EE 16B Fall 2017
Final Project

SIXT33N Project: Written Report

Nathan Mar: ee16b-aiy
Justin Lu: ee16b-ajc
Brent Sienko: ee16b-acc

Circuit:

Parts of the Circuit:

Mic (Mic Board): The major design concern here is that our MSP board can only process positive values. Since the mic board output is originally balanced around zero, we needed to be able to balance it around about 1.65 V. This is what necessitated the potentiometer and resistor setup. In order to get the value to 1.65, we needed to ensure that the values that are fed into OS1 and OS2 are exactly the same. However, because resistors have a tolerance band that indicates possible drift (and when talking about resistances in orders of 10K ohms a percentage can mean quite a bit), we adjust a potentiometer to equalize the OS1 and OS2 values. This ensures that we are able to get the mic gain to precisely what we want it to be. The other potentiometer is used to adjust the sensitivity of our mic, by simply controlling the amount of voltage that gets outputted. We set this potentiometer to the point where speaking from a reasonable distance produces visible waves at the scale of 0 to 3.3 V.

Mic (Frequency filter): Because the mic is capable of producing very high frequencies and the MSP is sensitive to these frequencies, we set up a low pass filter that will keep high frequencies from affecting our MSP. In order to ensure the safety of the MSP, we create a double filter. However, because the currents and voltages from different filters will affect each other when constructed in series, we use a pair of op-amps as buffers.

Encoders: The encoders can detect the speed of the wheels turning by sensing the rate at which the holes in the encoder disk pass, and directly feed the information to the MSP. This gives us a good estimate of the speed at which the wheel is turning. The encoders receive power through the 3.3V line that the MSP generates.

Motors: The motors receive an input that comes through a bipolar junction transistor. While we do not learn specifically how it operates, we know that by pushing a signal (generated by the MSP) into what is called the collector, we influence the voltage that comes out of the collector. As such, we can control the voltage that passes through the motors and therefore its speed through the MSP.

Regulator: We use 9V batteries, and the MSP can only take in five volts, so we use a regulator that brings the voltage down from 9V to 5V in order to power the MSP.

Final Design:

The MSP checks every two seconds for inputs (sound). If one is processed as being sufficiently close to one of the chosen words, then the MSP generates an impulse that drives the motors forward. Depending on the input, the MSP generates a signal for either a short straight move, long straight move, left turn, or right turn. Each of these is controlled by a closed loop system in order to produce the results closest to what we want.

Gain and Frequency Response:

We can use our calculations in HW5 for the gains in the various stages of the circuit.
Mic Gain: V^+ = 3.3 - 10^4 * I_mic
Buffer: Gain = 1
Mic drift: w*10^5*10^(-6)/ sqrt((w*10^5*10^-6)^2+1), where w is the frequency of the mic
Amplifier: V_out = (1 + 51000/51000)*(V_OS1 - V_OS2) + V_OS2

PCA Classification:

Commands: The words chosen were Totodile, Giratina, Ash, and the ‘e’ sound held for about 1 second. These words have ample differences in their centroids. Because two of the words were a bit longer, we extended the size of the sample from 40 to 90 values in order to better differentiate. The ‘e’ sound was a

single clear tone, with very little variance during its signal. Ash was a shorter sound, and had a much higher and sharper peak. Totodile and Giratina had different emphasis, with the stress from Totodile appear near the front of the word, and the sharp 't' in Giratina. We attempted many other words besides these (including other Pokemon), but these had very clear and distinct clusters as a result of these words.

Processing:
We took around 12 samples for each of the words we chose. We then selected a part of the sample that begin once a threshold was met for a given amount of time. From there we created a matrix out of the values by stacking them all together vertically. We then subtracted the mean of each column in order to normalize each data set around zero. Our data was then sufficient to use PCA.

