In [248]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [249]:

```
stationary = " Lidar 1 range(mm): 154  Lidar 2 range(mm): 132 5835 Heading: 5.32 Rad

FD_backward = ' 5854  Heading: 5.31 Radians    304.04 Degrees     Magnetometer readi

stop = ' 5864  Heading: 5.39 Radians    308.75 Degrees     Magnetometer readings: Mx

BK_forward = ' 5866  Heading: 5.36 Radians    306.94 Degrees     Magnetometer readi

zero_stop = ' 5877 Heading: 5.32 Radians    305.07 Degrees     Magnetometer readings

LT_right = ' 5878  Heading: 5.32 Radians    304.56 Degrees     Magnetometer readings


# On connection #0
#   got text: #0
zero_sero_stop = ' 5889  Heading: 5.15 Radians    294.91 Degrees     Magnetometer re

zero_dr = ' 5891  Heading: 5.13 Radians    293.86 Degrees     Magnetometer readings

stop3 = ' 5895  Heading: 5.12 Radians    293.57 Degrees     Magnetometer readings: M

RT_left =' 5898  Heading: 5.11 Radians    292.64 Degrees     Magnetometer readings:

stop4 = ' 5899  Heading: 5.12 Radians    293.50 Degrees     Magnetometer readings: M

RT_left2 = ' 5900  Heading: 5.19 Radians    297.25 Degrees     Magnetometer readings

stop5 =' 5923  Heading: 5.24 Radians    300.47 Degrees     Magnetometer readings: Mx
```

In [252]:

```
import re
allStrings = stationary + FD_backward + stop + BK_forward + zero_stop + LT_right + z

lidar_1 = re.findall("Lidar 1 range\(mm\): \d+", allStrings)
lid_1 = []
for r in lidar_1:
    lid_1 += [re.findall('\d\d+', r)]
lid_1

lidar_2 = re.findall("Lidar 2 range\(mm\): \d+", allStrings)
lid_2 = []
for r in lidar_2:
    lid_2 += [re.findall('\d\d+', r)]


len(lid_1) == len(lid_2)
```

Out[252]:

```
True
```

In [253]:

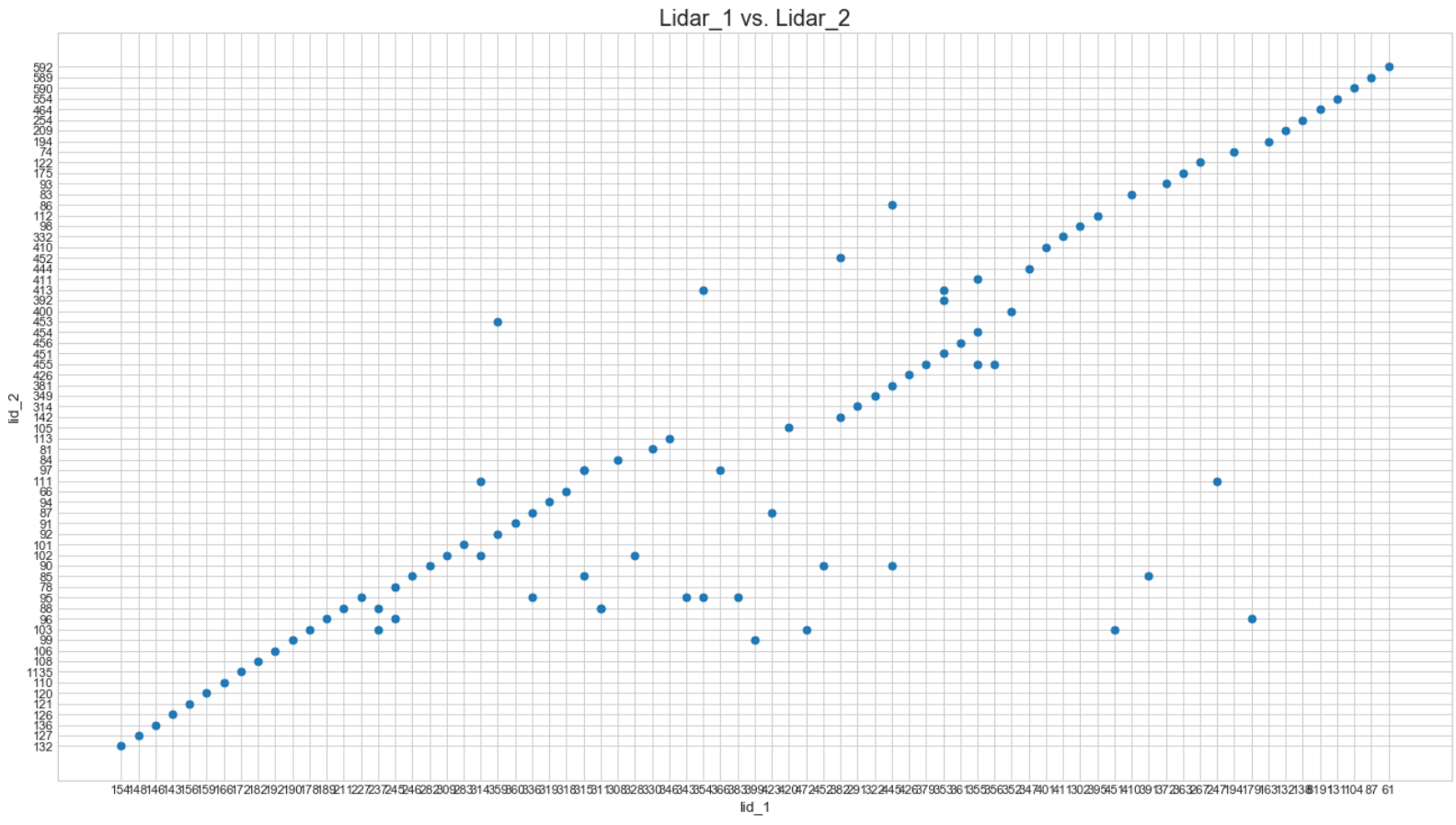
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

In [254]:

```
#  DATA EXPLORATION

plt.figure(figsize=(20, 11), dpi= 80, facecolor='w', edgecolor='k')

x = [lid_1[i][0] for i in range(len(lid_1))]
y = [lid_2[i][0] for i in range(len(lid_1))]
plt.scatter(x, y, marker='o')
plt.title("Lidar_1 vs. Lidar_2", fontsize=18)
plt.xlabel("lid_1", fontsize=12)
plt.ylabel("lid_2", fontsize=12)
```

