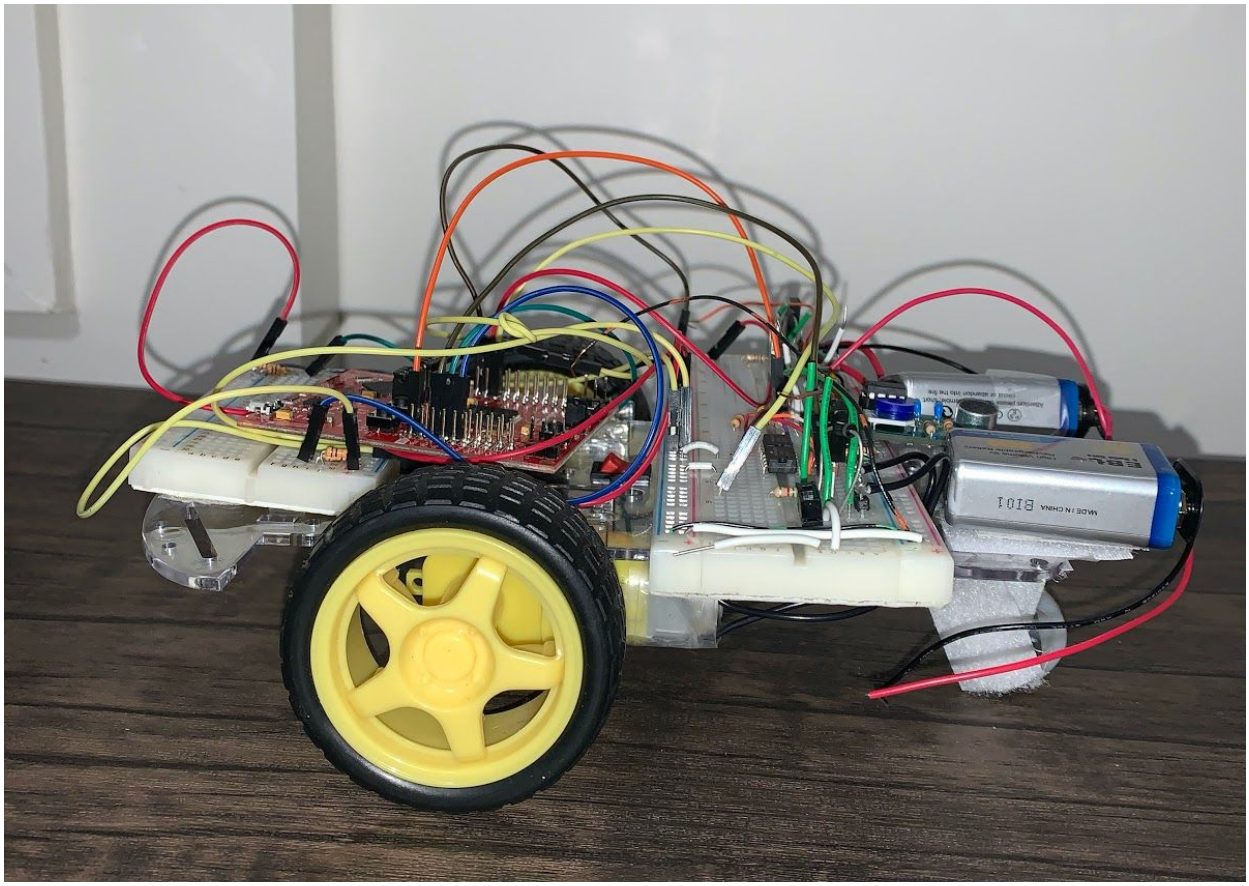
# SIXT33N PROJECT REPORT

*General; Front-end circuit; PCA classification; Control*



Yuchen Fu

Junliang Lu

Tianyi Guan

EE16b, Fall 2019

Professor Sahai

# GENERAL

In order to build a voiced-controlled car that will go straight, turn left, and turn right according to our voice instruction, we went through the following process:

1. Installed a microphone that converts auditory signals into voltage signals.
2. Completed the circuits with a breadboard, motors, filters, etc… We adjusted the DC filter and  bandpass filter so that regulates the input voltage and filter out noise at certain frequencies.
3. Wrote code to implement closed-loop control so that our launchpad picks up and analyzed the distance the wheels traveled, and output desired power to the two motors to make sure the car can go straight.
4. Used SVD/PCA to analyze the accuracy of our four chosen words.
5. Wrote code to implement open-loop control so now our car can go straight, turn left, and turn right according to our command.
6. Integrated all parts together so our car drives differently responding to our four chosen words.

