Homework 3 MAE 207

Code for design study

In [9]:

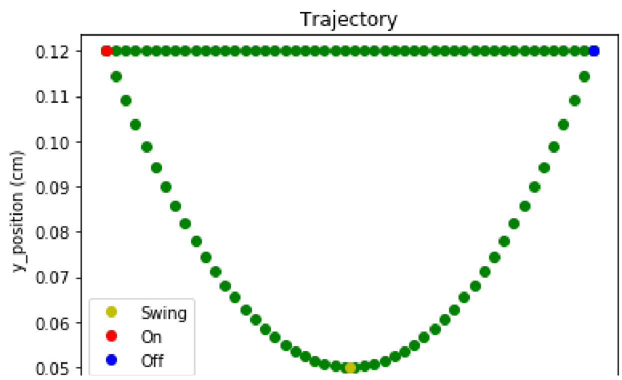
```
import numpy as np
import matplotlib as mpl

import matplotlib.pyplot as plt

from scipy.optimize import minimize
from scipy.optimize import fsolve

def IK_5_link(x, y, l1 = 0.09, l2 = 0.16, w = 0.07):

    def leg_wide(var):
        return np.linalg.norm([var[1] - np.pi, var[0]])

    def x_constraint_equation(var):
        # should be equal to zero when the
        return l1**2 - l2**2 + (x - w/2)**2 + y**2 - 2*l1*(y*np.sin(var[0]) + (x - w/2)*np.cos(var[0]))

    def y_constraint_equation(var):
        return l1**2 - l2**2 + (x + w/2)**2 + y**2 - 2*l1*(y*np.sin(var[1]) + (x + w/2)*np.cos(var[1]))


    res = minimize(leg_wide, (0.1, 9*np.pi/10), method="SLSQP", constraints= ({"type": "eq", "fun": x_c
onstraint_equation},
                                                                               {"type": "eq", "fun": y_
constraint_equation}))

    return (res, np.linalg.norm([x_constraint_equation(res.x), y_constraint_equation(res.x)]))

def design_study(x_on, y_on, x_off, y_off, x_swing, y_swing, T, d, N):
    thetaR_stance = []
    thetaL_stance = []
    thetaR_swing = []
    thetaL_swing = []

    ##### x_on to x_off (straight line movement) #####
    x_points = [x_on,x_off]
    y_points = [y_on,y_off]
    degree = 1
    z = np.polyfit(x_points, y_points, degree)
    p = np.poly1d(z)
    pprime = np.poly1d.deriv(p)

    increments_stance = N*d
    x_plot = np.linspace(x_on,x_off,increments_stance)
    y_plot = p(x_plot)

    plt.plot(x_plot, y_plot, 'go')



    for i in range(len(x_plot)):
        res = IK_5_link(x_plot[i], y_plot[i])
        thetaR = res[0].x[0]
        thetaL = res[0].x[1]
        thetaR_stance.append(thetaR)
        thetaL_stance.append(thetaL)

    #### x_off to x_swing to x_on
    x_points = [x_off, x_swing, x_on]
    y_points = [y_off, y_swing, y_on]
    degree = 2
    z = np.polyfit(x_points, y_points, degree)
    p = np.poly1d(z)
```

