SIXT33N Project Report

Rina Lu (3033073864)[1], Arthur Goldblatt (3033678925)[2], Nada Jamalallail (3034247597)[3]

[1]rinaq@berkeley.edu,

[2]arthurgoldblatt@berkeley.edu,

[3]nadajamalallail@berkeley.edu

Demo for Rina Lu's project can be found here: https://youtu.be/24Jiv4vmpFY.

Documentation for the project can be found here: https://inst.eecs.berkeley.edu/~ee16b/su20/ (This project encompassed the later labs, titled Proj1-5)

Mic Board:

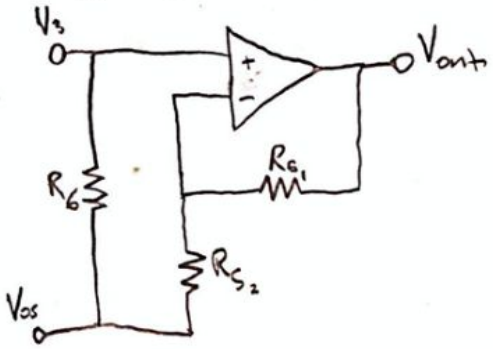
MIC BOARD TRANSFER FUNCTION

Ohm's Law : $V_1 = V_{DD} - R_1 k \sin(\omega t)$

Amp1 is unity gain: $V_2 = V_1$

$V_3 = V_{os} - R_1 k \sin(\omega t)$; C_1 gets rid of V_{DD} offset & output of Amp2 adds V_{os} *

$V_{os} = \frac{R_4 V_{DD}}{R_3 + R_4}$



Golden Rules:

(1) no current in or out of +, − terminals

(2) $V_+ = V_-$

Golden Rule 2: $V_- = V_3$

Golden Rule 1: $\frac{V_3 - V_{out}}{R_{S1}} = \frac{V_{os} - V_3}{R_{S2}}$

$\therefore V_{out} = V_3 + \frac{R_{S1}}{R_{S2}} V_3 - \frac{R_{S1}}{R_{S2}} V_{os}$

$= V_{os} - R_1 k \sin(\omega t) + \frac{R_{S1}}{R_{S2}} (V_{os} - R_1 k \sin(\omega t)) - \frac{R_{S1}}{R_{S2}} V_{os}$

$= V_{os} - 2R_1 k \sin(\omega t)$

$V_{out} = \boxed{\frac{R_4}{R_3 + R_4} V_{DD} - R_1 k \sin(\omega t)\left(1 + \frac{R_{S1}}{R_{S2}}\right)}$

* refer to HW 3 Q1 solutions

MAXIMIZING AMPLITUDE

Knowns: $K = 10^{-5}$, $R_1 = 10\,k\Omega$, & $R_5 = 50k\Omega$

- We want our off-set voltage to be 1.65 V because that is the middle of 0V & 3.3V.

$$V_{os} = \frac{R_4}{R_3 + R_4} V_{DD} = 1.65V \qquad (V_{DD} = 5V \text{ from lab})$$

$$\frac{R_4}{R_3 + R_4} = \frac{1}{3}$$

$$3R_4 = R_3 + R_4$$

$$R_3 = 2R_4 \quad ; \text{let } R_4 = 10k\Omega \text{ then } R_3 = 20k\Omega$$

- We want our amplitude to be 1.65 V as well.

$$\text{Amplitude} = R_1 K \left(1 + \frac{R_{S1}}{R_{S2}}\right) = 1.65V$$

$$\frac{R_{S1}}{R_{S2}} = \frac{1.65}{kR_1} - \frac{1}{1} = \frac{1.65}{(10^{-5})(10\times10^3)} - 1 = 15.5$$