The whole process of building this car only takes six bullet points in this report, but the truth is that we spent about 5 hours per week (we had to arrive in the lab hours before our regular lab started in order to finish that week's task in time), and ~15 weeks for this single project. We are glad that after a semester of hard word, our car finally works as expected.

The success does not just come from our commitment, but also depends on the friendly and collaborating atmosphere our lab section, and the course in general have. It is admitted that Professor Sahai has come up with lots of policies to generate such studying environment and he carefully managed to convert the knowledge taught in class to the practice in labs.

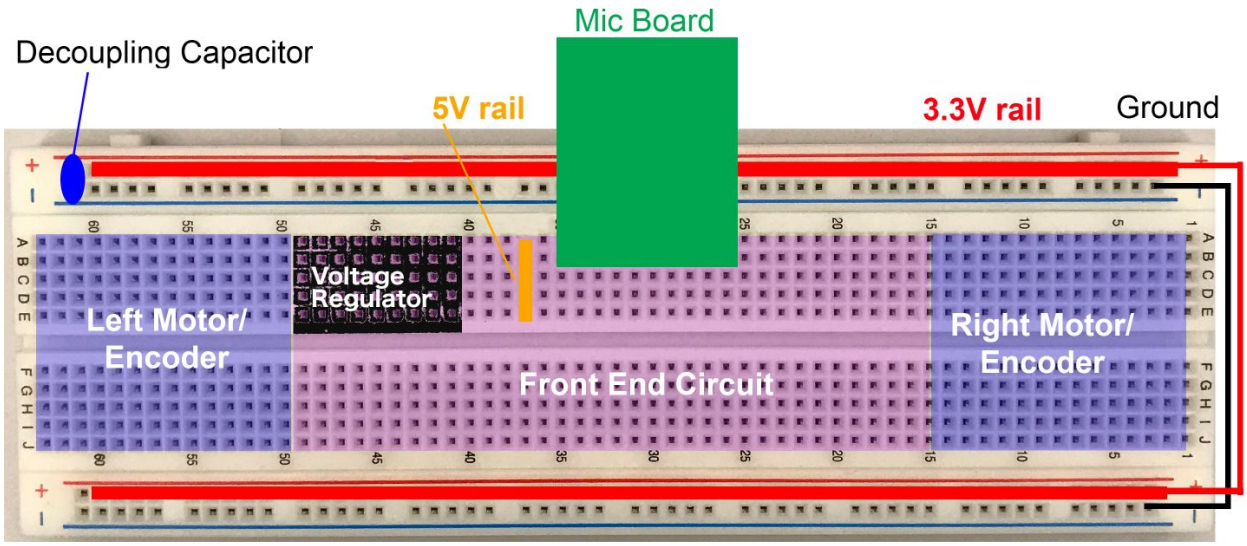
At last, we really want to thank our lab's (Wednesday 2-5pm) head TA, Maggie Payne, who always takes care of her students. Other TAs have been really helpful as well. For example, Victor Lee, Kaitlyn Chan, and Meera Lester helped us with debugging countless times. We could have never finished the project and learned so much without any of them.

