UNIVERSITY OF CALIFORNIA, BERKELEY

EE16B: DESIGNING INFORMATION DEVICES AND SYSTEMS II

SIXT33N: LAB REPORT

Authors:
ANDREW CHANG
DIXUN CUI
NATHAN HOROWITZ

Professor:
ANANT SAHAI

DECEMBER 12, 2019

TABLE OF CONTENTS

1 FRONT END CIRCUIT **2**
1.1 SCHEMATICS 2

2 SYSTEM ID **3**

3 CONTROLS **3**
3.1 CLOSED-LOOP 3
3.2 CONTROLLER VALUES 4
3.3 TURNING 4

4 PCA **4**

5 FUTURE IMPROVEMENTS **6**
5.1 ROBUSTNESS 6
5.2 FUNCTIONALITY 6

6 WHAT WE LEARNED **6**
6.1 PROJECT MANAGEMENT 6
6.2 REALISTIC LIMITATIONS 7
6.3 APPLICATION OF TOPICS 7

1 Front End Circuit

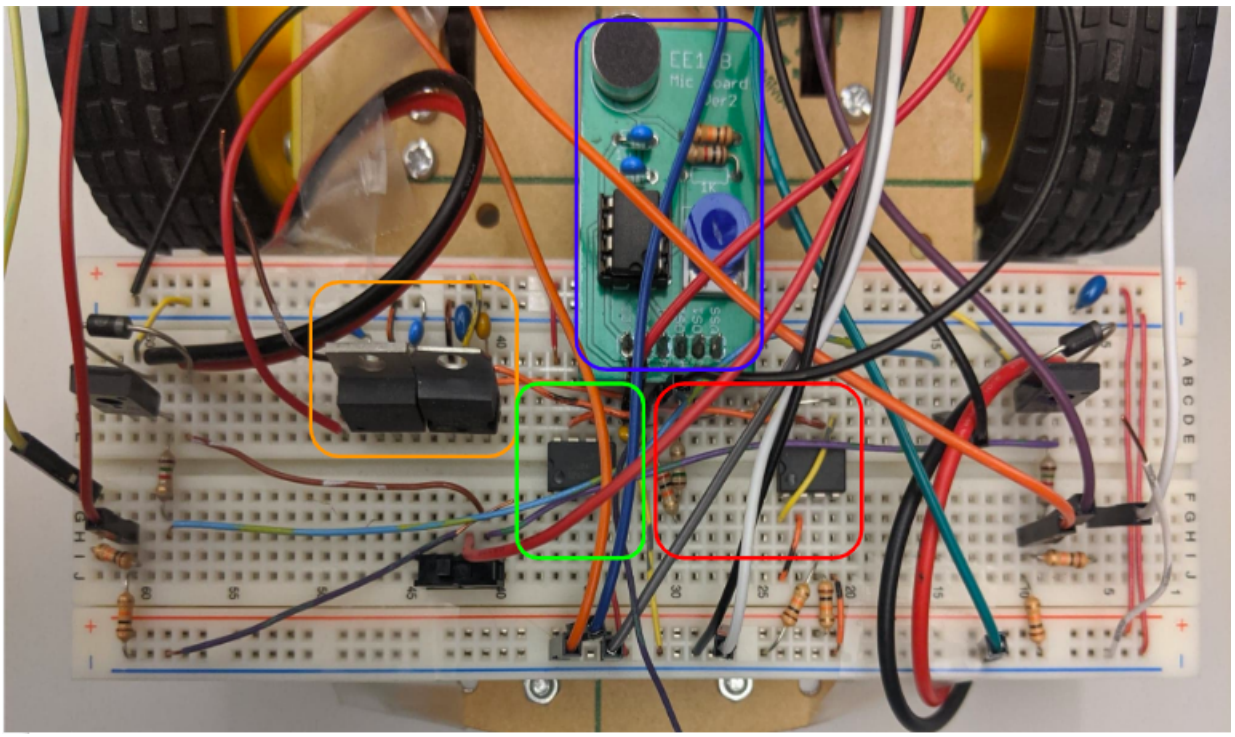
Mic-Board = BLUE
The Mic-Board was used to listen to voice commands and generate a voltage depending on the volume of the sound. This is how we take our inputs (from voice to voltage).

Filters = Green
We constructed a high-pass filter to filter out low frequency noise. Using a resistor of value of R=35000 and capacitor of value $C = 2.2e^{-8}$, we solve the expression f = 1/(RC*2pi) = 206.7 hz. By the construction of this high-pass filter, all frequencies under 206.7 hz will be filtered out. We need this step because most human speech falls between 250-2500 hz, so anything under this is considered noise.