Out[254]:

```
Text(0,0.5,'lid_2')
```

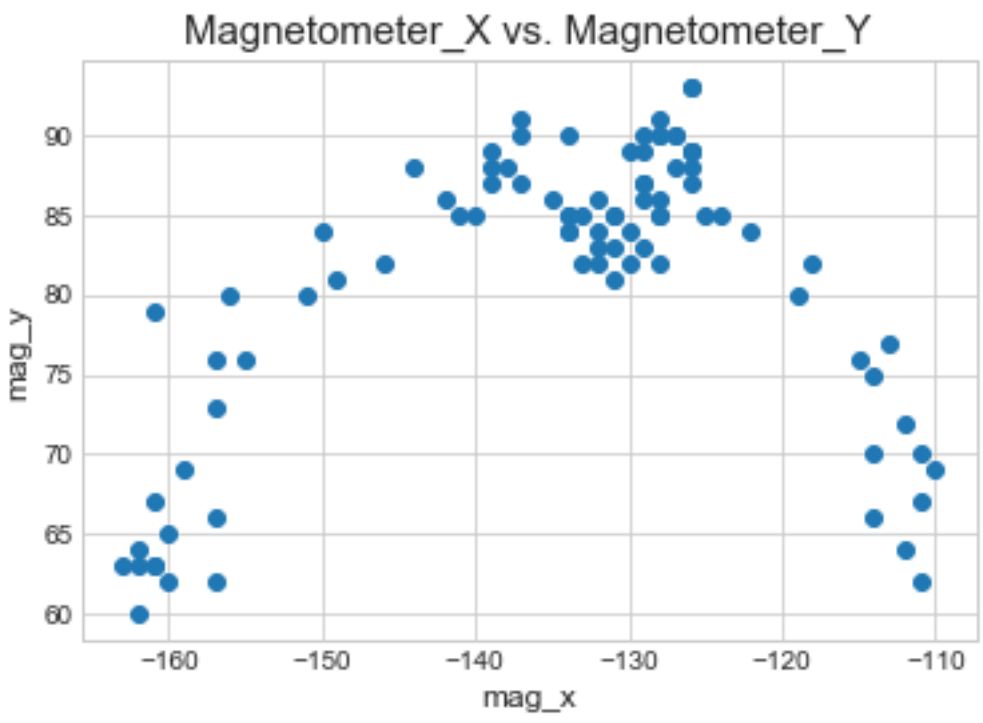


In [255]:

```
#  DRIVING FW --> BK --> LEFT --> RIGHT

m_x = re.findall('Magnetometer readings: Mx:\S*\d+', allStrings)
mag_x = []
for r in m_x:
    mag_x += [re.findall('\-*\d+', r)]


m_y = re.findall('My:\S*\d+', allStrings)
mag_y = []
for r in m_y:
    mag_y += [re.findall('\-*\d+', r)]
plt.scatter(mag_x, mag_y, marker='o')
plt.title("Magnetometer_X vs. Magnetometer_Y", fontsize=15)
plt.xlabel("mag_x", fontsize=12)
plt.ylabel("mag_y", fontsize=12)
```

Out[255]:

```
Text(0,0.5,'mag_y')
```



In [256]:

```
# WILL NEED THIS IN THE FUTURE FOR CALIBRATION

x_center = (min(mag_x)+max(mag_x))
y_center = (min(mag_y)+max(mag_y))

x_center, y_center
```

Out[256]:

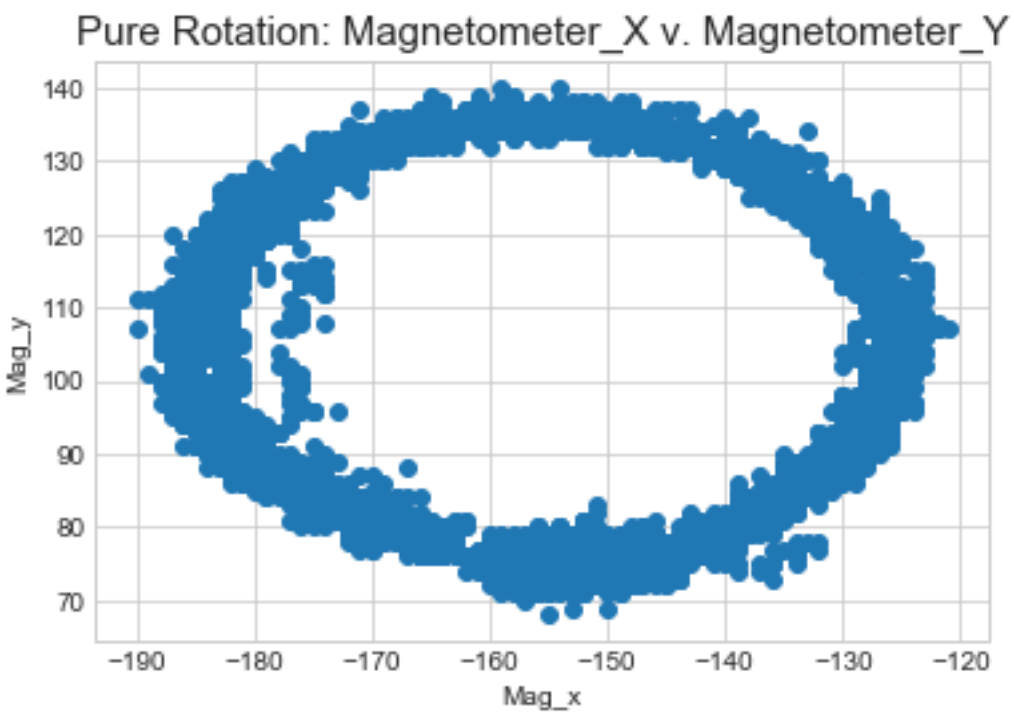
```
(['-110', '-163'], ['60', '93'])
```

In [260]:

```
# SLOWLY ROTATING 10 degrees at a time - full circle
df1 = pd.read_csv("IMU Data copy.csv")

mag_x = df1.iloc[:, 1]
mag_y = df1.iloc[:, 2]
plt.scatter(mag_x, mag_y, marker='o')
plt.title("Pure Rotation: Magnetometer_X v. Magnetometer_Y", fontsize=15)
plt.xlabel("Mag_x")
plt.ylabel("Mag_y")
```

Out[260]:

```
Text(0,0.5,'Mag_y')
```



In [9]:

```
Q = df1.iloc[:, 0]
angle = []
for row in Q:
    angle += [re.sub("\d+\\t", '', row)]
Q.head()
```

Out[9]:

```
0    1\t300.11
1    2\t299.18
2    3\t299.18
3    4\t298.27
4    5\t298.57
Name: 0\t298.72, dtype: object
```

In [10]:

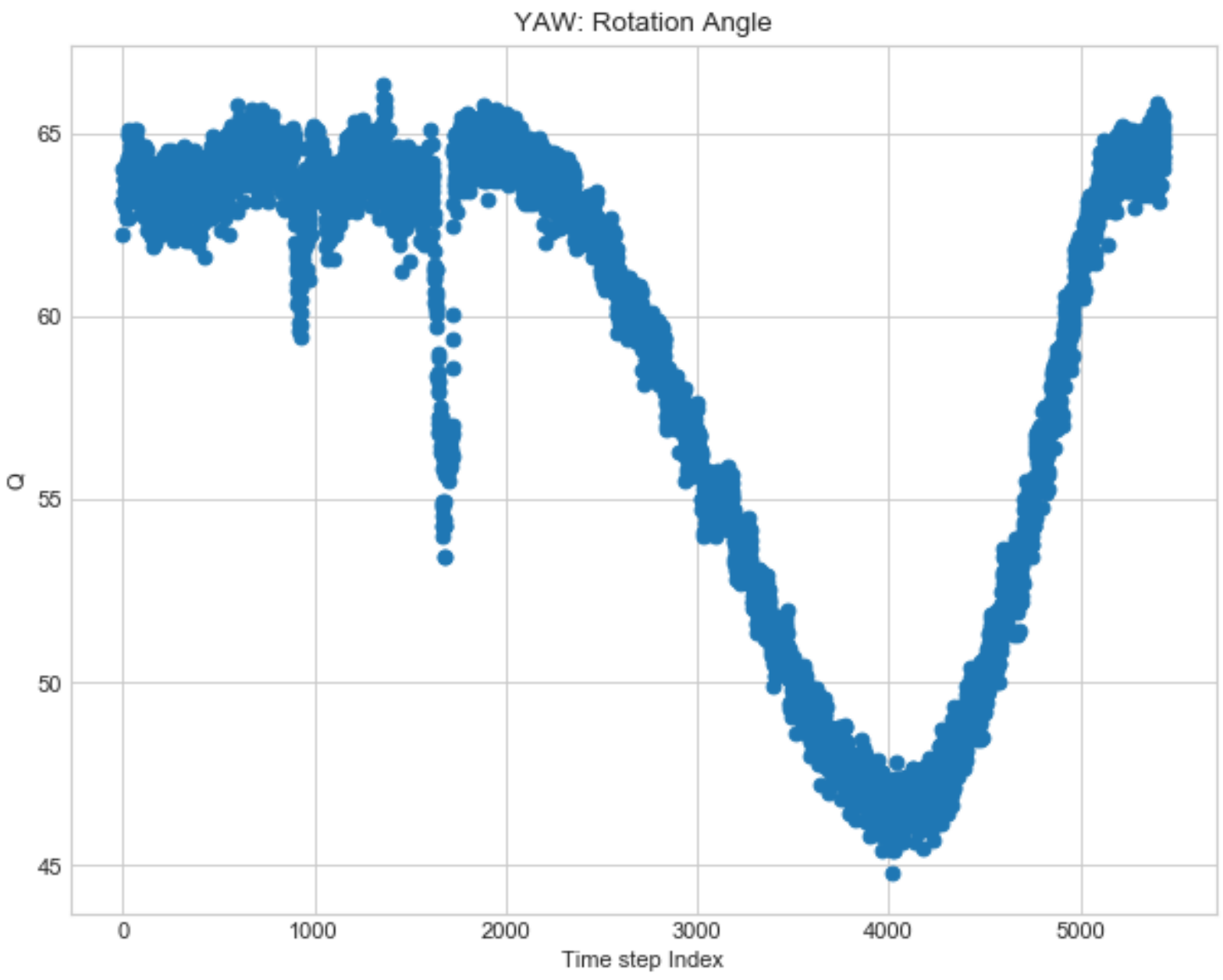
```
mag_x, mag_y = mag_y, mag_x
```

In [237]:

```
M_PI = 3.1415

yaw = 180*np.arctan2(-mag_y,mag_x)/M_PI
plt.figure(figsize=(9, 7), dpi= 80, facecolor='w', edgecolor='k')
plt.scatter(range(len(yaw)), yaw, marker='o')
plt.title("YAW: Rotation Angle")
plt.xlabel("Time step Index")
plt.ylabel("Q")
```

Out[237]:

```
Text(0,0.5,'Q')
```



"Open Loop Data Collection and Parameter Estimation"

Before trying to control the ESP8266-12E, we will first characterize the open loop reponse of the system. To make modeling easier, we will separate the models of the two wheels. The model that we create for both wheels can be identical because they are symmetric, but will have different parameters to account for subtle differences in the hardware. The distance that each wheel has moved (measured in number of TIME STEPS) at

sample will be denoted using and for the left and right wheels respectively. The open loop model has an input for PWM level and an output for the distance for that wheel (thus the matrix is just the identity matrix).

All distances are measured in cm (each TIME STEP corresponds to ~1cm of wheel travel) while 1 TIME STEP = 1 ms. As you can see, there are 2 unknown variables for each wheel, namely and . To determine these parameters, we will perform least squares regression on each row. We will do this for both wheels separately.

Sensor Model for Constant Velocity

By assuming constant velocity we get:

In [12]:

```
# How to use magnetometer: https://www.diva-portal.org/smash/get/diva2:606532/FULLTE

def rotation_matrix(yaw, index):
    heading = yaw[index]
    Rotat_M = [[np.cos(heading), np.sin(heading), 0],
               [np.sin(heading), np.cos(heading), 0],
               [0, 0, 1]]
    return Rotat_M
```

In [236]:

```
old_velocities = pd.read_csv("Lab 1 Data - Angular Velocities.csv")
old_velocities.head()
```

Out[236]:

	Left Motor	Unnamed: 1	Unnamed: 2	Unnamed: 3	Right Motor	Unnamed: 5	Unnamed: 6
0	PWM	Frequency	Period	NaN	PWM	Frequency	Period
1	0	1.2	0.83	NaN	0	1.3	0.77
2	10	1.2	0.83	NaN	10	1.3	0.77
3	20	1.2	0.83	NaN	20	1.3	0.77
4	30	1.2	0.83	NaN	30	1.3	0.77

In [261]:

```
#  DATA CLEANING

old_velocities = old_velocities.drop('Unnamed: 3', axis=1)
old_velocities = old_velocities.rename(columns={"Left Motor": "Left Motor PWM",
                                                "Right Motor": "Right Motor PWM",
                                                "Unnamed: 1": "Left Frequency",
                                                "Unnamed: 2": "Left Period (s)",
                                                "Unnamed: 5": "Right Frequency",
                                                "Unnamed: 6": "Right Period (s)"})
old_velocities = old_velocities.drop(old_velocities.index[[0]])
old_velocities.reset_index(drop=True, inplace=True, col_level=0, col_fill='')
old_velocities
```

Out[261]:

	Left Motor PWM	Left Frequency	Left Period (s)	Right Motor PWM	Right Frequency	Right Period (s)
0	0	1.2	0.83	0	1.3	0.77
1	10	1.2	0.83	10	1.3	0.77
2	20	1.2	0.83	20	1.3	0.77
3	30	1.2	0.83	30	1.3	0.77
4	40	1.2	0.83	40	1.3	0.77
5	50	1.2	0.83	50	1.3	0.77
6	60	1.2	0.83	60	1.2	0.83
7	70	1.15	0.87	70	1.05	0.95
8	75	0.9	1.11	75	0.8	1.25

In [262]:

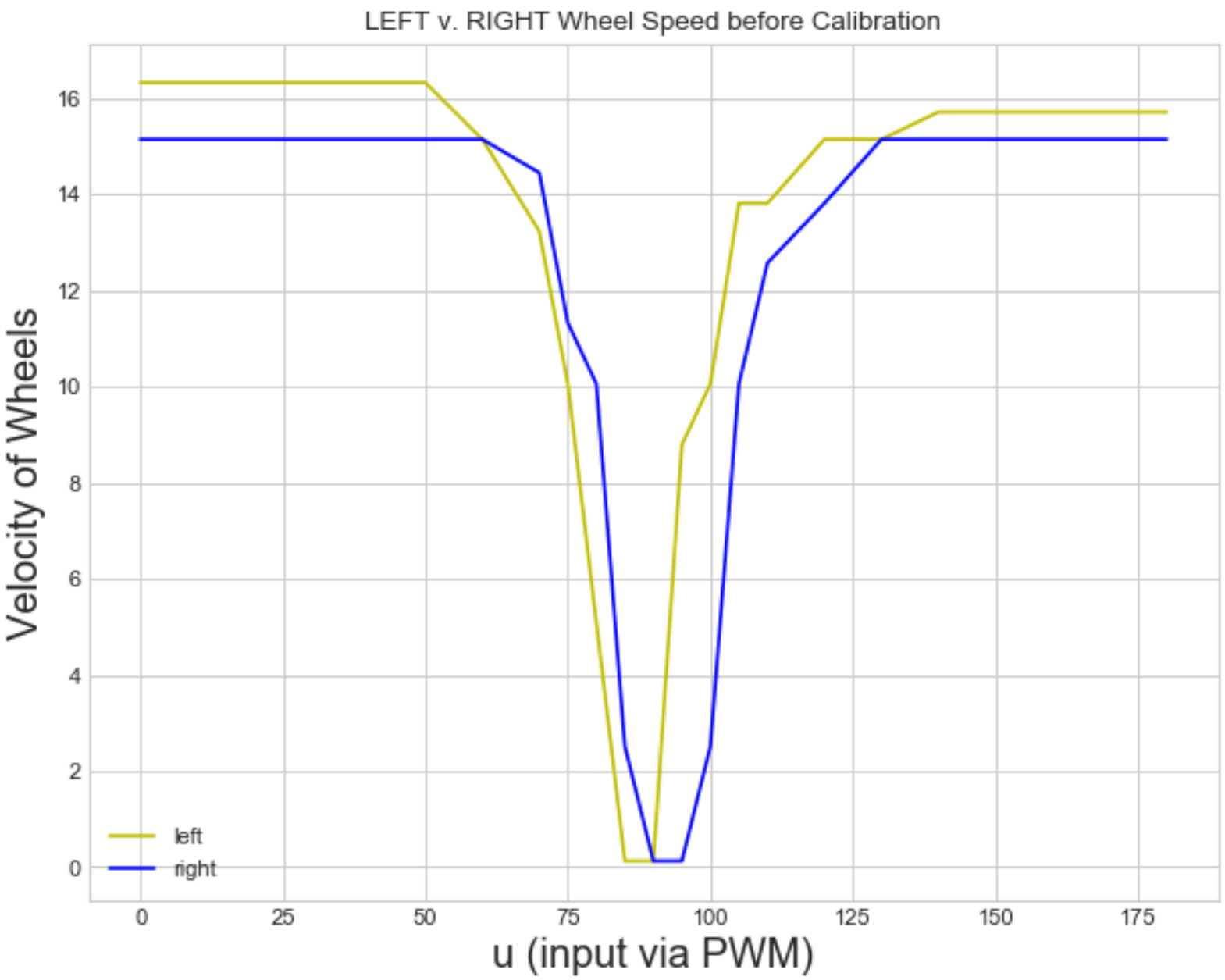
```
# pi * r^2 = circumference
# pi * r^2 / time = speed
pi = M_PI
r = 2 #cm


period_left = old_velocities["Left Period (s)"][:]
velocity_left = [3.1415 * r*r / float(p) for p in period_left]
pwm_left = [float(p) for p in old_velocities["Left Motor PWM"][:]]

period_right = old_velocities["Right Period (s)"][0:]
velocity_right = [3.1415 * r*r / float(p) for p in period_right]
pwm_right = [float(p) for p in old_velocities["Right Motor PWM"][:]]



t_f = [pwm_left[i]==pwm_right[i] for i in range(len(pwm_left))]
t_f2 = [velocity_left[i]== velocity_right[i] for i in range(len(pwm_left))]

old_velocities["Speed Left Wheel"], old_velocities["Speed Right Wheel"] = velocity_
old_velocities.head(10)
```

Out[262]:

	Left Motor PWM	Left Frequency	Left Period (s)	Right Motor PWM	Right Frequency	Right Period (s)	Speed Left Wheel	Speed Right Wheel
0	0	1.2	0.83	0	1.3	0.77	15.139759	16.319481
1	10	1.2	0.83	10	1.3	0.77	15.139759	16.319481
2	20	1.2	0.83	20	1.3	0.77	15.139759	16.319481
3	30	1.2	0.83	30	1.3	0.77	15.139759	16.319481
4	40	1.2	0.83	40	1.3	0.77	15.139759	16.319481
5	50	1.2	0.83	50	1.3	0.77	15.139759	16.319481
6	60	1.2	0.83	60	1.2	0.83	15.139759	15.139759
7	70	1.15	0.87	70	1.05	0.95	14.443678	13.227368
8	75	0.9	1.11	75	0.8	1.25	11.320721	10.052800
9	80	0.8	1.25	85	0	99.00	10.052800	0.126929

In [263]:

```
plt.figure(figsize=(9, 7), dpi= 80, facecolor='w', edgecolor='k')
plt.xlabel("u (input via PWM)", fontsize=18)
plt.ylabel("Velocity of Wheels", fontsize=18)
labels = ("left", "right")
plt.plot(pwm_right, velocity_right, 'y-')
plt.plot(pwm_left, velocity_left, 'b-')
plt.legend(labels, loc=0)
plt.title("LEFT v. RIGHT Wheel Speed before Calibration")
plt.show()
```



Now that we have some data, we can try performing least squares regression. We use function `np.linalg.lstsq` here. The problem formulation is nearly the same for both wheels, so we can write a function that takes the data as parameters, performs least squares, and extracts the parameters.

Record the values of and for each wheel.

In [25]:

```
old_velocities.columns
```

Out[25]:

```
Index([u'Left Motor PWM', u'Left Frequency', u'Left Period (s)',
       u'Right Motor PWM', u'Right Frequency', u'Right Period (s)',
       u'Speed Left Wheel', u'Speed Right Wheel'],
      dtype='object')
```

In [216]:

```
from sklearn.model_selection import train_test_split

#  SPLIT DATA SELECTING RANDOM ROWS W/O REPLACEMENT
train, test = train_test_split(old_velocities, test_size=0.3)
```

In [217]:

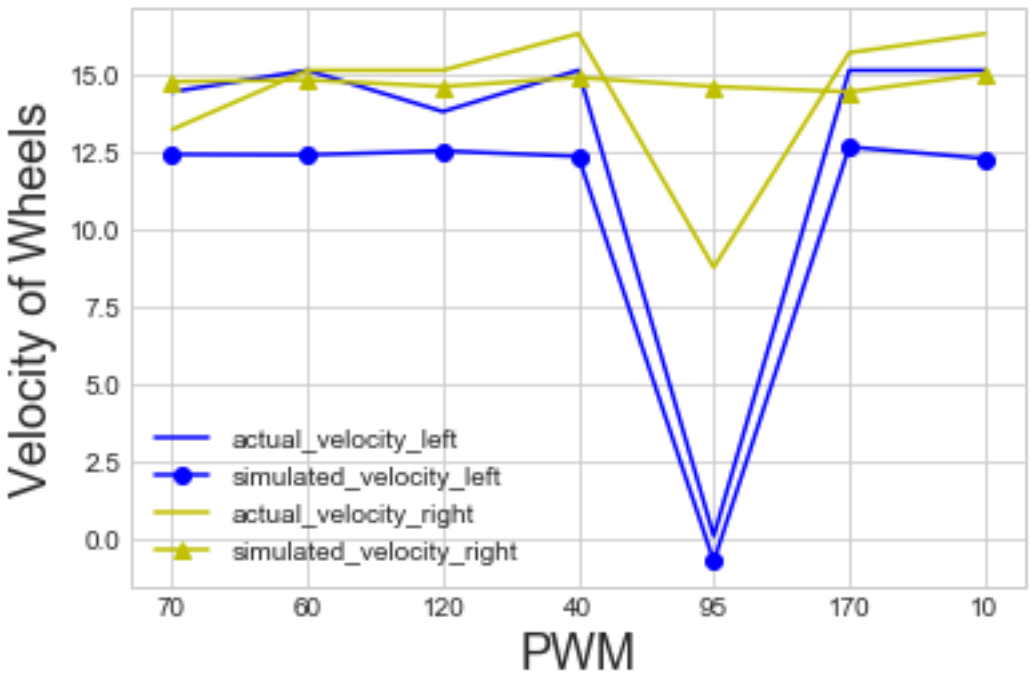
```
#  ACTUAL DATA
y_actual_left, y_actual_right = test["Speed Left Wheel"], test['Speed Right Wheel']

# SAME for actual vs. predicted
pwm_left, pwm_right = test['Left Motor PWM'], test['Right Motor PWM']
```

