Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

EECS16B Hands-On Final Lab Report

1) Midterm Lab Report Review

Summary:

In Introduction to Simulation, we built 2 circuits in Tinkercad making use of the LM741 op-amp as a buffer and as an inverter. For both, we observed the activity of 2 LEDs blinking pattern upon being powered by a voltage square wave generated by an Arduino.
In Analog and Digital Interfaces, we built a 3-bit and 4-bit DAC in TinkerCAD using resistor ladders and an Arduino. Each pin of the Arduino powering the DAC represented a bit, and the code for the Arduino provides an increasing and decreasing binary number to the DAC. Later in the lab, we turned the 4-bit DAC into a 4-bit ADC using an LM741 op-amp. We also built an analog input circuit using a potentiometer as our input to the op-amp. From this, we set different input voltages to the ADC and observed the binary outputs. Furthermore, we probed the output of the DAC and observed the conversion process of the ADC.
In Motion, we constructed motor controller and encoder circuits for S1XT33N and tested them. We built the motor controller circuits by outputting a PWM signal from the Launchpad MSP430 output pin which is connected to a BJT, and the motor. By changing the duty cycle of the Launchpad PWM, we can change the speed of S1XT33N. A diode was also connected across the motor to dissipate the current when the BJT turns off, mitigating the inductive properties of the motor and preventing a large voltage spike in the collector pin of the BJT. We used photo interrupters as an encoder, which work by interrupting a beam of light with the spokes of an encoder wheel that rotates at a rate proportional to the velocity of the car. By observing the number of interruptions in the beam of light, the Launchpad can determine how far and fast the wheel is turning over the specified time interval.
In Sensing Part 1, we built the voltage regulator and biasing circuits with the mic board. We also tuned the mic board and measured/recorded the frequency response. We used an LM340T5 for the 5V regulator circuit and an LM317KCT for the 3.3V regulator to supply those voltages to other parts of our circuit. The biasing circuits consisted of microphone gain, buffer, remove mic drift, and variable gain amplifier stages. During the remove mic drift stage, we introduced a DC offset to center the signal at 1.65V since the Launchpad cannot accept negative voltages. To prevent this offset from being amplified during the amplifier stage, we use the 1.65V as the reference for the non-inverting amplifier instead of the ground. To tune the mic board circuits and measure the frequency response, we probed the mic board output with the oscilloscope, generated a tone in the range 1500 - 2000 Hz, and tuned it until the oscilloscope showed a peak-to-peak 2 - 2.5V and center of 1.65V. Then we recorded the peak-to-peak voltage of the mic board output for several other tones.
In Sensing Part 2, we designed, built, and tested a band-pass filter to partition the audible frequency spectrum of the mic board. When designing the filter, we chose our cutoff frequencies and calculated the appropriate resistor and capacitor values to build the lowpass and high pass filters that satisfy the cutoff frequencies. To build the band-pass filter, we built a low-pass filter and high-pass filter using the calculated resistor and capacitor values and cascaded them. For the low-pass filter, we used resistor and capacitor values corresponding to a cutoff frequency of 2500 Hz, since most of the human vocal range falls below 0-2500 Hz. We also generated frequency response plots of the low-pass and band-pass filters.

Questions:

Describe the connection of each of the following labs with the S1XT33N car project:

1. **Introduction to Simulation**
   This lab required the use of the LM741 op-amp as a buffer and as an inverter, as well as powering a circuit by a voltage square wave. This is related to the S1XT33N car project, as it allowed us to become familiarized with the lab equipment and debugging strategies that we would need in the future.
2. **Analog and Digital Interfaces**
   DACs and ADCs like the ones we built in this lab allow us to bridge the real world with electronics by allowing digital electronic equipment to process the analog signals from the world, and vice versa. This relates to the S1XT33N car project, as we "teach" the car to process sound signals from the world and respond accordingly.

Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

3. **Motion**
   The motor controller circuits that we built in this lab are related to the S1XT33N car project because the Launchpad has 5v logic and very low current output, it is unable to output the 9V that the motors require. In general, a low duty cycle PWM signal will cause the car to drive faster, and a high duty cycle will cause the car to drive slower. With them, we can power the wheels of S1XT33N and allow it to move, while the encoders provide information to the Launchpad about the distance and velocity of the car's movement. The encoders are important for S1XT33N to know about how it is driving and "correct" itself during closed loop feedback control.
4. **Sensing Part 1**
   The voltage regulator circuits that were built in this lab are directly related to the S1XT33N car project, since they were necessary in order to supply 3.3V and 5V to parts of our circuits that the 9V battery cannot supply directly, such as the mic board, which required 5V.
5. **Sensing Part 2**
   The band-pass filter that we built int this lab allowed us to isolate only the midrange frequencies, and cut out background/other noise based on cutoff frequencies that we chose ourselves. This is related to S1XT33N car project because it partitions the audible frequency spectrum of the mic board, allowing S1XT33N to "hear better" in a sense as it is ignoring sounds that are not in the range of our future commands.

2) System ID

Summary:
In this lab, we verified that our encoders were still functioning correctly, switched our car to a mobile configuration using the batteries as a power source, and collected data about our car's driving. We then performed least squares regression on the data collected on each wheel in order to approximate the system as a linear model. We did this by uploading the sketch `dynamics_data.ino` to our launchpad, which feeds a varying input PWM signal to the launchpad and records the position of the wheel. From this, we were able to gather information about how our motors responded to different input PWM signals. We did this with a PWM range of 100-200, as well as a smaller range covering 50-60 PWM. These data were written into data_course.txt and data_fine.txt, respectively. For our smaller range, we chose a PWM signal between 105 and 150 PWM. Using that data, we wrote a function to performs least squares regression on it and determine the values of θ and β for each wheel.

Questions:

1. **How did you choose the region to collect finer data on (data fine.txt)? Why is it important to choose such a region to run least-squares regression on?**
   We chose a PWM signal between 105 and 150 PWM, which we determined by looking at the data from the coarse data collection and seeing which PWMs gave us velocities for both motors that were both achievable and not too far apart. This is important because the response curve will be more roughly linear, and so it will be easier to run least-squares regression on it.
2. **What do θ and β represent physically, not mathematically?**
   θ represents a change in velocity for every change in input u. This can represent any external influence that changes as the car drives, such as the motor's sensitivity to changes in the PWM signal from maybe some slight nudge or collision.
   β represents a constant offset in the velocity for each wheel that can be attributed to some constant physical resistance to the motor, such as the floor's friction coefficient.
3. **Why do we have separate θ and β values for the left and right wheels?**
   We need to have separate values for θ and β because the values of these will be dependent on physical phenomena that might not necessarily be the same for both wheels, such as the sensitivity of the motor to the changes in input and the amount of friction they experience.

Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

4. **Why do we set v^* to the midpoint of our overlapping wheel velocity range, instead of closer to the boundaries? What would happen if we operated our car at a velocity outside of the overlapping velocity range?**
   We set v^* to the midpoint of our overlapping wheel velocity range to ensure that the operating velocity is actually achievable by both wheels. This choice also provides us the largest margin of error on both sides of the operating point. If we operated the car at a velocity near or outside the overlapping velocity range, we might get a wheel that is unable to achieve the operating velocity and hence the wheels will not be moving at the same velocity and the car will not be driving straight.
5. **Explain how using each of the following methods as our model can produce a better fit for the data we collected than our current linear model of the car: a) higher-order functions and b) piecewise-linear functions. You may not manipulate the data set in any way (i.e. remove outliers, collect more data points, etc.).**
   The data we collected relates PWM signal to motor speed. This data may not actually follow a linear relationship due to the way the motor is designed and its physical limits, so there are a few options we can use. A higher-order function such as x^2 gives the data more room to vary, like following a curve. A piecewise-linear function also may be appropriate if the motor has several voltage ranges where the speed increase is linear. However, we must not overfit the data, or else we run the risk of losing the actual trend of the points. For example, using an overfitted higher-order function may result in outliers influencing the model even more than normal, and using too many steps for a piecewise-linear function can approach a higher-order function.