Biasing circuit = RED
While our raw range of the Mic-Board is a certain value, we want our inputs to be between 0 V and 3.3 V, so we adjust the gain potentiometer on the mic board to make this possible (so that the loudest voice is 3.3 V and silence is 0 V). The now adjusted voltage uses the bias circuit to get our bound. 3.3 V split in 10K resistors means that the first input into the first opamp is 3.3/2 = 1.65 V. This 1.65 V is then added to the value of OS1, which is the offset, and then this goes through another opamp in OS2 to give the range 0-3.3 V.

Voltage Regulators = ORANGE
These regulators give us 3.3V and 5V rails given a 9V power source (from the batteries). We also added decoupling capacitors to minimize noise that could be coming from the power rail.

1.1 Schematics



See color coded parts in Front End Circuit (Mic-Board = BLUE, Filters = Green, Biasing circuit = RED, Voltage Regulators = ORANGE)

In addition to the essential circuit components described above, at each end of our breadboard, we had components that allowed us to power our motors. Each motor had its own subsystem and they were all

connected to a switch several breadboard columns across from the voltage regulators. These switches allowed us to prevent the motors from spinning when it was undesired.

2 System ID

Using System ID, we're able to use the data that we extracted by applying it to a linear model, and then using least squares linear regression on it to find a set of estimated parameters. With the parameters, we are then able to calculate the operating point that allows both wheels to achieve the same velocity. The linear model is as follows:

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

We first collected data by setting WRITE == 1 in dynamics_data.ino, and then uploading it to our MSP. Once this has been done, we press the leftmost bottom button which initiates the car to begin moving and proceed to collect data. After the data has been recorded, we read the data by setting WRITE == 0 in the same file, reuploading and then pressing the leftmost bottom button again, while the car is still hooked up to the computer. The data collected from the last run will then begin to be printed onto the Serial Monitor.

Once this is done, we copied the data printed on the Serial Monitor onto a text file called data_course.txt. Then, we picked a PWM range that is linear to the plot that we graphed on Jupyter Notebook using the data from data_course.txt, and collected data once again using the same step as mentioned before. We put this data into a file called data_fine.txt, and this is the data where we performed linear regression on.

The data we collected corresponds to v and u in our linear model, and essentially we're solving for theta and beta values through least squares linear regression. After doing least squares on Jupyter Notebook, our resulting theta_left and theta_right values were 4.095 and 2.425 respectively, and the beta_left and beta_right values were -2079 and -1043 respectively.

To calculate the operating point, we were provided with a cell in Jupyter Notebook that essentially finds the midpoint of the overlapping range of velocity. To this, we find the max velocity of the left and right wheels, then picking the minimum between the two max velocities. Then, we also calculate the minimum velocity of the left and right wheels, and then picking the max of the two minimum velocities. These two velocities is the overlapping range of velocity, and we find the midpoint by calculating their average, so adding them together and then dividing by 2. Our operating point was 1910.2.

3 Controls

3.1 Closed-Loop

The purpose of having a closed-loop design is to ensure that both wheels would move at the same velocity, even when exposed to unexpected environmental factors and model mismatches. This closed-loop design essentially provides feedback that will adjust our input in order to match the velocities of the two wheels. To derive the closed-loop model, we start off with the open-loop equations:

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

Then, we also come up with equations where the velocities are dependent on $\delta[k]$, where $\delta[k] = d_L[k] - d_R[k]$:

$$v_L[n] = d_L[k+1] - d_L[k] = v^* - k_L\delta[k]$$
$$v_R[n] = d_R[k+1] - d_R[k] = v^* - k_R\delta[k]$$

Finally, by setting the equations equal to each other, and solving for $u_L[k] and u_R[k]$, $u_L[k]$ and $u_R[k]$ becomes:

$$u_L[k] = \frac{v^* - \beta_L}{\theta_L} - k_L \frac{\delta[k]}{\theta_L}$$
$$u_R[k] = \frac{v^* - \beta_R}{\theta_R} - k_R \frac{\delta[k]}{\theta_R}$$

and then after plugging in for $u_L[k]$ and $u_R[k]$, our final model becomes:

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

To find $\delta[k+1]$ in terms of $\delta[k]$, we derive starting with $\delta[k+1] = d_L[k+1] - d_R[k+1]$:

$$\begin{aligned}
&\delta[k+1] = d_L[k+1] - d_R[k+1] \\
&= v^* - k_L\delta[k] + d_L[k] - (v^* + k_R\delta[k] + d_R[K]) \\
&= v^* - k_L\delta[k] + d_L[k] - v^* + k_R\delta[k] + d_R[K] \\
&= -k_L\delta[k] - k_R\delta[k] + (d_L[k] - d_R[k]) \\
&= -k_L\delta[k] - k_R\delta[k] + \delta[k] \\
&= \delta[k](1 - k_L - k_R)
\end{aligned}$$

3.2 Controller Values

We chose the values k_left = 0.5, k_right = 0.5, these values are stable as both are less than 1, this means that our car input motor values will not exceed the max amount. We chose these k values because this is what allows our car to drive straight, we need no extra bias to one side to allow our car to drive straight. This was partly due to the fact that in terms of physical weight, our car was very symmetric and had pretty similar pressure on the wheels and motors.