In [218]:

```
"""SIMPLER MODEL WITH 2 PARAMETERS"""

from sklearn.linear_model import LinearRegression
# TRAINE DATA FOR LEFT WHEEL
Lin_Regression_left = LinearRegression()
left_list = ['Left Motor PWM', 'Left Period (s)']
X_left = train.loc[:, left_list]
y_left = train["Speed Left Wheel"]
Lin_Regression_left.fit(X_left, y_left)
LinearRegression(copy_X=True, fit_intercept=True, normalize=False)
print("FOR THE LEFT WHEEL:", "Your coeffs: ", Lin_Regression_left.coef_, " And your

# TRAIN DATA FOR RIGHT WHEEL
Lin_Regression_right = LinearRegression()
right_list = ['Right Motor PWM', 'Right Period (s)']
# right_list = left_list
X_right = train.loc[:, right_list]
y_right = train["Speed Right Wheel"]
Lin_Regression_right.fit(X_right, y_right)
LinearRegression(copy_X=True, fit_intercept=True, normalize=False)
print("FOR THE RIGHT WHEEL:", "Your coeffs: ", Lin_Regression_right.coef_, " And you


#  PREDICTED TEST DATA
y_simulated_left = Lin_Regression_left.predict(test.loc[:, left_list])
```

```
y_simulated_right = Lin_Regression_right.predict(test.loc[:, right_list])
len(y_actual_left) == len(y_actual_right) == len(pwm_left) == len(pwm_right) == len(


plt.plot(pwm_left, y_actual_left, 'b-', label='actual_velocity_left')
plt.plot(pwm_left, y_simulated_left, 'b-o', label='simulated_velocity_left')
plt.plot(pwm_right, y_actual_right, 'y-',  label='actual_velocity_right')
plt.plot(pwm_right, y_simulated_right, 'y-^', label='simulated_velocity_right')
plt.xlabel("PWM", fontsize=18)
plt.ylabel("Velocity of Wheels", fontsize=18)
plt.legend()
plt.show()
```

```
('FOR THE LEFT WHEEL:', 'Your coeffs: ', array([ 0.00232447, -0.134247
73]), ' And your intercept is: ', 12.38499161407312)
('FOR THE RIGHT WHEEL:', 'Your coeffs: ', array([-0.0035133 , -0.14889
155]), ' And your intercept is: ', 15.162307502272094)
```



In [219]:

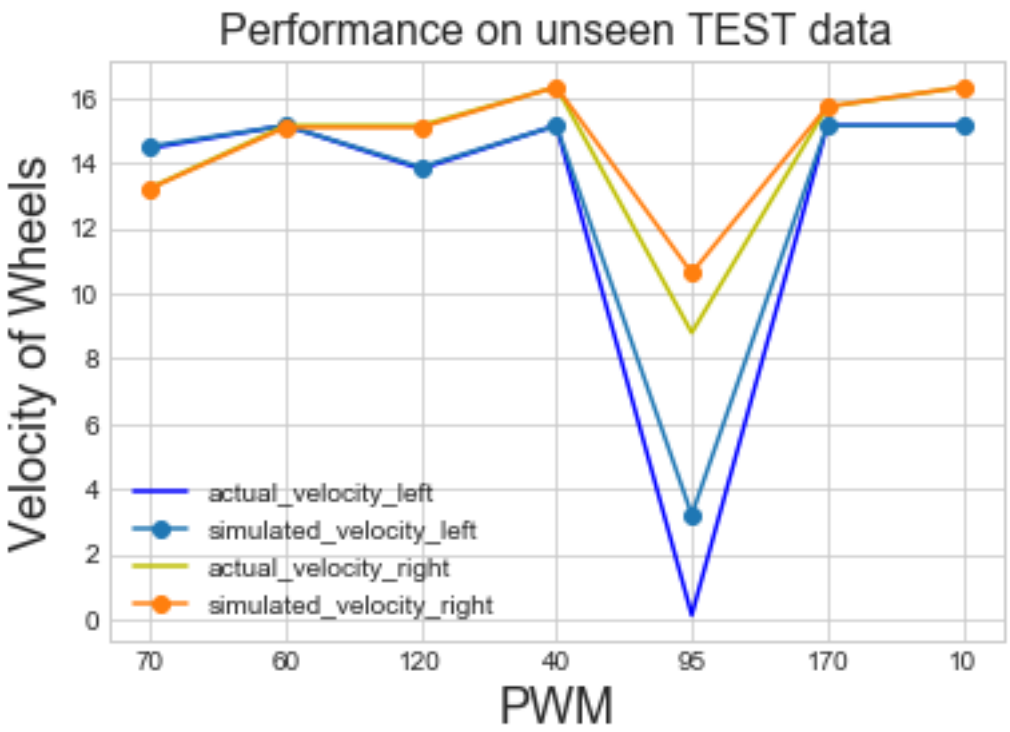
```
# Lin reg Example: https://stackoverflow.com/questions/19991445/run-an-ols-regressi

"""MORE COMPLEX MODEL - 6 parameters"""

# FOR LEFT WHEEL
Lin_Regression_left = LinearRegression()
full_list = ['Left Motor PWM', 'Left Frequency', 'Left Period (s)',
             'Right Motor PWM','Right Frequency','Right Period (s)']
X_left = train.loc[:, full_list]
y_left = train["Speed Left Wheel"]
Lin_Regression_left.fit(X_left, y_left)
print("FOR THE LEFT WHEEL:", "Your coeffs: ", Lin_Regression_left.coef_, " And your

# FOR RIGHT WHEEL
Lin_Regression_right = LinearRegression()
X_right = train.loc[:, full_list]
y_right = train["Speed Right Wheel"]
Lin_Regression_right.fit(X_right, y_right)

print("FOR THE RIGHT WHEEL:", "Your coeffs: ", Lin_Regression_right.coef_, " And you
```

```
('FOR THE LEFT WHEEL:', 'Your coeffs: ', array([-0.6165782 , 12.772639
38,  0.03455734,  0.61656314, -0.03262234,
       -0.03175093]), ' And your intercept is: ', -0.1495426129127732
7)
('FOR THE RIGHT WHEEL:', 'Your coeffs: ', array([-0.36280219,  0.12275
548,  0.01955708,  0.36293837, 12.45780666,
       -0.0181331 ]), ' And your intercept is: ', -0.026314786875015983
)
```

We can see that using only 2 inputs (i.e. second degree polynomial) gives bad predictions, so we made the model more complex by taking into account all available metrix.