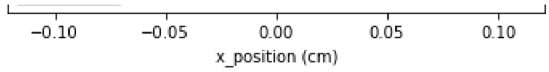
```
    increments_swing = N*(1-d)
    x_plot = np.linspace(x_off,x_on,increments_swing)
    y_plot = p(x_plot)

    plt.plot(x_plot, y_plot, 'go')
    plt.plot(x_swing,y_swing,'yo', label = 'Swing')
    plt.plot(x_on,y_on,'ro', label='On')
    plt.plot(x_off,y_off,'bo', label='Off')
    plt.legend()
    plt.title("Trajectory")
    plt.ylabel("y_position (cm)")
    plt.xlabel("x_position (cm)")
    plt.show()

    #### Getting theta values #####
    for i in range(len(x_plot)):
        res = IK_5_link(x_plot[i], y_plot[i])
        thetaR = res[0].x[0]
        thetaL = res[0].x[1]
        thetaR_swing.append(thetaR)
        thetaL_swing.append(thetaL)


    time_increments_stance = np.linspace(0, d*T, increments_stance)
    time_increments_swing = np.linspace(d*T, T, increments_swing)

    plt.plot(time_increments_stance,thetaL_stance, 'bo')
    plt.plot(time_increments_swing,thetaL_swing, 'bo')
    plt.title("Theta L vs Time")
    plt.ylabel("Radians")
    plt.xlabel("Time (s)")
    plt.show()
    plt.plot(thetaR_stance, thetaL_stance, 'go')
    plt.plot(thetaR_swing, thetaL_swing, 'go')
    plt.title("Theta R vs Theta L")
    plt.show()
    plt.plot(time_increments_stance,thetaR_stance, 'bo')
    plt.plot(time_increments_swing,thetaR_swing, 'bo')
    plt.title("Theta R vs Time")
    plt.ylabel("Radians")
    plt.xlabel("Time (s)")
    plt.show()
```

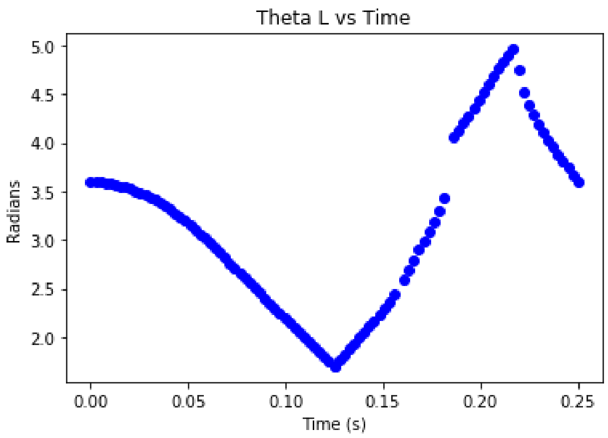
Design Study Gait 1 Plots

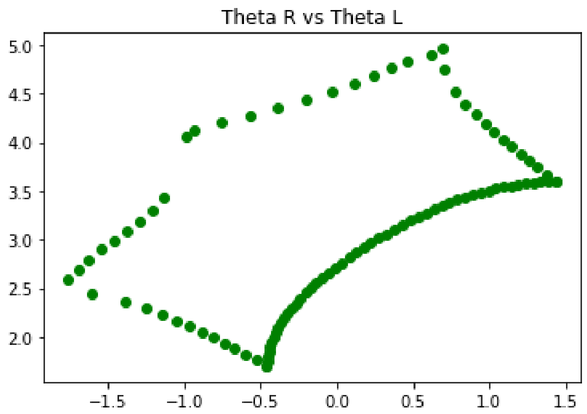
In [10]:

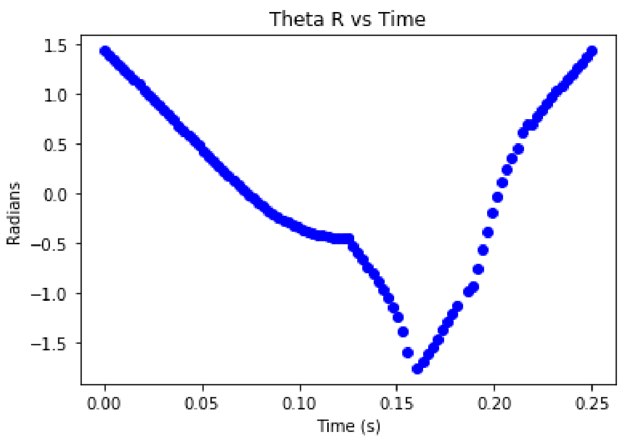
```
x_on = -0.11
y_on = 0.12
x_off = 0.11
y_off = 0.12
x_swing = 0
y_swing = 0.05
T = 0.25
d = 0.5
N = 100
design_study(x_on, y_on, x_off, y_off, x_swing, y_swing, T, d, N)
```