We know $R_5 = 50\,k\Omega$: $R_{S1} + R_{S2} = 50\,k\Omega$

$$R_{S1} = 15.5(50k\Omega - R_{S1})$$

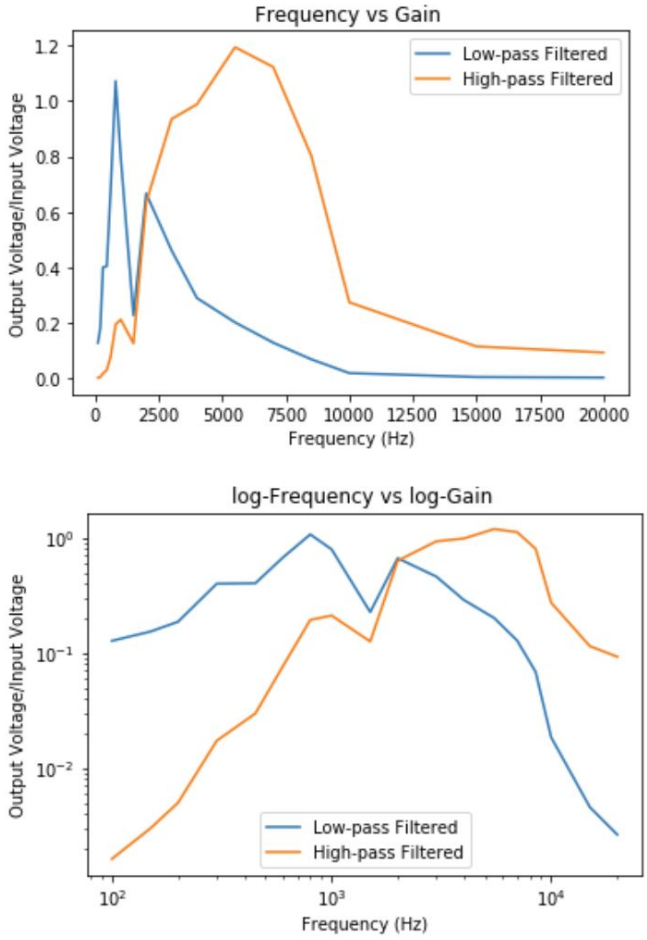
$$16.5R_{S1} = 775\,k\Omega$$

$$R_{S1} = 47\,k\Omega, \; R_{S2} = 3\,k\Omega$$

$V_{os} = 1.65V$, $R_3 = 20\,k\Omega$, $R_4 = 10\,k\Omega$, $R_{S1} = 47k\Omega$, $R_{S2} = 3k\Omega$

Mic Board Performance:

Our Mic Board did not quite match the optimal performance that we expected from the mic board transfer function. Instead, some frequencies had higher amplitudes than other frequencies. Whether this was due to physical imperfections, the nature of the filter with the microphone board circuit, or some other factor, we are not sure. We noticed that the microphone was more receptive to someone with a higher voice, aka more sensitive to high frequencies. Frequencies (or voices) that were too low tended to be slightly attenuated. This is reflected in the graph shown below.



Mic Board Description:

The mic board works by first converting noise signals into a variable current. The variable current alternates its frequencies based on the frequency of the noise signal. Then, that alternating current source causes an alternating voltage. This alternating voltage is fed through a unity buffer op-amp to deal with any loading issues. The DC portion of the signal is then removed with an RC-filter. It is then offset with a voltage supplied by amp2 and fed through a variable gain amplifier, whose strength is changed by adjusting the mic board potentiometer. The

off-set voltage is formed from a voltage divider that uses Vdd and ground as the source and drain.

Schematic:

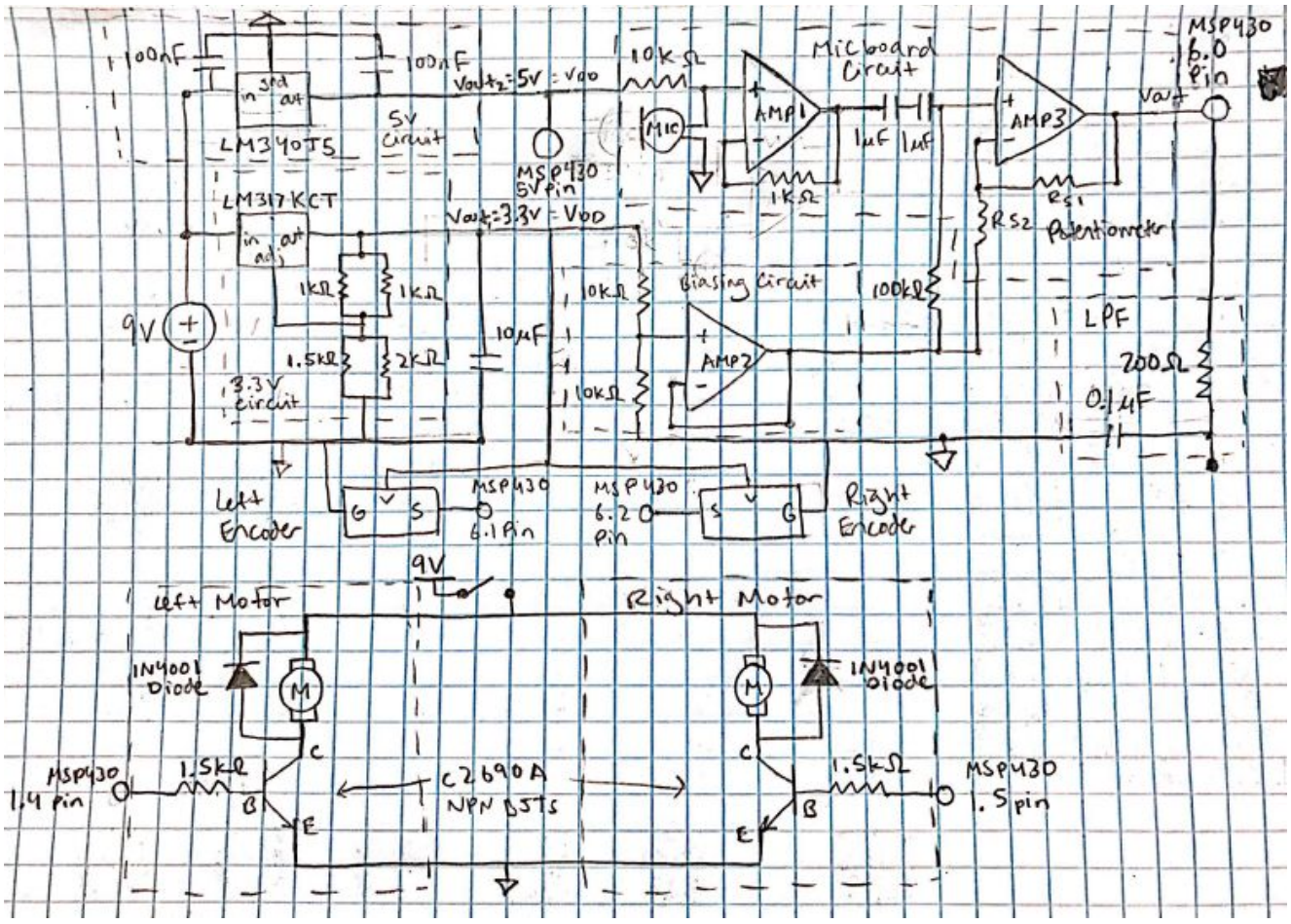


Figure 2: S1XT33N schematic (reference: Labs & first page of Lab Report PDF)

Control:

System Model:

The model equations are:

$$v_L[k] = d_L[k+1] - d_L[k] = \theta_L u_L[k] - \beta_L$$
$$v_R[k] = d_R[k+1] - d_R[k] = \theta_R u_R[k] - \beta_R$$

$v[k]$ is the velocity of the wheel at each time step. d[k] is the number of ticks calculated by an encoder at each timestep. We can determine the velocity with two ways: either by the difference between the number of ticks of the next time step and the current one, or by the equation $\theta u[k] - \beta$. $u[k]$ is the control applied to each wheel at each time step. θ relates $u[k]$ and $v[k]$. β is a constant offset in velocity of the wheel, and that is why we need to subtract it from $\theta u[k]$ to determine $v[k]$.