In [243]:

```
#  PREDICTED DATA
y_simulated_left = Lin_Regression_left.predict(test.loc[:, full_list])
y_simulated_right = Lin_Regression_right.predict(test.loc[:, full_list])
len(y_actual_left) == len(y_actual_right) == len(pwm_left) == len(pwm_right) == len(


plt.plot(pwm_left, y_actual_left, 'b-', label='actual_velocity_left')
plt.plot(pwm_left, y_simulated_left, '-o', label='simulated_velocity_left')
plt.plot(pwm_right, y_actual_right, 'y-',  label='actual_velocity_right')
plt.plot(pwm_right, y_simulated_right, 'o-', label='simulated_velocity_right')
plt.xlabel("PWM", fontsize=18)
plt.ylabel("Velocity of Wheels", fontsize=18)
plt.title("Performance on unseen TEST data", fontsize=16)
plt.legend()
plt.show()
# plt.legend(labels, loc=0)
# plt.show()
```



When we take into account 6 measurements (with Period and Frequency being linearly dependent): 'Left Motor PWM', 'Left Frequency', 'Left Period (s)','Right Motor PWM','Right Frequency','Right Period (s)' we get really good prediction. Simulated and predicted speeds even have overlapping regions.

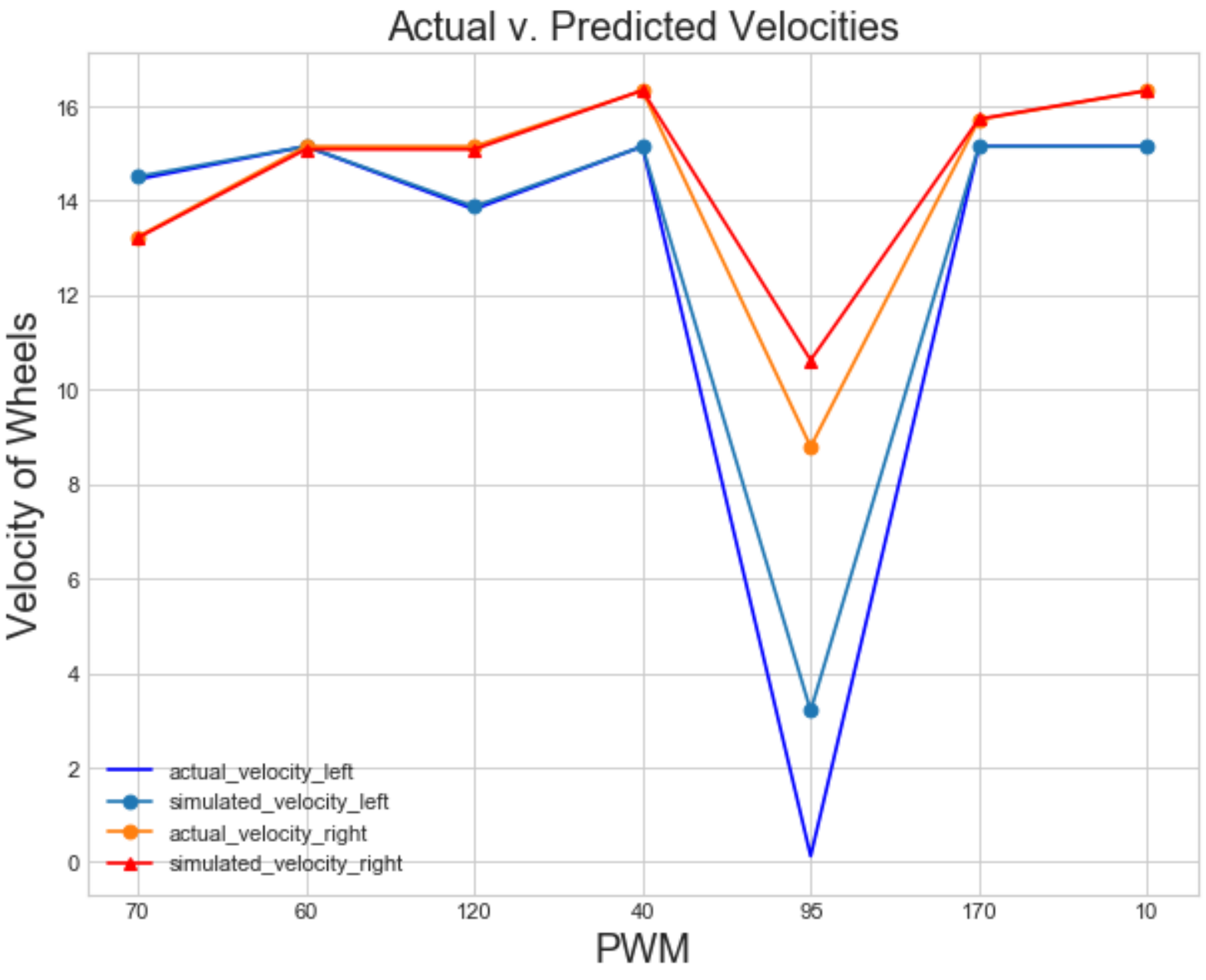
"""I split (randomly sample) the data @Yuci Shen and @Nathan March collected separately. And the final graph is model predicting velocities it never saw based on PWM, Frequency, and Period of left and right wheels ---> X has 6 columns and we have 2 models: one for Left Wheel and one for Right Wheel"""

** It is really interesting how having both frequency and period gives better prediction than when using period only.

*** BECAUSE WE USE RANDOM SPLITS FOR TRAIN AND TEST DATA - EVERYTIME THE PLOTS LOOK SLIGHTLY DIFFERENT***

In [245]:

```
"""I split (randomly sample) the data @Yuci Shen and @Nathan March collected separat
And the final graph is model predicting velocities it never saw based on PWM, Freque
of left and right wheels ---> X has 6 columns
and we have 2 models: one for Left Wheel and one for Right Wheel"""

y_actual_left, y_actual_right = test["Speed Left Wheel"], test['Speed Right Wheel']

plt.figure(figsize=(9, 7), dpi= 80, facecolor='w', edgecolor='k')
plt.plot(pwm_left, y_actual_left, 'b-', label='actual_velocity_left')
plt.plot(pwm_left, y_simulated_left, '-o', label='simulated_velocity_left')
plt.plot(pwm_right, y_actual_right, 'o-',  label='actual_velocity_right')
plt.plot(pwm_right, y_simulated_right, 'r-^', label='simulated_velocity_right')
plt.xlabel("PWM", fontsize=18)
plt.ylabel("Velocity of Wheels", fontsize=18)
plt.title("Actual v. Predicted Velocities", fontsize=18)
plt.legend()
plt.show()
```



It looks like on HIGH and LOW PWM values our model makes nearly perfect prediction of the wheels' velocities, while UNDERESTIMATING for LEFT wheel and OVERESTIMATING for RIGHT wheel velocities.

