear, we assume that θ is the same for every unit increase in $u[n]$, as it represents the slope of the graph of velocity versus $u[n]$. This is empirically measured using the car: θ depends

on many physical phenomena, so for the purpose of this class, we will not attempt to create a mathematical model based on the actual physics. However, you can conceptualize θ as a "sensitivity factor", representing the idiosyncratic response of your wheel and motor to a change in the duty cycle (you will have a separate θ for your left and your right wheel, as your two motors will differ in their sensitivity).

- $\beta_{L,R}$ - Similar to θ, β is dependent upon many physical phenomena, like static friction, so we will empirically determine it using the data we collect with the car. We model this β as just a constant offset in velocity, and hence **its units are ticks/timestep**. Note that you will also have a different β for your left and your right wheel. This β is also not related to the current gain of the BJT in the motor circuit, β_f.

Note that all mathematical models have their limitations, beyond which observations of the system in reality start to deviate from the model. Our model only applies when we are near our operating point; if we stray too far from it, the model no longer models the physical system very well. To illustrate this, consider when $u[n] = 0$: our model tells us that the velocity should be equal to $-\beta$! Clearly, this is wrong; when we don't apply any input, the car is at rest, not moving backwards. We thus need to keep these limitations in mind when using our model so that we remain near the operating point.

Linear Least Squares

Before trying to control SIXT33N, we will first determine the system operating point: since your motors are not identical, we need to find an operating velocity that both motors can reach, as our end goal is to make sure SIXT33N can drive in a straight line. In order for the car to drive straight, the wheels must be moving at the same velocity, after all. However, the motors (and hence the wheels) have different achievable velocity ranges and sensitivities, so we need to ensure we choose an operating velocity that is actually achievable by both wheels. A good choice of target velocity is the midpoint of the overlapping range of the two motors' velocity ranges, as it provides us the largest margin of error on both sides of the operating point.

We will assume that velocity varies approximately linearly with applied $u[n]$, so we will collect data across a range of values for $u[n]$ and then perform a least-squares linear regression on a linear subset of the data (located around your operating point) to find θ and β and get our model.

Linear Regression

The goal of regression is to fit a mathematical system model to a set of observed data points. When we say "fit," what we mean is we want to determine the values of the free model parameters (in our case, θ and β) that allow the model to best approximate the real system performance that we measured and stored in our data.

Recall our linear model:

$$v_L[n] = \theta_L u_L[n] - \beta_L$$
$$v_R[n] = \theta_R u_R[n] - \beta_R$$

Going forward we will drop the L and R subscript to generalize our model equation, but just remember that we have different values of $v[n]$, $u[n]$, θ, and β for the two different wheels, so we will have 2 models, one for each wheel. We know $v[n]$ and $u[n]$: $v[n]$ is the measured velocity of the wheel (from the encoder data) and $u[n]$ is the input PWM value. Note that because $v[n]$ is the **measured** velocity we obtained from our data collection, there will be some perturbations/nonidealities that are introduced into the measurements; our model will not perfectly match what we observe, only approximate it. Thus, since in our model, $u[n]$ is multiplied by θ, while $v[n]$ is the result, we can set up the following equations:

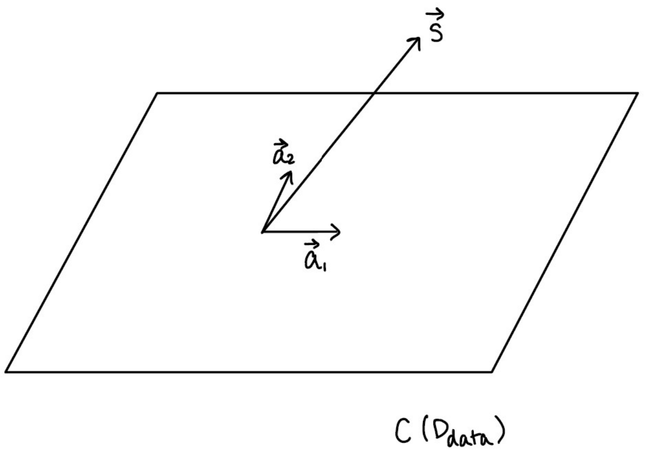
$$\theta u[0] - \beta \approx v[0]$$
$$\theta u[1] - \beta \approx v[1]$$
$$\vdots$$
$$\theta u[n] - \beta \approx v[n]$$

Now, we can condense the above linear equations into a single matrix-vector equation of the form $D_{data}\vec{p} \approx \vec{s}$ as follows:

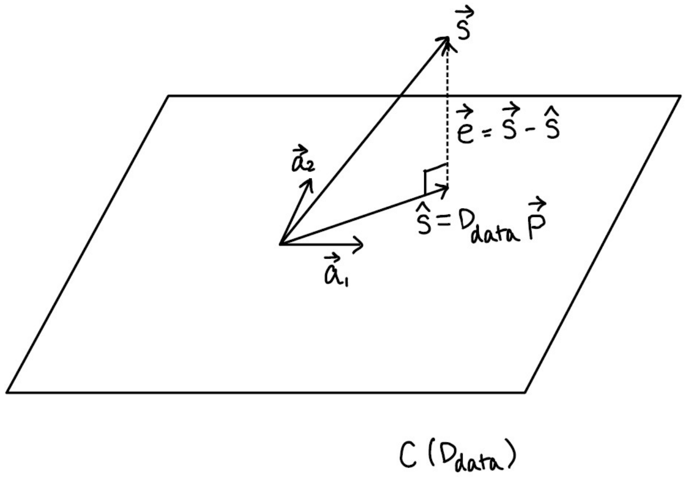
$$\begin{array}{ccccc} D_{data} & \vec{p} & \approx & \vec{s} \\ \begin{bmatrix} u[0] & -1 \\ u[1] & -1 \\ u[2] & -1 \\ \vdots & \vdots \\ u[n] & -1 \end{bmatrix} & \begin{bmatrix} \theta \\ \beta \end{bmatrix} & \approx & \begin{bmatrix} v[0] \\ v[1] \\ v[2] \\ \vdots \\ v[n] \end{bmatrix} \end{array}$$

Here, we know $u[n]$ and $v[n]$, so our unknown values we need to solve for are the θ and β that make up the $\vec{p}$ vector. Recall that the column space $C(D_{data})$ of a matrix D_{data} is the set of linear combinations of the columns of D_{data}, and the system $D_{data}\vec{p} = \vec{s}$ is solvable only if $\vec{s}$ (the velocity vector in this case) is in $C(D_{data})$. However, we already know that $\vec{s}$ doesn't fit our model perfectly, and is only approximated by our model as shown in the equation above, so it's probably not in the column space. So how can we solve for $\vec{p}$ then?

Let's examine this problem graphically. Consider $\vec{a}_1$ and $\vec{a}_2$ to be vectors in $C(D_{data})$ (i.e. they lie in the $C(D_{data})$ plane). $\vec{s}$ is not in $C(D_{data})$, so it does not lie in the plane. The plane $C(D_{data})$ represents the set of velocity vectors the wheel should reach, according to our model.



This is where the regression comes in. Rather than perfectly fitting the observed vector $\vec{s}$, we accept that we will have some error, and we want to find parameters θ and β so that our the model outputs the closest possible vector to $\vec{s}$ that is in $C(D_{data})$ (i.e. minimize the error). This vector is the **projection** $\hat{s}$ of $\vec{s}$ onto $C(D_{data})$, and $\vec{e} = \vec{s} - \hat{s}$ is the error between the model and the observation.



$\hat{s}$ is clearly in the column space of D_{data}, so it must be equal to D_{data} times some vector $\vec{p}$. We want to find the $\vec{p}$ that yields $D_{data}\vec{p} = \hat{s}$. $\vec{p}$ is the vector of θ and β in the matrix equation above, so what we're really saying is we want to find the θ and β values for our model that best approximate our observations.

Since $\vec{e}$ is perpendicular to $C(D_{data})$, we know the dot product $D_{data}^T\vec{e} = 0$. Since $\vec{e} = \vec{s} - \hat{s}$ and $\hat{s} = D_{data}\vec{p}$, we have

$$D_{data}^T\vec{e} = D_{data}^T(\vec{s} - D_{data}\vec{p}) = 0$$

and, solving for the parameter vector $\vec{p}$, we have:

$$D_{data}^T\vec{s} - D_{data}^T(D_{data}\vec{p}) = 0$$
$$D_{data}^T\vec{s} = D_{data}^T(D_{data}\vec{p})$$
$$\vec{p} = (D_{data}^T D_{data})^{-1} D_{data}^T\vec{s}$$

Having solved for $\vec{p}$, we now have the Least Squares approximation for the parameters θ and β that define our model!

Now you are ready to complete the lab! Go to the ipython notebook and complete parts 2 and 3 of the lab.

References

Chamberlain, Andrew. (2016) *The Linear Algebra View of Least-Squares Regression.* [online] Available at: https://medium.com/@andrew.chamberlain/the-linear-algebra-view-of-least-squares-regression-f67044b7f39b [Accessed October 14, 2019].

Notes written by Mia Mirkovic (2019)
Edited by Steven Lu, Bozhi Yin (2021). Version 2.0, 2021.