# Front-End Circuit

- ## MATERIALS

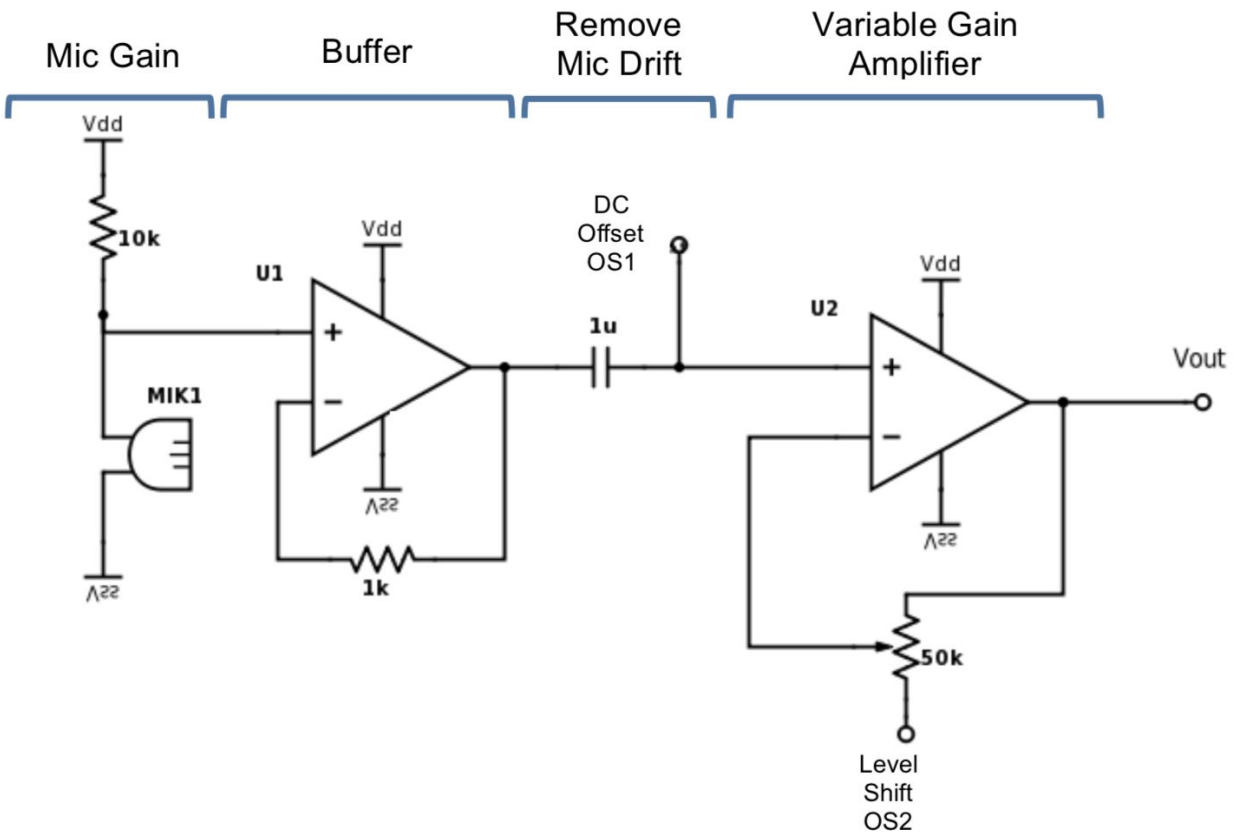
1. Power Supply
2. Mic Board
3. Microphone Gain
4. Buffer
5. Filters

The following diagram summarizes how we put the parts together:

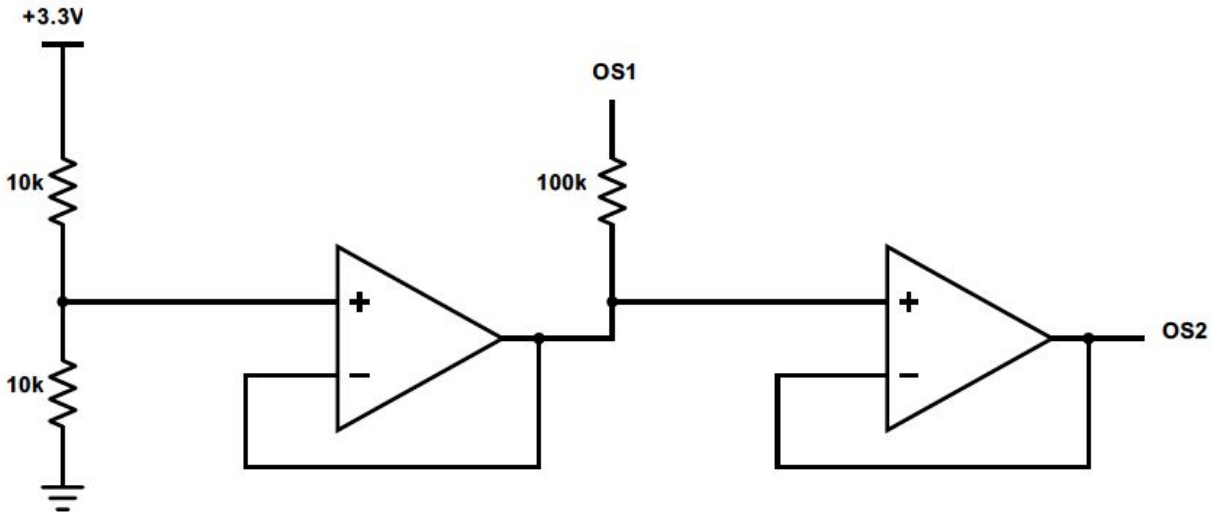



Frist, we need Voltage-Regulator LM7805 to convert 9V to 5V, and LD1117 to convert 9V to 3.3V. The 5V is for launchpad and mic board, and the 3.3V is for breadboard signals. 9V is used because our main power supply will be two 9V batteries, one for Front End Circuit and one for both motors.

The Mic Board is using the following schematic, which shows that it is converting sound to voltage signals. Since DC Offset shouldn't be amplified with the non-inverting amplifier, the mic DC offset needs to be removed in the mic drift phase and being added at the end.




Therefore, we need the following biasing circuit and connect OS1 and OS2 to the above nodes, the high pass filter is then connected to OS2 to keep the signal centered at 1.65V:



Another important thing is, we need to filter out human voices from voices that are not from humans.  That is why we need to build a high pass filter.  Here is the resistor and capacitor that we used and the cutoff frequency we have:

High-pass filter:

==========

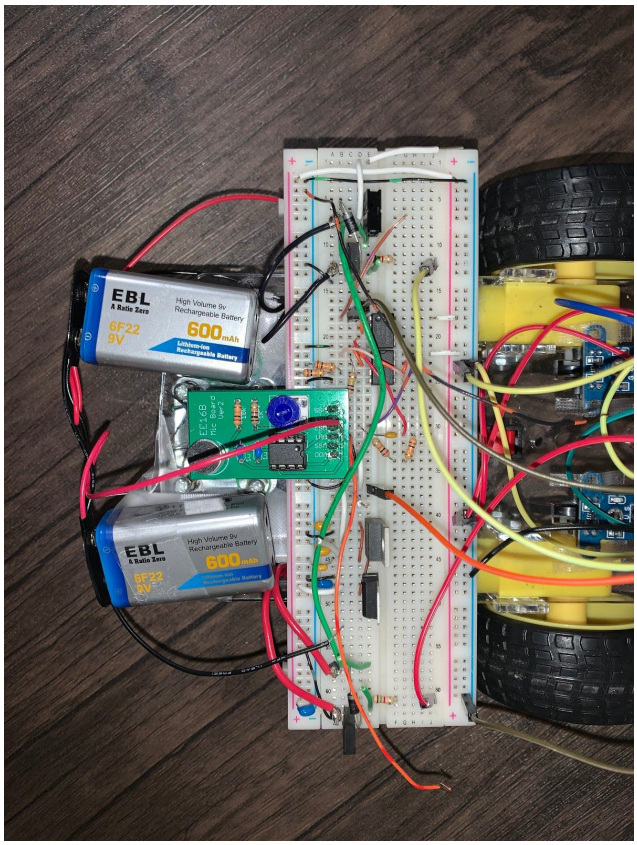
Resistor: 30000 ohms

Capacitor: 1e-07 farads

Cutoff frequency: 53.05164769729845 Hz

After that, tuning the microphone's gain by turning the potentiometer until its output is centered around 1.65V and ranges from 0 to 3.3V to prevent damage to the launchpad.

Finally, after constructing the motor circuit and connects to both motors, that finishes building the Front-End Circuit.  The following is a picture of the car:

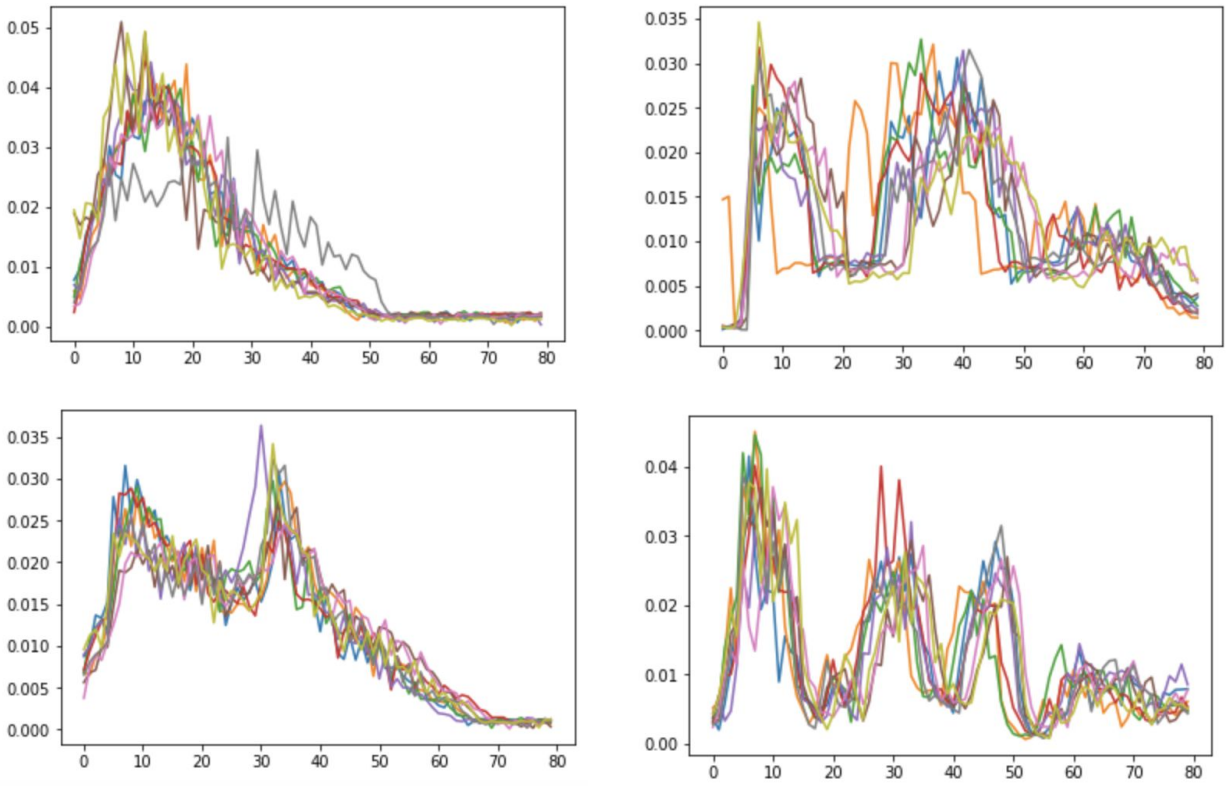


# PCA Classification

- ## Commands

1. Hi (Go straight far)
2. Luna (Turn left)
3. Banana (Go straight short)
4. Contradiction (Turn right)

See the shapes of those above four words:

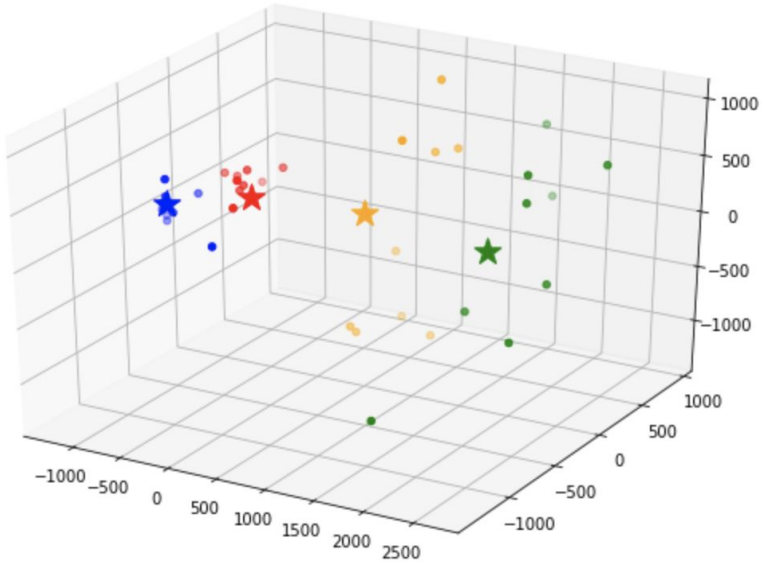



Initially, we chose different words set: New, New York, Berkeley, Contradiction, Normalization, pronounced by our partner Guan. The idea was that each of the words had a unique syllables. However, after applying PCA via SVD to find the clusters, we found that contradiction and normalization overlapped a lot. It made sense as those two vocabulary resembles. As a result, we deleted normalization. However, although the overlap of clusters appeared to be reduced by a significant amount, our accuracy is still lower than 80%. We only succeeded after changing three words (New to Hi, New York to

Luna, Berkeley to Banana) pronounced by a different partner, Fu. The reason why we switched to luna and banana is because “u” and “a” are both vowels, so they have a strong and consistent emphasize, which helps to distinguish between them.