Now let's plot ERROR as L2 loss =

In [222]:

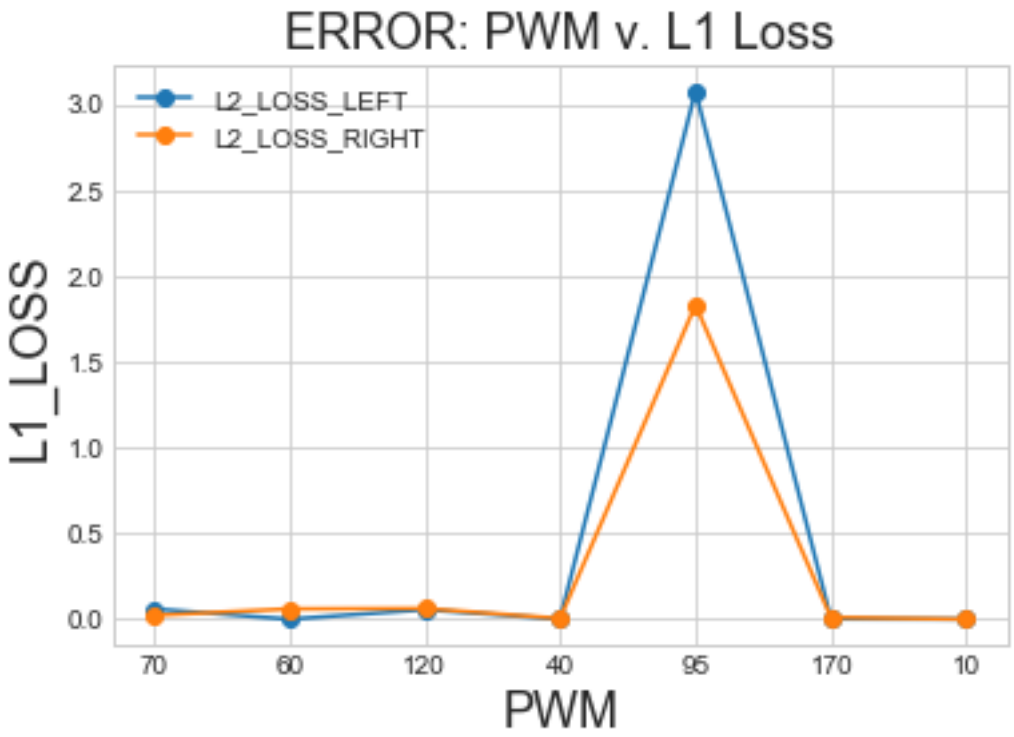
```
import math
y_actual_left.reset_index(drop=True, inplace=True)
y_actual_right.reset_index(drop=True, inplace=True)
diff_left = [y_actual_left[i]-y_simulated_left[i] for i in range(len(y_actual_right)
diff_right = [y_actual_right[i]-y_simulated_right[i] for i in range(len(y_actual_ri
L2_LOSS_LEFT = [math.sqrt(pow(diff_left[i], 2)) for i in range(len(y_actual_right))
L2_LOSS_RIGHT = [math.sqrt(pow(diff_right[i], 2)) for i in range(len(y_actual_right

plt.figure(figsize=(9, 7), dpi= 80, facecolor='w', edgecolor='k')
plt.plot(pwm_left, L2_LOSS_LEFT, 'o-', label='L2_LOSS_LEFT')
plt.plot(pwm_right, L2_LOSS_RIGHT, '-o', label='L2_LOSS_RIGHT')
plt.xlabel("PWM", fontsize=18)
plt.ylabel("L2_LOSS", fontsize=18)
plt.title("ERROR: PWM v. L2 Loss", fontsize=18)
plt.legend()
plt.show()

print("L2_LOSS_LEFT = ", sum(L2_LOSS_LEFT), "L2_LOSS_RIGHT = ", sum(L2_LOSS_RIGHT))
```


```
('L2_LOSS_LEFT = ', 3.1943584489037584, 'L2_LOSS_RIGHT = ', 1.99032047
17806116)
```

Now let's ALSO plot ERROR as L1 loss =

In [223]:

```
diff_left1 = [(y_actual_left[i]-y_simulated_left[i]) for i in range(len(y_actual_ri
diff_right1 = [(y_actual_right[i]-y_simulated_right[i]) for i in range(len(y_actual_
L1_LOSS_LEFT  = [np.absolute(diff_left1[i]) for i in range(len(y_actual_right))]
L1_LOSS_RIGHT = [np.absolute(diff_right1[i]) for i in range(len(y_actual_right))]

plt.plot(pwm_left, L1_LOSS_LEFT, 'o-', label='L2_LOSS_LEFT')
plt.plot(pwm_right, L1_LOSS_RIGHT, '-o', label='L2_LOSS_RIGHT')
plt.xlabel("PWM", fontsize=18)
plt.ylabel("L1_LOSS", fontsize=18)
plt.title("ERROR: PWM v. L1 Loss", fontsize=18)
plt.legend()
plt.show()

print("L1_LOSS_LEFT = ", sum(L1_LOSS_LEFT), "L1_LOSS_RIGHT = ", sum(L1_LOSS_RIGHT))
```




```
('L1_LOSS_LEFT = ', 3.1943584489037584, 'L1_LOSS_RIGHT = ', 1.99032047
17806116)
```

This could be because our model is simple

It looks like on HIGH and LOW PWM values our model makes nearly perfect prediction of the wheels' velocities, while UNDERESTIMATING for LEFT wheel and OVERESTIMATING for RIGHT wheel velocities.

Now let's plot ERROR as L2 loss =

In [224]:

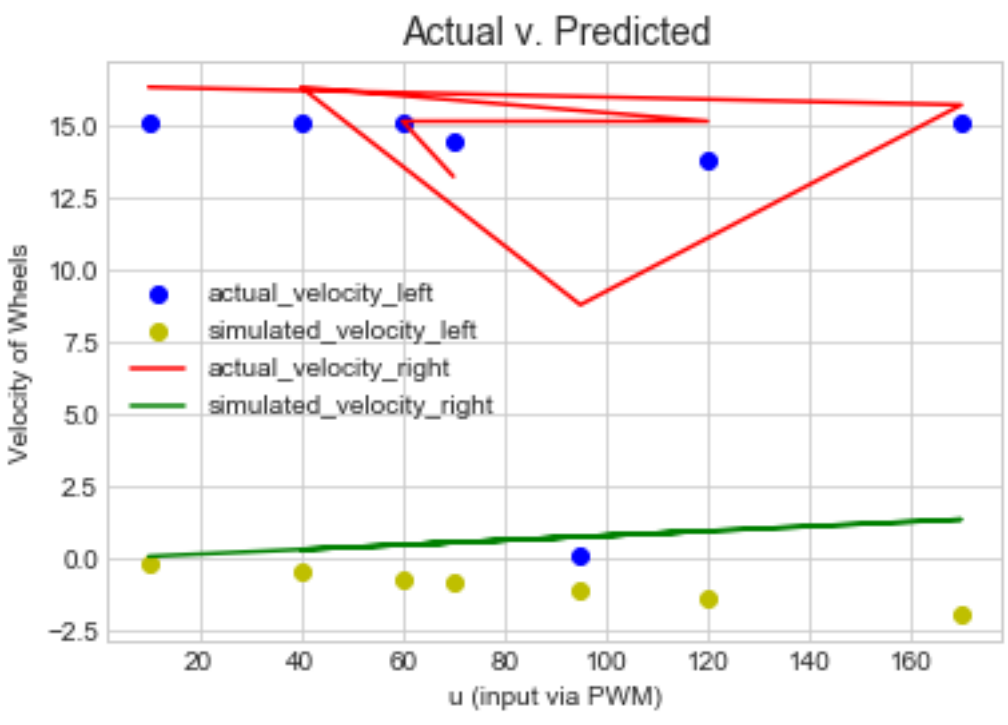
```
# extracting parameters, still we will use simpler model from above for now
theta_left, beta_left = -1.14978406e-02,  1.82203795e-03
theta_right, beta_right = 7.99460365e-03, -1.36993882e-04
```

In [225]:

```
# plot results of least squares fit
y_left_LS, y_right_LS = y_simulated_left, y_simulated_right
pwm_left.reset_index(drop=True, inplace=True)
pwm_right.reset_index(drop=True, inplace=True)
u = [[int(pwm_left[i]), int(pwm_right[i])] for i in range(len(y_left_LS))]
pwm_l = np.array([int(pwm_left[i]) for i in range(len(pwm_left))])

# u = np.array(u).reshape(-1)
u
vleft_LS = theta_left*pwm_l.T - beta_left
```

In [232]:

```
pwm_r = np.array([int(pwm_right[i]) for i in range(len(pwm_right))])
vright_LS = theta_right*pwm_r.T-beta_right

#Shifted for RIGHT WHEEL to avoid OVERPLOTTING
plt.plot(pwm_l.T, y_actual_left, 'bo', label='actual_velocity_left')
plt.plot(pwm_l.T, vleft_LS, 'yo', label='simulated_velocity_left')
plt.plot(pwm_r.T, y_actual_right, 'r-', label='actual_velocity_right')
plt.plot(pwm_r.T, vright_LS, 'g-', label='simulated_velocity_right')
plt.xlabel("u (input via PWM)")
plt.ylabel("Velocity of Wheels")
plt.title("Actual v. Predicted", fontsize=14)
plt.legend()
plt.show()
```



It looks like on HIGH and LOW PWM values our model makes nearly perfect prediction of the wheels' velocities, while UNDERESTIMATING for LEFT wheel and OVERESTIMATING for RIGHT wheel velocities.