3) Controls Part 1

Summary:

In this lab, we designed an open-loop controller and a closed-loop controller that uses the desired wheel velocity to set the input u to an appropriate PWM value. To start the motors of the car from rest, we applied a maximum PWM to each wheel. We obtained these values by using the Utils.find_jolt with our data from data_coarse.txt. Since each motor may respond differently to the input PWM, we want to use jolt values for each motor that would result in both wheels achieving the same velocity from the start. The inputs will be the jolt values that are associated with whichever is the lower velocity between the left wheel's and the right wheel's maximum velocity. While implementing the closed-loop controller, we chose values for our feedback gain. We tuned the f-values so that f_left = 0.4 and f_right = 0.6. These values produce a system eigenvalue of 0, which is stable. We did this because we initially observed that our car was shifting to the left quite strongly. With these f-values, we tell the right wheel controller to "work harder" and the right wheel will experience a stronger correction and the velocity changes significantly compared to the left wheel.

Questions:

1. **What are the open-loop model equations for our PWM input, u[i]?**

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

2. **What is the purpose of the jolts? Why might the left and right jolts be different?**
   The purpose of the jolts is to give the motors an initial PWM input to start the motors from rest so that they achieve the same velocity from the start. This is to minimize the effects caused by differences in motor parameters and motor efficiencies, or the mass of the car being distributed unevenly between the two motors in the first place. The left and right jolts may be different because each motor responds differently to an input PWM, so using the same jolt value for the left and right motors could cause the car's motors to start at different velocities and cause the car to immediately turn at the start.

Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

3. **Why does open-loop control fail? Why do we need to implement closed-loop control in order to have the car travel straight?**
   Open-loop fails because it has the limitation that it does not provide any position feedback of the car. We need to implement closed-loop control in order to provide the Launchpad information to detect that the car is turning. From the encoders on the motors of the car, we can determine the information about the distance traveled by each wheel and "feed it back" to the Launchpad as an input so that it can make a decision to increase or decrease the left or right wheel velocity. This decision helps the car adjust the trajectory and drive in a straight line.
4. **What are the closed-loop model equations for our PWM input, u[i]? Explain the purpose of each term.**

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

$u_{R,L}[i]$: the input to the system in duty cycle which changes the PWM delivered to the motor controller/motor.

$1/\theta_{L,R}$: change in velocity / change in input u a sensitivity factor for the response of the motor to the change in PWM

v^* : the target velocity that is desired, and the input should try to achieve this.

$f_{L,R}\delta[i]$: the difference in the distance traveled between the left / right wheels in a timestep multiplied by a feedback gain f

$\beta_{L,R}$: a constant offset in velocity to account for the effects of physical phenomena like friction / motor sensitivity

5. **Derive the system eigenvalue. Under what condition is the system stable (in theory)?**

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

coefficient that changes the state variable over time, so $1 - f_L - f_R$ is the system eigenvalue

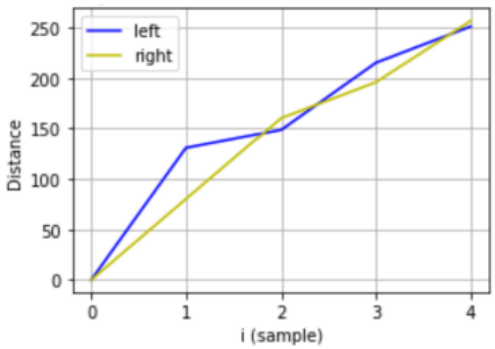
The system is stable when the eigenvalue is greater than or equal to -1.