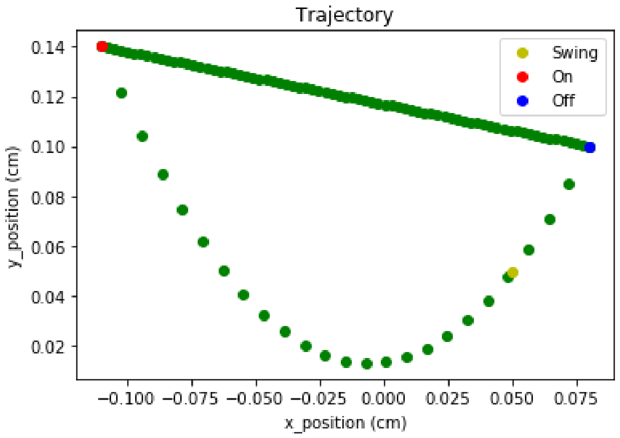


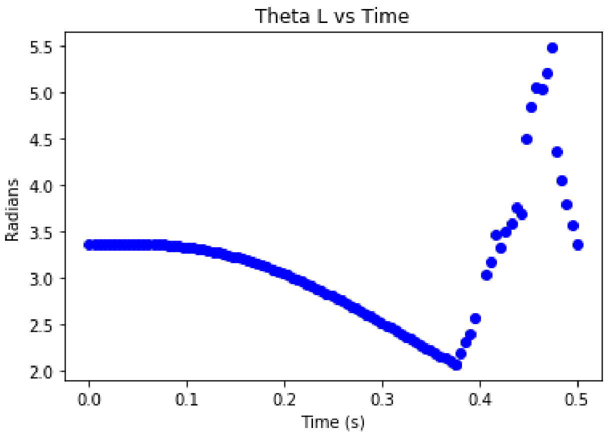


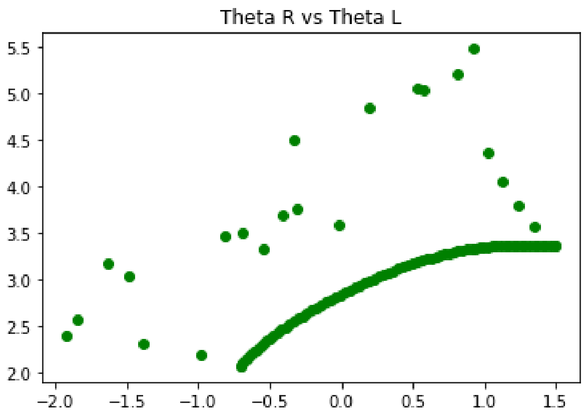
Design Study Gait 2 Plots

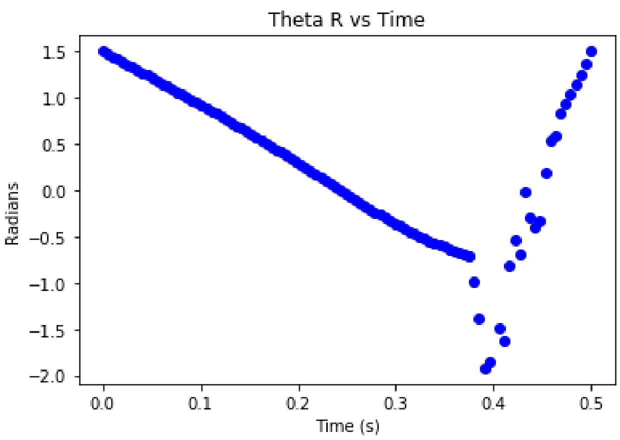
In [11]:

```
x_on = -0.11
y_on = 0.14
x_off = 0.08
y_off = 0.10
x_swing = 0.05
y_swing = 0.05
T = 0.50
d = 0.75
N = 100
design_study(x_on, y_on, x_off, y_off, x_swing, y_swing, T, d, N)
```