3.3 Turning

To turn our car, we can modify our control scheme. For some delta (which is our turning) we can define the following,

float driveStraight_left(float delta) { return ((v_star + beta_left)/theta_left) - k_left*delta/theta_left;; }

float driveStraight_right(float delta) { return ((v_star + beta_right)/theta_right) + k_right*delta/theta_right; }

In english, this means that in addition to maintaining our straight, we are adding or subtracting an additional amount depending on the size of our turn. K represents the scaling to maintain evenness, theta represents the ticks, and delta represents the turn amount.

To turn, we can compute our delta for some k as

Delta = CAR_WIDTH * v_star * k / TURN_RADIUS

Which represents the turn amount. If we wish to turn the other direction, we can multiply our delta by -1. CAR_WIDTH / TURN_RADIUS is just the arc length, which we then multiply by v_star and k to maintain our value in terms of ticks.

4 PCA

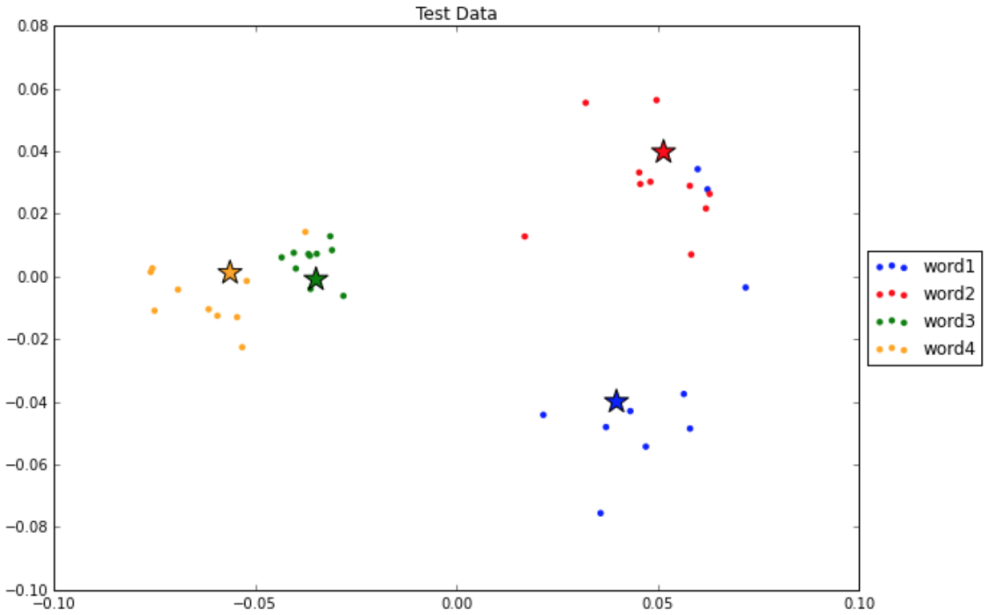
To be able to give our commands, we used PCA. PCA is used to find a basis in which variance between signals can be easily found. By making our signals as orthogonal as possible, we reduce the information

stored in the launchpad while also making classification easier.

First, we recorded our voice signals. We chose the word "baa," "boba," "SVD," and "KAKAKAKA." The reasoning behind our choices was that we wanted the loud syllables to come at different times within the recording period so that the words could be separated. Initially, we had planned to use words such as "left" and "right," but we realized that these one-syllable words may be hard to differentiate from each other.

Next, after our recordings, we demeaned our data matrix. After that, we used numpy to find the SVD and we found our PCA basis from the V matrix generated. The V matrix is the correct size and by taking the three columns corresponding to the largest singular values of our matrix, data in the new basis would show the separation between different signals.

After that, we projected our data onto the new basis. We found the centroids of each word so that in classification, we could check which centroid a point was closest to. The training data can be plotted on a 2D grid for reference:



Finally, to classify, we wrote a function that computes the minimum 3D Euclidian distance from each centroid. In our car's implementation, we increased the robustness by modifying a minimum distance for a word to count and by setting a loudness threshold. We set the k-means threshold to 0.04, referring to the distance to a centroid, and we set the loudness threshold to 700. Both these values were determined through iterating over potential values until we were able to classify our words at a success rate over 80%.