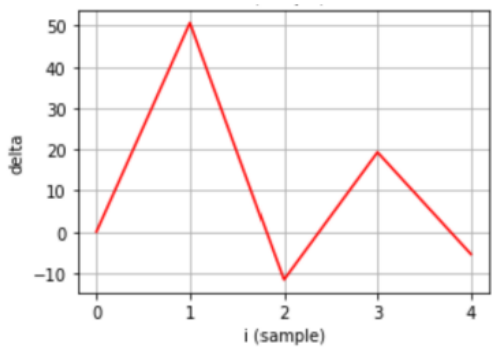
Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

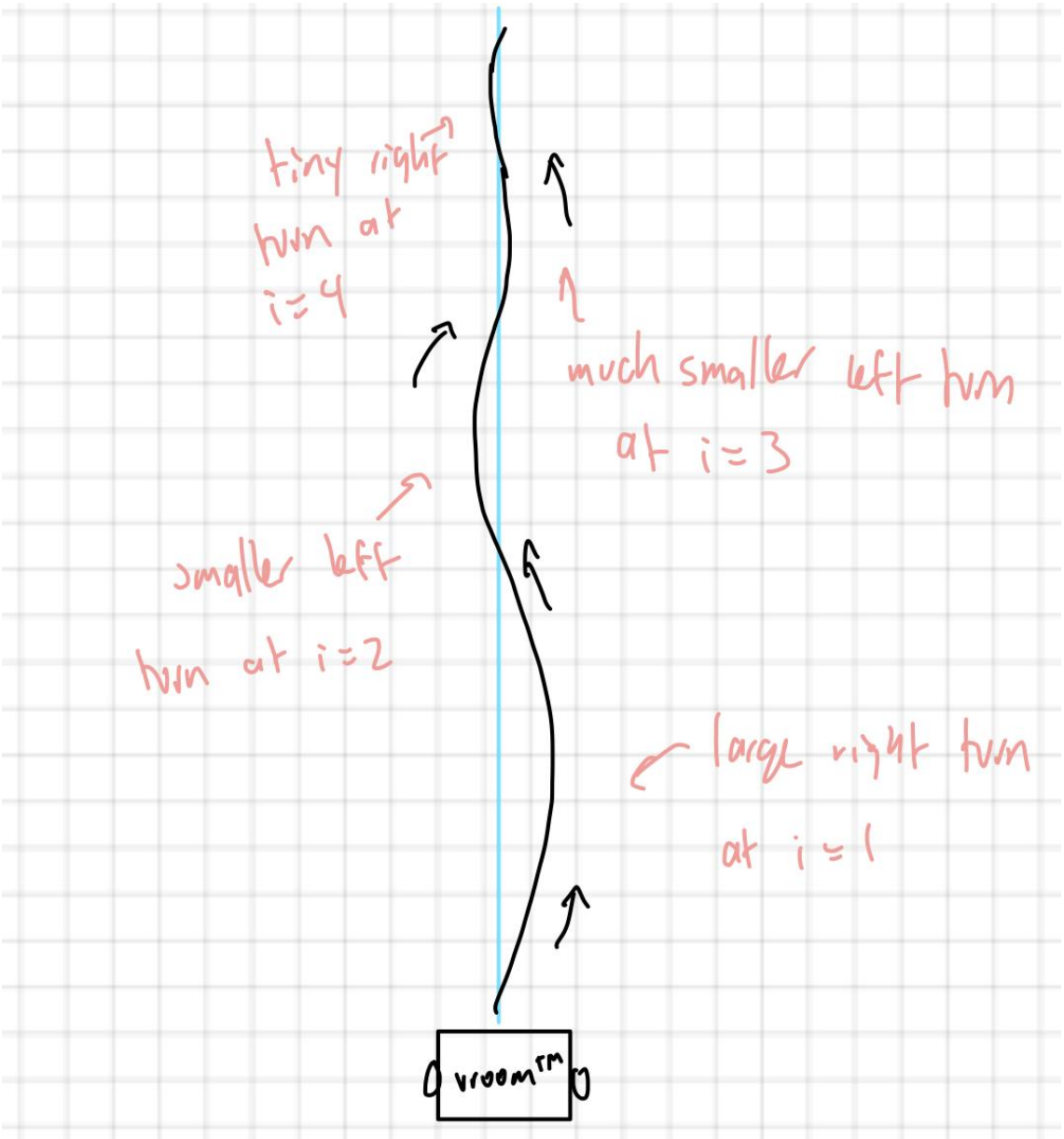
6. **When testing out different f-values in practice, how do you know if the system eigenvalue has gone from positive to negative based on the car's behavior?**
   When the system eigenvalue is positive, the sign of δ remains positive at each timestep. This translates to the normal behavior of the car assuming that the eigenvalue is a stable one. This will mean that δ approaches 0 and the wheels will receive corrections to achieve the same velocity. When the system eigenvalue is negative, the sign of δ changes at each timestep. This translates to oscillatory behavior in the car or turning from side to side.
7. **What effect does setting both f-values to 0 have on the car's control scheme? How is this different from non-zero f-values? Why are non-zero f-values necessary?**
   Setting both f-values to 0 will cause the car's control scheme to behave like open-loop control. This is different from non-zero f-values because the system eigenvalue will be 1, meaning that δ is multiplied by 1 at each timestep and there is no correction being applied to the wheel controllers. This is essentially the same as open-loop control, where there is no feedback to correct the motors.
8. **Why can't we use negative f-values for both wheels? If we wanted to use negative f-values for both wheels, how should we change our closed-loop model equations such that our car goes straight and corrects any errors in its trajectory?**