Controls:

Open Loop Model:
In the open loop model, we simply assume that each of the wheels are essentially the same. Both wheel motors are separately modeled by the equation $v(t) = d(t+1) - d(t) = \theta u(t) - \beta$, where $u(t) = \frac{v^* + \beta}{\theta}$ is the input to the system in PWMs, *t* is the current timestep, *d(t)* is the current number of ticks advanced by the wheel, *v(t)* is the discrete-time wheel velocity (ticks/timestep), θ is the measure of how much the motor turns per increase in PWM, β is the model of friction, and v^* is the target velocity. To find the parameters of the angle of the wheel as well as its friction, we gathered data from multiple driving tests and then performed linear regression to find values that worked best for our car. Using those parameters in our open-loop system, we attempt to power each motor turn at the same speed. Our system was thus governed by $\delta(t) = d_L(t) - d_R(t)$, ideally where $\delta(t \to \infty)$ converges to either zero or a constant value.

Closed Loop Model:
We then implemented a closed-loop system to better deal with the various forms of mismatch. To do so, in addition to the previously described open-loop system, we incorporated a proportional control k_L and k_R to $\delta(t)$ so that $|\delta(t)|$ does not explode. The new system is modified such that the inputs to the system continuously account for any mismatch: $u_R(t) = \frac{(v^* + \beta_R)}{\theta_R} + k_R \frac{\delta(t)}{\theta_R}$, $u_L(t) = \frac{(v^* + \beta_L)}{\theta_l} - k_L \frac{\delta(t)}{\theta_L}$. This closed-loop model is much-desired and necessary over the open-loop one in that it allows for the system to adjust itself in light of any external disturbances.

Choosing Controller Values:
The controller values we chose were 0.45 and 0.55. We incrementally increased our k values starting from around 0.05. Given very low values, the vehicle would take too long to reach its steady state, essentially making a very long turn. High values worked the best for us, and we found that it was most straight when the values were slightly unequal. For turns, we simply tweaked using the given turn radius so that one wheel would travel farther than the other.

General:
Learning how to quickly and effectively debug a circuit is also something that we worked at. Our mic board ended up failing once, so we had to rebuild that from scratch. We also ended up shorting the MSP at least once in the process of making the car. By probing using an oscilloscope and knowing what should be happening at each point in the circuit, we can debug to either figure out whether some error has been made in the creation of the circuit and find out where the shorting occurred and to not make the same mistake.

Figure 1: Final Front End Circuit Diagram

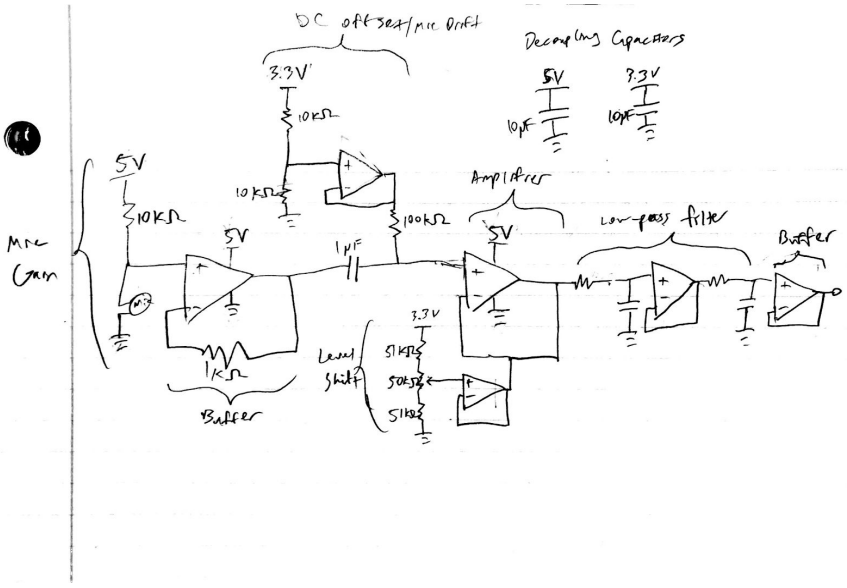

Figure 2: Closed Loop Control Scheme Block Diagram