Closed-Loop Control: (reference: HW5&7 and Project Part 4 Lab Note)

In the closed-loop model, we want to decrease the difference between the position of the wheels to zero so that the car goes straight. First, we define the difference between positions to be the position of the left wheel minus the position of the right wheel. This can be determined by the data we got from the ticks of each encoder: $\delta[k] = d_L[k] - d_R[k]$. We notice here that the difference of positions will be positive if the left wheel is ahead and negative if the right wheel is ahead. In order to decrease the difference at each timestep k, we need to add a control to our model for the car to move straight. Since we define v^* to be our ideal velocity in SystemID, and we previously defined that $v[k] = \theta u[k] - \beta$, we need to first pick k_L and k_R so that we can get to our desired goal without the car oscillating. The velocities we want to reach are $v^* - k_L\delta[k]$ and $v^* + k_R\delta[k]$ to adjust the car to go straight. Rearranging the model equations, we can derive the control that can achieve this:

$$u_L[k] = (v^* + \beta_L)/\theta_L - k_L^* \delta[k]/\theta_L$$
$$u_R[k] = (v^* + \beta_R)/\theta_R + k_R^* \delta[k]/\theta_R$$

Finally, we plug this control to our initial model to get the model that "closes the loop" by making the car go straight:

$$v_L[k] = d_L[k+1] - d_L[k] = \theta_L((v^* + \beta_L)/\theta_L - k_L^* \delta[k]/\theta_L) - \beta_L$$
$$v_R[k] = d_R[k+1] - d_R[k] = \theta_R((v^* + \beta_R)/\theta_R + k_R^* \delta[k]/\theta_R) - \beta_R$$

In order to derive the system eigenvalue, we need to take a look at $\delta[k]$ at the next timestep.

$$\delta[k+1] = d_L[k+1] - d_R[k+1]$$
$$= (v^* - k_L\delta[k] + d_L[k]) - (v^* + k_R\delta[k] + d_R[k])$$
$$= \delta[k](-k_L - k_R + 1)$$

Thus, ($-k_L - k_R + 1$) is the eigenvalue. Since this system is discrete, the system in theory is stable when $|\lambda_i| > 1$. When we were solving HW7, we found that $\lambda \varepsilon (-1, 1)$ is considered stable, and thus had to narrow our options to which is considered "better," $\lambda \varepsilon (-1,0]$ or $\lambda \varepsilon [0, 1)$. We found that the latter is better since the former will cause oscillations.

Based on the previous Jupyter notebook simulation graphs and TAs' recommendations, I first chose my k values as 0.1 and then I increased the values until I reached the desired behaviour of the car going straight (at k = 0.5). Choosing these k values allowed the eigenvalue of the system to remain within [0, 1). This gain was better than the initial one since it allowed the car to converge faster. When I further increased my gain, I saw the car oscillate, which was expected since the gain is too high. The oscillation is how we know the eigenvalue has gone from positive to negative. From this part, we learned that an ideal gain shouldn't be too low that it would take the car too long to converge to a straight line, and also shouldn't be too high that it would make my car oscillate.

I personally had trouble with this part of the project. No matter what k value I chose, my car would always go left (even with a high gain, it would oscillate to the left). To debug, I tried changing the jolt value, but that didn't change the behaviour. I tried making the "stronger" motor weaker by increasing the resistance value of the motor circuit, but that didn't work. Emily then suggested changing the weight distribution of the car, and what finally drove the car straight was taping five 9 volt batteries at the right top edge of the car. The added weight created a more easily controlled system.

<u>Turning:</u>

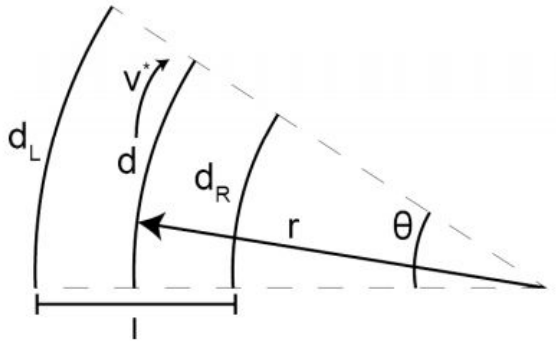
As stated above, rearranging the model equations gave us the desired inputs for each wheel:

$$\omega_L[k] = (v^* + \beta_L)/\boldsymbol{\theta}_L - k_L^* \boldsymbol{\delta}[k]/\boldsymbol{\theta}_L$$
$$\omega_R[k] = (v^* + \beta_R)/\boldsymbol{\theta}_R + k_R^* \boldsymbol{\delta}[k]/\boldsymbol{\theta}_R$$

To make the car go straight, we set the PWM for the left wheel and the right wheel based on the equations above, and set our input, delta, to be the difference between the left wheel position and the right wheel position, plus delta_ss.