   If both f-values are negative, the system eigenvalue would never be less than 1. This would mean that δ is multiplied by at least 1 at each timestep and δ will never go towards zero, resulting in the distances covered by each wheel never approaching each other, and not achieving the same target velocities. If we want to change our closed-loop model equations so that negative values for both wheels are possible, we would want to add the f-values to 1 rather than subtracting them.
9. **What does a zero delta ss value tell you about your car's trajectory? What about a non-zero delta ss value? What kind of error is it supposed to correct when we add it to our control scheme? (Hint: Think about the difference between the trajectories for a zero versus a non-zero delta ss value.)**
   δss represents a steady-state error in the system that is caused by a mismatch between the physical system and the model. A δss that is 0 will mean that there is no steady-state error, and there is no mismatch between the physical system and the model. This means that the car's trajectory should be identical to the simulation, driving perfectly straight from the start. This means that the car drives straight with reference to the exact same bearing from when the car started moving. If δss is a non-zero constant, this means that the car drives straight according to itself, meaning that the car turns from its initial direction when set down on the floor, but drives straight eventually. By adding it to our control scheme, we can correct this mismatched bearing and add the steady-state error to the calculation of the error so that the error can converge to 0 instead of a nonzero constant. This allows the car to travel in (approximately) the same direction as when it is set on the floor and starts moving.

Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

10. **Draw the trajectory of the car whose distances and delta over time are plotted below. The blue line represents dl and the yellow line represents dr on the distances plot. Please clearly indicate the direction of your car's movement in your drawing.**







Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

4) Controls Part 2

Summary:
In this lab, we modified the closed-loop controller we made in Lab 7: Controls Part 1 so that we can get the car to turn. To implement turning with a specified radius r and linear velocity $v*$, we added an additional δref to our δ value in every time step during the turn. This makes it seem like the car has turned by some amount in some direction to the controller. This way, we are "tricking" the controller into correcting the fake turn by turning in the other direction. Further in the lab, we accounted for mechanical issues such as axle wobble or mismatch in the wheel sizes by setting a constant slight turn called STRAIGHT_RADIUS to use so that our car can go perfectly straight and turn left and right equally rather than turning slightly while trying to go straight. Finally, we verified our micboards by making sure that the biasing circuits and front-end circuitry still work as expected, so that we can use them in the following SVD/PCA lab.

Questions:

1. **How did you change the closed-loop model equations to allow the car to turn? Write the equations below and explain how they change for turning left, turning right, and going straight.**

$$u_L[i] = u_L^{OL} - \frac{f_L}{\theta_L}\delta[i]$$

$$u_R[i] = u_R^{OL} + \frac{f_R}{\theta_R}\delta[i]$$

In order to turn, we can manually add an offset to the δ[i] value to make the car calculate that it is already turning while it is not, causing it to correct and turn. To make the car turn left, we applied a positive non-constant offset δref to δ[i] for every timestep, and to turn right, we applied a negative non-constant δref to delta[i] for every timestep . δ[i] is equal to d_L[i] - d_R[i]. While the car is turning left, δ[i] will take a more positive value since d_L[i] will be greater than d_R[i]. Then, u_L[i] will become smaller, causing u_R[i] to be increased, turning the car right to continue going straight. If the car is turning right, δ[i] will take a more negative value since d_R[i] will be greater than d_L[i]. Then, u_R[i] will become smaller, causing u_L[i] to increase and turning the car left to continue straight. If the car is straight, ideally, δ[i] will be zero. u_L[i] and u_R[i] may not be the same, though, since the car may need different inputs to each wheel to run straight.

2. **Describe how the trajectory of the car would look if we added a constant δ_{ref} instead of a δ_{ref} that changes as a function of the timestep.**
The ideal turn we want is a smooth, gradual turn. To achieve this, δ_{ref} must start small, increase until the middle of the turn, and then decrease to zero. If a constant δ_{ref} was added, the car will immediately turn on the spot where it is instead of turning gradually. In some situations, this might be helpful, but this is not what we are looking for in the lab.
3. **Why do we divide v * by $m = 5$ for the turning expressions?**
During sampling, data was collected every 0.5 seconds. However, the Launchpad runs at a "sampling rate" of 0.1 seconds. Let's say that the car was running at 2 meters per second (very fast). That means that in 0.5 seconds, the data measured a distance of 1 meter. Since we directly use this data in the Launchpad, if we did not divide the data by 5, it would result in the code thinking that the car moved 1 meter in 0.1 seconds instead of 0.5 seconds, leading to a velocity of 10 meters per second (too fast).
4. **How is using `STRAIGHT_CORRECTION` different from delta ss in Controls Part 1?**
If delta converges to a non-zero value, this indicates that the car will drive straight after some time, but not in the same direction it started moving in. To correct this, we collect data to determine what delta converges to, and use this value as delta ss. On the other hand, STRAIGHT_CORRECTION is used when the car does not travel straight, but rather turns over time. STRAIGHT_CORRECTION will apply a turning correction to the car's trajectory to counter the turn that was there before.

Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

5) Classification

Summary:
In this lab, we picked 6 words that we considered to each have a different number of syllables and intonation. We then collected data by recording the words being spoken by a text-to-speech tool to form our data set. To collect the data, we uploaded a sketch `collect-data-envelope.ino` which collected 2-second long recordings every 0.35ms. Data from this recording was written into a .csv file. To clean up our data set and remove any recordings where the full word wasn't captured, we graphed the data as a line plot in Excel, identified the individual recordings whose plots appeared to be cut off or consisted of empty samples of noise or silence, and removed those rows. We then preprocessed our data to align the words before splitting it into a training set and a smaller test set using a 70:30 ratio. Preprocessing aligns the samples for each word to account for the fact that recordings of the same word will look very different due to factors like when the word was said and how quickly it was said. So we align the audio recordings together using length, pre_length, and threshold values. With the data for 6 words, we created and used the PCA classifier that allows S1XT33N to tell the difference between commands. PCA creates clusters from our data that we can use to classify data according to the centroid that it is closest in Euclidean distance to. From this, we can see how well the PCA separates the training set and then pick 4 words that are the most separated/unique and use them for our car commands.