Overall, we found that words with different syllables were easiest to distinguish because of their vastly different profiles. To increase robustness and improve our classification, we used PCA to find a basis in which words could be easily classified. We used vectors corresponding to the largest singular values in order to essentially reduce our dataset size while still representing the variances in data points. From there, in the new basis, we found centroids and using certain thresholds, we were able to classify our words.

For our final implementation, we simply uploaded our data to our launchpad. With the integrated controls and classification code, our car was able to either drive straight for long, turn left, drive straight for short, or turn right.

5 Future Improvements

After the completion of our project, we felt like there were two major areas we could improve: robustness and functionality.

5.1 Robustness

We believed that most of the circuitry was capable and fairly effective for the tasks that were required. However, we ran into many problems each week of having circuit elements fall aparts. Batteries had to be reconnected and the heavy batteries often pulled battery wires out of the breadboard. This caused us to have to reassembly many parts of the circuit every week.

One way to improve the robustness of the car is to improve the design of the chassis. By gluing on custom slots, we could ensure that elements such as the batteries are locked into place when they are used. In addition, a large shell over most of the breadboard would prevent any additional damage.

Overall, these hardware improvements would make the car more durable. This would allow more efficiency while working on it which could lead to increased functionality.

5.2 Functionality

One major issue we ran into was that we felt like the mic board was not too capable of detecting different pitches. Instead, it seemed like it simply detected the different volumes of sound within a specific time intervals. Therefore, it was factors such as timing, loudness, and distance from the speaker that would affect the readings. To account for this, we chose words with varying numbers of syllables; however, when testing the words initial from the energia serial monitor, we found that distance and volume still played a major role. The limited capabilities as well as the additional factors restricts the number of different commands that can be performed.

We believe there are two ways to improve the system to be able to perform more commands. The first would be to look into getting a more sophisticated micboard. Doing so would allow for more consistent and effective readings. The next would be to control the volume and distance better. Instead of shouting out words, we could record our voice on our phones as sound clips. Then, we could control the distance and volume at which the sound is projected. One way to control distance would be to attach a string to the phone and to hover the phone at a height where the string is stretched out fully.

Making these changes would allow the micboard to pick up on different tones in a more controlled manner. More words and phrases could be used as different inputs, allowing us to program the car to perform more commands. These could include driving backwards, turning backwards, doing a 180 turn, and driving straight for more different increments.

6 What We Learned

The project was a great learning experience for us all, as we were able to apply concepts learned during class to create a cool application. The main lessons we learned can be described in three categories: project management, limitations experienced, and applications of concepts.

6.1 Project Management

For the project, we had to work on different increments of it over most of the semester. As a result, we had to be very organized. Not only did we have to make sure all our progress carried over; we had to

ensure that we kept track of decisions we made to write our report and also to make sure we know what to do in future weeks. Initially, we did not account for these factors and we ran into great inconvenience redownloading certain files and trying to figure out why we had chosen resistor values and why we had built our circuits the way that we did.

To make things easier for us, we decided to keep track of all our files and decisions better. We created a google drive for our project, where we had a sheets where we recorded any design decisions we made. In addition, after each lab, we uploaded all our files to a subfolder based off of the date. With that system, we could retrieve all the .ino and ipython files that we had for each day. This allowed us to continue working on ipython notebooks easily.

Learning to manage the project was a great experience. It improved our efficiency and made sure we kept good records of our project.

6.2 Realistic Limitations

Throughout the project, we encountered many limitations from the resources and time that we had. One major limitation we had was our hardware. We were limited by our hardware provided and as a result we had to make adjustments such as taping certain sections down. Also, we were given instructions on circuit modifications to fix issues that existed before.

Another limitation was our micboard and software. Due to variance in distance, volume, and because of our micboard that mostly detected volume instead of pitch, we could not use the command words we initially wanted, which would have been more specific terms referring to the commands. Instead, we had to use words that were different syllables, adjusting the commands we chose to achieve our goal given our limitations.

6.3 Application of Topics

Though we knew that going into the project, we would have to apply concepts learned during the course, it was interesting to see the necessity of using these concepts. One clear example was when we used SVD and PCA to reduce the space of our measurements. Though it seemed like a cool concept when introduced during lecture, during the project, we were forced to use PCA because the Launchpad could not contain all the data we originally had. We saw the necessity of these concepts and combined with other concepts such as filters and closed-loop controls, we had a great learning experience applying course concepts.