To make the car turn left, we want to increase $\omega_R[k]$ and decrease $\omega_L[k]$. We do this by setting $\boldsymbol{\delta}[k]$ so that it appears the left wheel has travelled farther than the right wheel. Our system will try to correct this error by applying more power to the right wheel, which will ultimately cause our car to turn. Since $\boldsymbol{\delta}[k] = d_L[k] - d_R[k]$, we want $\boldsymbol{\delta}[k]$ to be positive. To make the car turn right, we analogously want to set $\boldsymbol{\delta}[k]$ to make it look like the right wheel travelled farther than

the left wheel, to make the system apply more power to the left wheel. This requires a negative $\boldsymbol{\delta}[k]$. To find our desired $\boldsymbol{\delta}[k]$, we derived the following equations based on the diagram below.



First, using the equation for arc length, $d = r\boldsymbol{\theta}$, we derived the equations for $\boldsymbol{\theta}$, d_L, and d_R.

$$\boldsymbol{\theta} = d/r = \boldsymbol{v}^*k/r$$

$$d_L = (r + \ell/2)\boldsymbol{\theta} = (r + \ell/2)\,(\boldsymbol{v}^*k/r)$$

$$d_R = (r - \ell/2)\boldsymbol{\theta} = (r - \ell/2)\,(\boldsymbol{v}^*k/r)$$

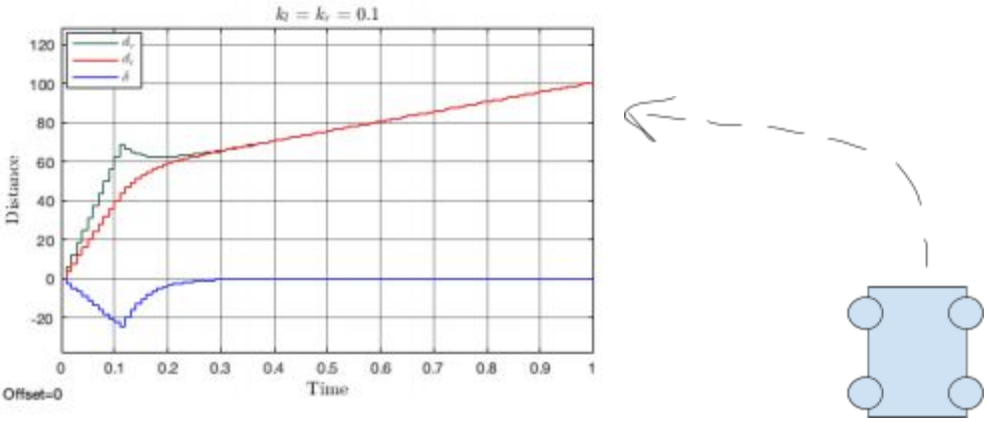
Next, we plugged in our new values into

$$\boldsymbol{\delta}[k] = d_L[k] - d_R[k].$$

$$\boldsymbol{\delta}[k] = (r + \ell/2)\,(\boldsymbol{v}^*k/r) - (r - \ell/2)\,(\boldsymbol{v}^*k/r) = \boldsymbol{v}^*k\,\ell/r$$

In our code, k corresponds to the number of steps, l is the car width, and r is our preferred turn radius. We implemented turning by subtracting delta_reference from our original delta value. We set delta_reference to $\boldsymbol{v}^*k\,\ell/5r$ to turn right, $-\boldsymbol{v}^*k\,\ell/r$ to turn left, and 0 to go straight.

Trajectory Error Correction:



For the car whose performance is plotted above, the car initially turns to the left because the right wheel moves a farther distance than the left wheel, before eventually going straight as it stabilizes.