## Results

Indeed, now we get very compact and unique clusters (though the old word contradiction is still a bit spread out), as shown below:



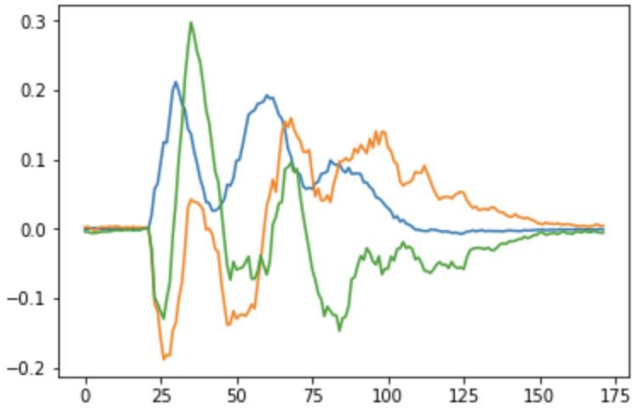

with accuracy as following:

```
Percent correct of word 1 = 88.88888888888889%
Percent correct of word 2 = 100.0%
Percent correct of word 3 = 100.0%
Percent correct of word 4 = 88.88888888888889%
```

## Process

1. After reading the data into our directory, we had to align audio recordings. This is because we may start to pronounce the word a bit after the red light blinks (indicating that it starts recording), so we cut off the part where the data is noises. We achieved this by setting *length* and *threshold* variables relative to the maximum value of the data.

2. We then generated and preprocessed PCA matrix by stacking the four words together. We also demeaned the matrix because we want to center the data around the mean to eliminate the intensity fluctuations (our speaking might vary).
3. To save memory and ease computation, we picked the first four principal (with the biggest sigma values, meaning the biggest variance) components to represent the whole data set. See the following plot to see that the four lines vary enough:



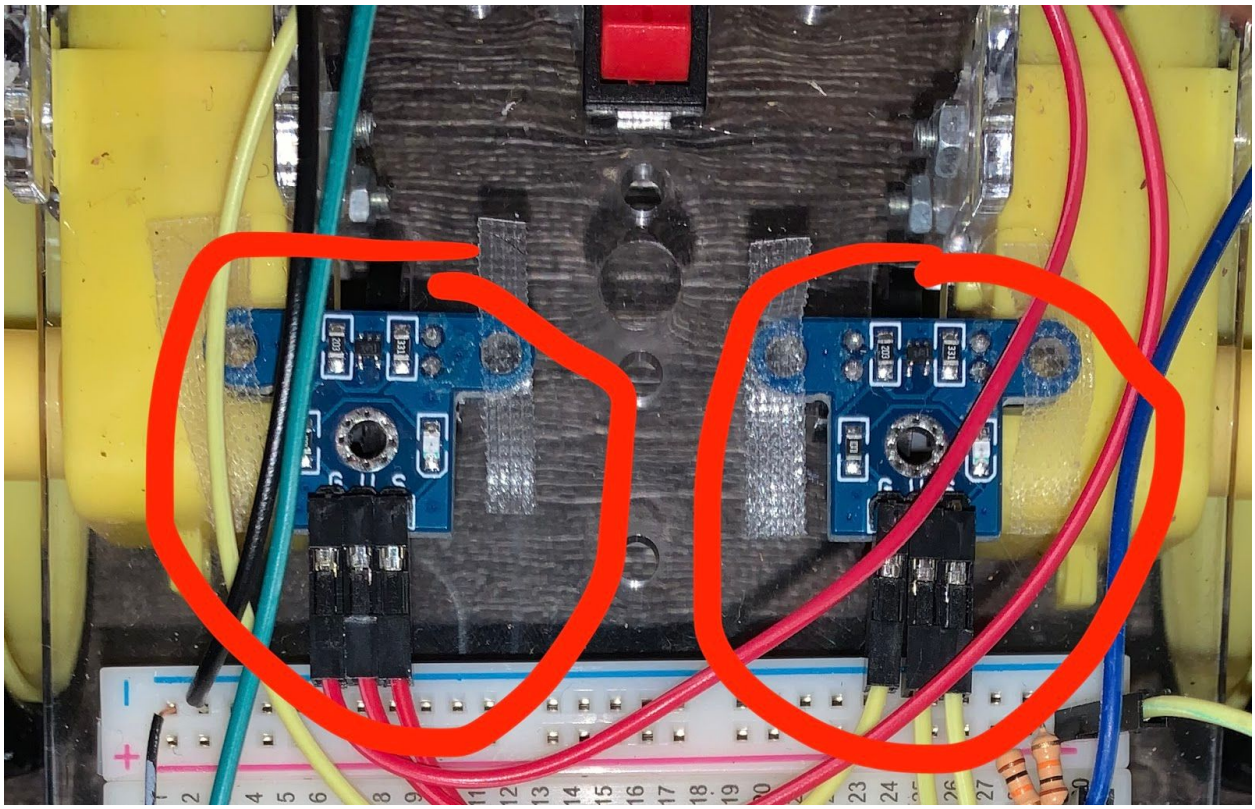

4. We wrote code of find_centroids function to find clustering data points. The results were shown above.
5. We then did the same preprocessing to the test data
6. At last, we tested the classifier and find the accuracy. The results were shown above.

# Control

Unfortunately, even though we can give exact same voltage to the two DC motors, they will still perform differently so that the performance of the car (to drive straight forward or turning at certain degree) will be greatly influenced. In order to deal with that and increase the car's performance, we implemented a feedback close-loop control system to adjust voltage we give to the two motors according to their actual performance (rotary speed).

## • Hardware Implementation

In order to accomplish the close-loop control system, we need to know the two motors real performances at the moment, so we need sensors.



The two encoders with two black plastic wheels can work as photointerrupter modules, giving us data reflecting the real time motor performance.

- ## Training Data Collection & Procession

In order to control our system better, we need to know how the system is performing now. So we collect data of motor performance and our input the the motor, running  Least-Squares Regression on it, and get the value we need to finish our controlling algorithm.  To get perfect data with little interruptions, we go to the parking lot where is more spacious. The screenshots show exactly how we did it (they are  a bit small to fit in the page, zoom in if needed):

```
# YOUR CODE HERE
# Write a function that formulates and performs least squares
# and returns theta and beta for one wheel
def identify_parameters(u, v):
    # Returns (theta, beta) for given (u, v)
    neg_ones = np.full((u.shape[0], 1), -1)
    A = np.column_stack((u, neg_ones))
    return np.linalg.lstsq(A, v)[0]

# extract parameters, call your function here
theta_left, beta_left = identify_parameters(u, vleft) # 0, 0
theta_right, beta_right = identify_parameters(u, vright) # 0, 0

print("float theta_left = {:.4g};".format(theta_left))
print("float theta_right = {:.4g};".format(theta_right))
print("float beta_left = {:.4g};".format(beta_left))
print("float beta_right = {:.4g};".format(beta_right))
```

```
float theta_left = 0.4828;
float theta_right = 0.4836;
float beta_left = 3.839;
float beta_right = 5.457;
```

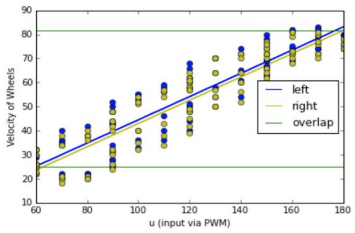
```
In [22]: min_vel = max(min(vleft_LS), min(vright_LS))
         max_vel = min(max(vleft_LS), max(vright_LS))
         print('Velocity range = [{:0.1f}, {:0.1f}]'.format(min_vel, max_vel))
         midpoint = (min_vel+max_vel)/2
         print('\nOperating point:\nfloat v_star = {:.1f};'.format(midpoint))

         u = u.reshape(-1)
         vleft_LS = theta_left*u-beta_left
         vright_LS = theta_right*u-beta_right
         plt.plot(u, vleft_LS, 'b-', u, vright_LS, 'y-')
         for i in (min_vel, max_vel):
             plt.plot(u, 0*u + i, 'g-')
         plt.plot(u, vleft, 'bo',  u, vright, 'yo')
         plt.xlabel("u (input via PWM)")
         plt.ylabel("Velocity of Wheels")
         plt.legend(("left", "right", "overlap"), loc=0)
```

```
Velocity range = [25.1, 81.6]

Operating point:
float v_star = 53.4;
```

```
Out[22]: <matplotlib.legend.Legend at 0x814a550>
```