Questions:

1. **What are some characteristics of a good set of four words for classification? Provide at least two features.**
   A good set of words for classification contains words that are easily distinguished from each other. This is because SIXT33N cannot process words the same way humans can, and it requires simpler features of words to distinguish them. These features can be the number of syllables, intonation, and the magnitude, all of which make up the shape of the speech wave. The words in a good set should have very different features from each other so that misclassification does not occur when the words are too similar.
2. **What are the length, pre-length, and threshold for our data processing? Include both the definitions and the values you chose.**
   The threshold is what we use to decide when the word has started in the recording. It is defined as the fraction of the maximum value of the data, and we use it by determining that a speech command has begun when any signal crosses the threshold. The pre-length is what we use to include the part of the word from before the threshold was reached. It is defined as a number of samples to keep from the first couple samples of the speech command, and we used it by saying that the command actually starts pre_length samples before the threshold is crossed. The length is what we use as the "window" to capture the entire command. It is defined as the total number of samples that your word takes to say in each recording, and we use it to take a window of the data that is length long and captures the entire sound of the command. For our data preprocessing, we used the default values for length, pre-length, and threshold, which are 80, 5, and 0.5, respectively.
3. **Why do we process our data so that the words are aligned before we run SVD/PCA on it?**
   Preprocessing so that the words are aligned accounts for the fact that recordings of the same word will look very different due to factors like when the word was said and how quickly it was said. By aligning the words, we can organize the samples and see how unique the words may actually be. This helps us determine how well the PCA will be able to tell the words apart if the aligned samples for each word still look similar.
4. **Why do we need to use SVD/PCA to represent our data set?**
   We need to use SVD/PCA to represent our data set because the Launchpad has limited resources available. Because of this, storing and using each word's vector and mean vector is not possible. As a result, we need to compress the vectors to a size that is manageable by the Launchpad, and we are able to compress it down to 3 PCA vectors. To prevent information loss while compressing our vectors down to a size that is usable with the Launchpad, we need to implement PCA using SVD. We implement PCA using SVD to use in our classifier so that when classifying words, we only need to calculate distances from those compressed-length vectors.

Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

5. **Why do we use the V^T vectors for our lab instead of the vectors inside of the U matrix returned by SVD?**
   We use V^T simply because of how our data is stored. The rows of our data matrix contain the different recordings of the different words, while the columns contain the different samples/time steps. Our PCA input matrix from the data was created by by stacking all the data vertically, and V^T contains the row vectors that are the PCA vectors for the rows of the input data matrix. Because of this, the PCA basis we use for compression will be formed by the vectors in V, not U.
6. **Why can we simply take the dot product when projecting our recorded data vector onto the principal component vectors?**
   The vector projection of the recorded data vector **R** onto the principal component vector **PC** is $\frac{R*PC}{||PC||^2} * PC.$
   The PC vectors all are orthonormal, so they have unit norm. The dot product is already the projection of one vector **R** onto the other vector **PC** multiplied by the magnitude of the other vector **PC**. Since the magnitude of the **PC** vector is 1, the dot product is equal to the vector projection of **R** onto **PC.**
7. **Describe all the steps in the procedure we use to take a recorded data point and identify which word it is.**
   Before we classify the recorded data point to determine what word it is, we preprocess the previously collected data.. First, we perform enveloping and trimming of the recorded data to align the samples. Then we construct the PCA matrix by stacking all the data vertically. We project our data point onto the PCA basis and demean our data point using the projected mean vector. Using the centroids of the clusters, we can classify the demeaned data point by comparing its Euclidean distance to the centroids and picking the centroid that it is closest to.
8. **How many PCA vectors would we need to represent a data set that, when graphed, looks like a thin, straight line? How many would we need if the data set looks like a circle when plotted? How many would we need if the data set looks like a cylinder with a large base area and small height when plotted?**
   A thin, straight line can be represented by a single vector, since vectors can represent a line in any direction. A circle is a two-dimensional shape, so it requires two PCA vectors. A cylinder with a large base area and a small height can be approximated by a circle, which only requires two PCA vectors. However, if it is significantly tall, three PCA vectors will be required, since it is now closer to a three-dimensional shape rather than two.
9. **If we keep increasing the number of PCA vectors, how does the increase in accuracy with each subsequent PCA vector change?**
   While working through the lab, we noticed that there was not much of a difference in accuracy between two and three PCA vectors. After looking at the plots of the data in two and three dimensions, this made sense to us. There was already enough separation in the two-dimensional plot to separate the clumps of data, and the three-dimensional plot did not show any significantly larger separation between them. Therefore, the accuracy increases dramatically from one to two PCA vectors, then slightly from two to three, and significantly less with each further additional PCA vector.
10. **What is `EUCLIDEAN_THRESHOLD`? What is `LOUDNESS_THRESHOLD`?**
    LOUDNESS_THRESHOLD is used to separate the voice command audio from quieter background sounds. Our data collected on the command words had values mostly above 1500, while background noises were below 500, so we chose 1500 as our LOUDNESS_THRESHOLD. EUCLIDEAN_THRESHOLD is used to determine whether the measured voice command is close enough in Euclidean distance, which is essentially just 3D straight-line distance, to any of the recorded data centroids to be classified. This value was determined by observing the plots of the recorded data centroids and selecting a value that was slightly higher than a rough radius around the centroids.