For our car, adding delta_ss changed our car's trajectory by making it travel in the same direction as when it starts moving, rather than turning slightly at first. It is meant to correct errors that arise from external disturbances that affect the model system. It is different from using STRAIGHT_CORRECTION because it focuses on errors external to the car system, while STRAIGHT_CORRECTION deals with mechanical errors, such as wobbly wheels. For example, we scraped off too much of our wheels during the car's assembly, which contributed to the car veering off when it thought it was going straight.

SVD/PCA:

Justification:

To classify our words, we used envelope-based PCA. Envelope-based PCA is a good choice because it compresses our vector sizes and decreases the amount of information lost in classification. This linear projection is useful since while using a Launchpad, it is very easy to run out of space.

Data Processing and Classifier Design:

To help the classification process, we preprocess our data to make it cleaner and more uniform. We first align our data points so that each sample begins at the same time. We do this by defining a threshold value to determine when a speech command occurs, a pre_length value for when the command actually begins before the threshold is crossed, and a length value for how long the command takes place. Increasing the threshold helps to reduce noise, but increasing it too much can cause some parts of the sample to be lost. Similarly, having too small of a pre_length value can cause important data to be lost, but having too large of a value can add unnecessary noise. Then, we stack our data points and demean them to place them into a single matrix. In our A matrix, the rows represent different recordings, and the columns represent the features at different timesteps.

The matrix equation for SVD is $A = U\Sigma V^T$. V^T is a rotation matrix made up of the orthonormal eigenvectors of A*A. Σ is a scaling matrix with the square root of the eigenvalues of A*A on its diagonal, ordered by magnitude. U is a rotation matrix that undoes the rotation of V^T, and is also made up of orthonormal vectors.

We used three singular values, because the three largest singular values corresponded to the dominant principal components. We felt that three singular values were sufficient in displaying the variance of the data.

Classification Process:

After preprocessing our training data into a matrix, we performed PCA on the matrix using SVD. We created a new basis using the principal components we chose. Then, we projected our training data onto this new basis to display four clusters, one for each word, and found a centroid for each cluster. During the actual classification process, we converted the input into the microphone into a vector of data points. We preprocessed the vector in the same way we

processed our training data. Then, we demeaned the new data, projected it onto the PCA basis, and identified which centroid it was closest in Euclidean distance to. Since each centroid corresponds to a command, the closest centroid would determine which command to classify our word under.

We all chose different words, but tried to choose words with unique shapes and/or said them with different intonations. For example, choosing a word with two syllables with an emphasis on the last one, vs choosing a word with one long, drawn-out syllable. We did this to help the classifier differentiate between the words.

- Arthur: beep–beep–beep, tic-TACK, LEFT!, riiiggghhht
- Rina: WAHffle, puTAO, cake, banana
- Nada: yallAH, Go, banana, waTer

Performance: While our classifier performed well on our prerecorded training and testing data, it did not perform as smoothly when we tested it in real time. In the SVD/PCA lab each word had an 88-100% accuracy rate, but in practice, we found it was less accurate and highly dependent on how we said our words. One possible reason for this was that when it tested the model in the lab, it would use some of the 30 words we recorded at once. When we recorded we were careful to say the words the same way each time, and it's easy to get into a pattern of saying the words in a particular way during the recording stretch. However, later in practice, we'd be out of rhythm and did not necessarily say the words with the same stresses as when we recorded. We tried improving the real-time accuracy by adding a loudness threshold to make the car ignore random noise, as well as a classification threshold so that the car would only move when the sound closely matched one of the four words.

Integration:

What we learned: Planning is key. Especially when space and materials are limited. We could've avoided having to repeatedly rebuild circuit components by planning out the distribution of the circuit on the breadboard beforing building, and keeping it neat from the beginning. We also learned how to debug a circuit by compartmentalizing and analyzing individual components of the circuit, similar to how you debug code. Finally, we learned that taking caution and double checking voltages before connecting voltage-sensitive components is better than burning encoders and having to wait for an Amazon shipment.

Suggestions: If this lab continues to operate remotely, I would suggest providing two extra encoders. Frying one encoder means you will probably fry the other as well. In addition, I would update the labs to include clearer instructions—specifically when it comes to using the launchpad to keep it from burning out.

Favorite part: When the car became voice controlled.
Least favorite part: Burning parts.