- Algorithm

$$v_L[k] = d_L[k+1] - d_L[k] = \theta_L \left( \frac{v^* + \beta_L}{\theta_L} - k_L \frac{\delta[k]}{\theta_L} \right) - \beta_L$$

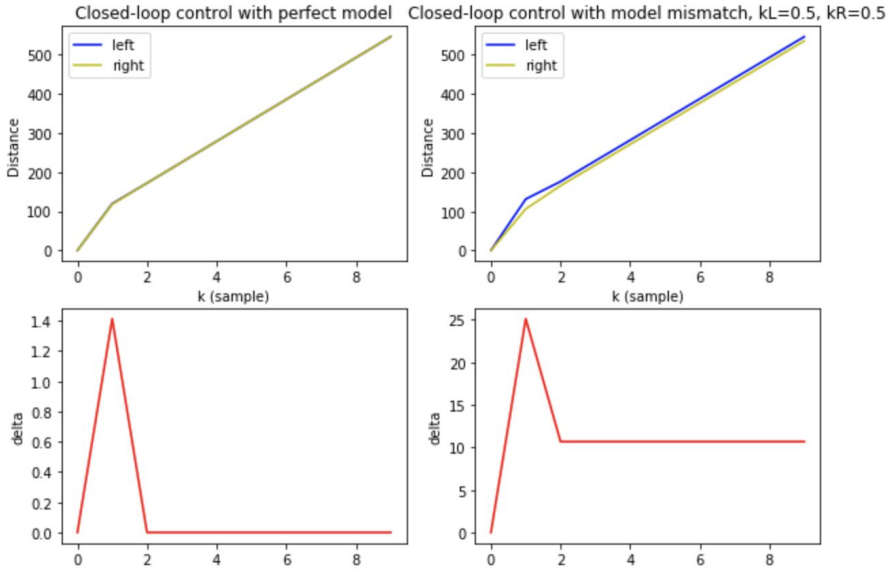
$$v_R[k] = d_R[k+1] - d_R[k] = \theta_R \left( \frac{v^* + \beta_R}{\theta_R} + k_R \frac{\delta[k]}{\theta_R} \right) - \beta_R$$

This is the formula we used to complete the software implementation.  With the value we already calculated in the previous part, we just have one more task to

do: choose the K value.

With jupyter notebook stimulation, we find that the whole close-loop control system is stable as long as Ks are smaller than 1. So we choose .5 and .5 which seems to perform best.

```
Steady state error = 0.00
Steady state error = 10.68
kL=0.5, kR=0.5
Eigenvalue of system: 1-kL-kR=0; stable, corrects error in one step
```



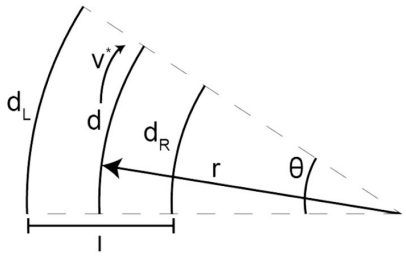

- ## Turning

To achieve a perfect turn, we need to calculate more things to adjust our input to the system. This task needs a bit of physics knowledge,  and this is how we do it:

Use the following variables:

- $k$ - time-step
- $r$ [cm] - turn radius of the center of the car; 1 cm ~ 1 encoder tick
- $d$ [ticks] - distance traveled by the center of the car
- $l$ [cm] - distance between the centers of the wheels; 1 cm ~ 1 encoder tick
- $\omega$ [rad/tick] - angular velocity
- $\theta$ [rad] - angle traveled

Inspect the following diagram:




From this geometry, can you write $\delta[k]$ in the following form?

$$\delta[k] = f(r, v^*, l, k)$$

****Check your answer with the lab staff!****

Answer:(v* k l) /r

$(v^* * k * l) /r$

So we can control our car to turn at certain radius and angle.