Group ID: 115-001
Partner #1: Nicole Lee (leenicole@berkeley.edu)
Partner #2: Matthew Song (matthewsong@berkeley.edu)

6) Integration/Final Demo

Summary:
In this lab, we assembled code from our previous labs into a single Launchpad program in order to get SIXT33N to respond with an appropriate drive mode when it hears a specific command word. We did this by putting together code from Lab 8 `turning.ino` to implement wheel control, code from Lab 9 `classify.ino` to implement our enveloping, PCA, and classification functions. For our lab, we used the command words "amogus", "cringe", "hilfinger", and "your mom", which corresponded to the drive modes "go straight for a long-distance", "turn left", "go straight for a short distance", and "turn right". We were given a random sequence of 8 voice commands to demonstrate that our classification for each drive mode works correctly at least two times.

Questions:

1. **Briefly discuss what you learned throughout the S1XT33N car project and in the labs. What was your favorite part? Least favorite part? Please answer this question individually.**
   Matthew - Besides the various linear algebra and data processing techniques we learned in labs, I also learned how important of a role these concepts can play in modern applications such as AI and the classification of voice and video data. To me, it was surprising how simple things such as dot products and matrix transposing could somehow lead to a program that could respond to simple voice commands. My favorite part of this lab was finally getting the car to classify voice commands. I hadn't done anything like this in the past, so it was very cool to see it work, and of course, it's the culmination of a semester of lab work. My least favorite part was probably spending an extended period debugging and finally figuring out it was a hardware problem, like a fried Launchpad (sadly, we fried a grand total of 1), instead of our code or faulty connections.
   Nicole - My favorite part of this lab was seeing the visualization graphs with 2d and 3d representations for 3 principal components that were taken from vectors with a length of 40. The compression of the data with minimal data loss was very cool to see. My least favorite part was adjusting f-values to make the car drive straight and matching the velocities of the car's wheels, as well as implementing turning. While it was a concept that is common in any electric car project, it was extremely tedious and it made me sad. Perhaps the most essential skill that I learned in this lab that is necessary for any class I take in the future involving EE skills is how to properly debug circuits using tools like the oscilloscope, multimeter, and function generator without guidance.
2. **What was the most difficult bug you encountered this semester? How did you resolve the bug? What did you learn from the debugging experience?**

7) Feedback

Overall, the labs have been pretty good! We don't have other feedback to give.

8) Collaborators and Sources

Each member contributed equally to the lab report.
Nicole: I was responsible for writing the lab summaries. We split the questions equally among us, discussing when we were unsure about our responses.
Matthew: I added onto the lab summaries in places that we got stuck on, and worked on half of the questions.

The additional source we used was https://www.electronics-tutorials.ws/, a provider of free educational tutorials on all things electronic.