Now let's plot ERROR as L2 loss =

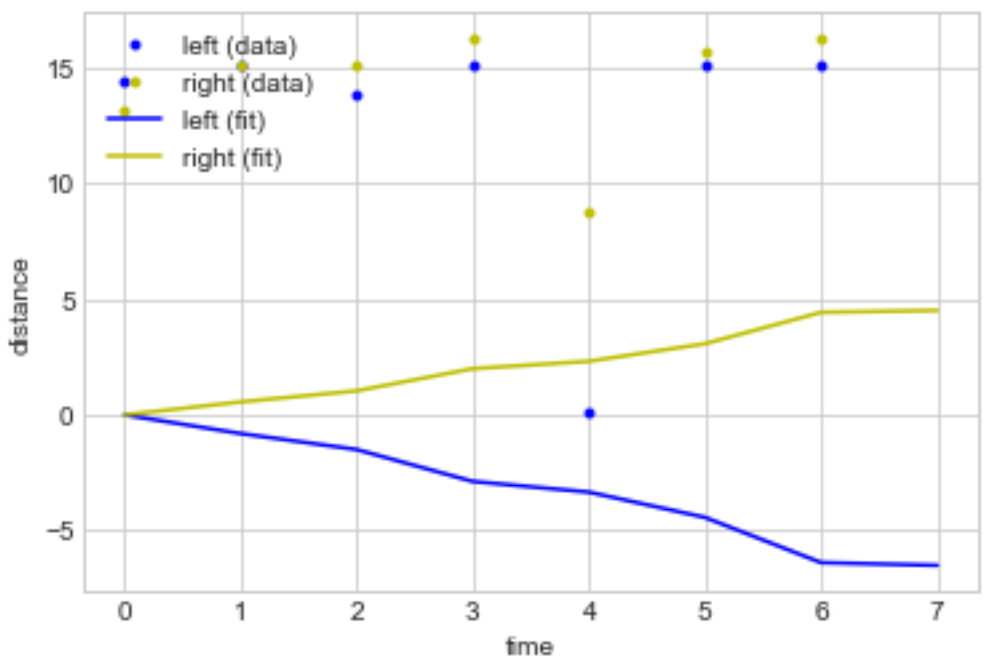
To test that we have estimated the parameters accurately, we will simulate the model using the parameters we have found. When we compare the simulated data with the collected data, we expect them to match up pretty well.

In [233]:

```
# model for simulation
def simulation(d0, u, theta, beta):
    d = np.zeros(len(u)+1)
#     d[0] = d0
    for t in range(len(u)):
        d[t+1] = d[t] + theta*u[t] - beta
    return d

# plot simulated trajectories
u = pwm_l.T
dleft_simulated  = simulation(y_actual_left, u, theta_left, beta_left)
dright_simulated = simulation(y_actual_right, u, theta_right, beta_right)
plt.plot(y_actual_left, 'b.',
         y_actual_right, 'y.',
         dleft_simulated, 'b-',
         dright_simulated, 'y-',
        )
plt.xlabel("time")
plt.ylabel("distance")
plt.legend(("left (data)", "right (data)", "left (fit)", "right (fit)"), loc='upper
```

Out[233]:

```
<matplotlib.legend.Legend at 0x1a176c1ed0>
```



In order to drive straight, the car must be operating at a velocity achievable by both wheels. A good choice of target velocity is the midpoint of the overlapping range of velocity. The below cell will calculate this. Green area in between 2 green lines is the optimal speed - in theory.

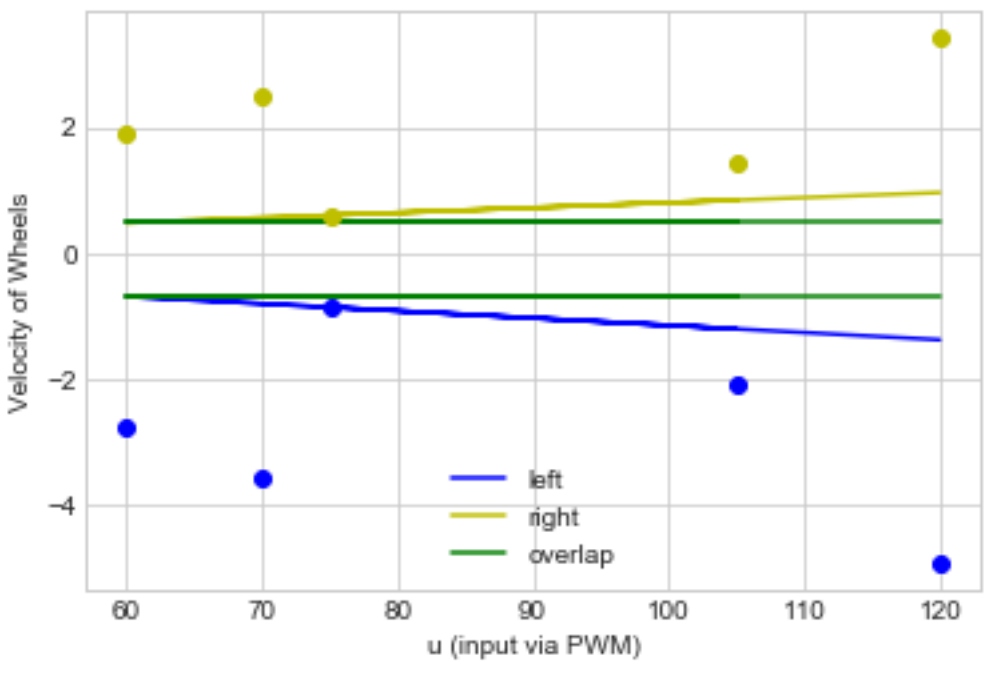
In [58]:

```
# This is from my old notes from Signal & Systems course I took in the past

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:\nv_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, dleft_simulated[1:], 'bo',  u, dright_simulated[1:], 'yo')
plt.xlabel("u (input via PWM)")
plt.ylabel("Velocity of Wheels")
plt.legend(("left", "right", "overlap"), loc=0)
```

```
Velocity range = [0.5, -0.7]
Operating point:
v_star = -0.1;
```

Out[58]:

```
<matplotlib.legend.Legend at 0x1a190ceb10>
```



Hence: Optimal VELOCITY RANGE = [0.8, -1.1]

Operating point: v_star = -0.2;

In [ ]:

```
```