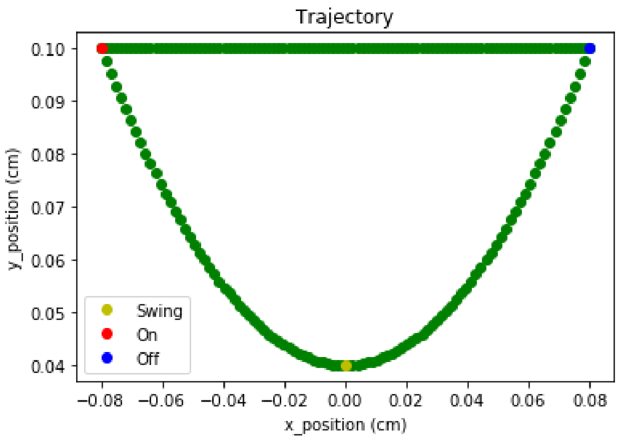


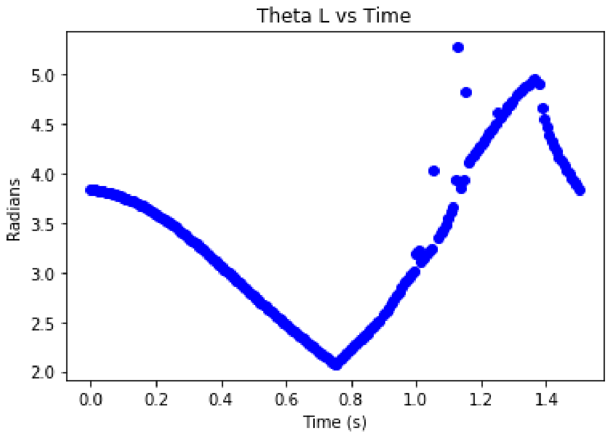


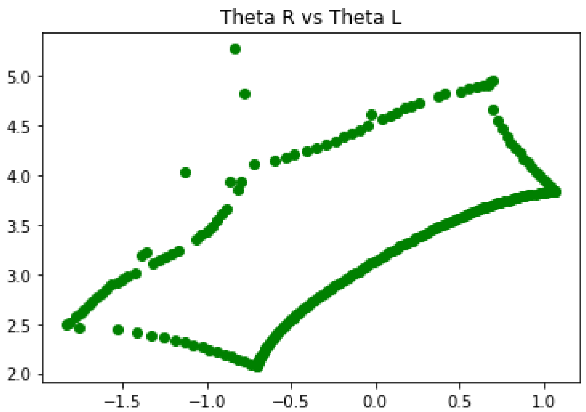
Design Study Gait 3 Plots

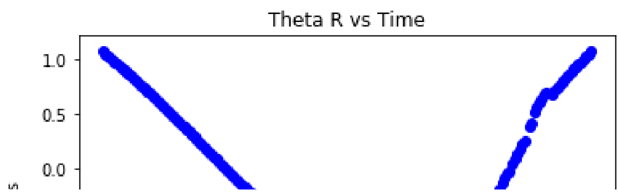
In [12]:

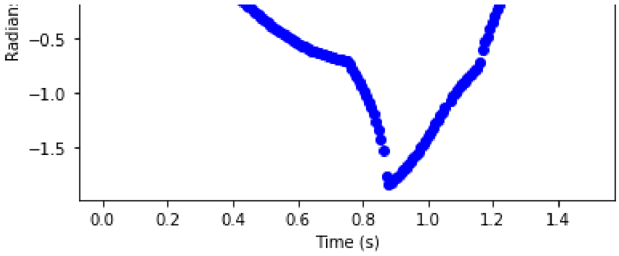
```
x_on = -0.08
y_on = 0.10
x_off = 0.08
y_off = 0.10
x_swing = 0.0
y_swing = 0.04
T = 1.5
d = 0.5
N = 200
design_study(x_on, y_on, x_off, y_off, x_swing, y_swing, T, d, N)
```





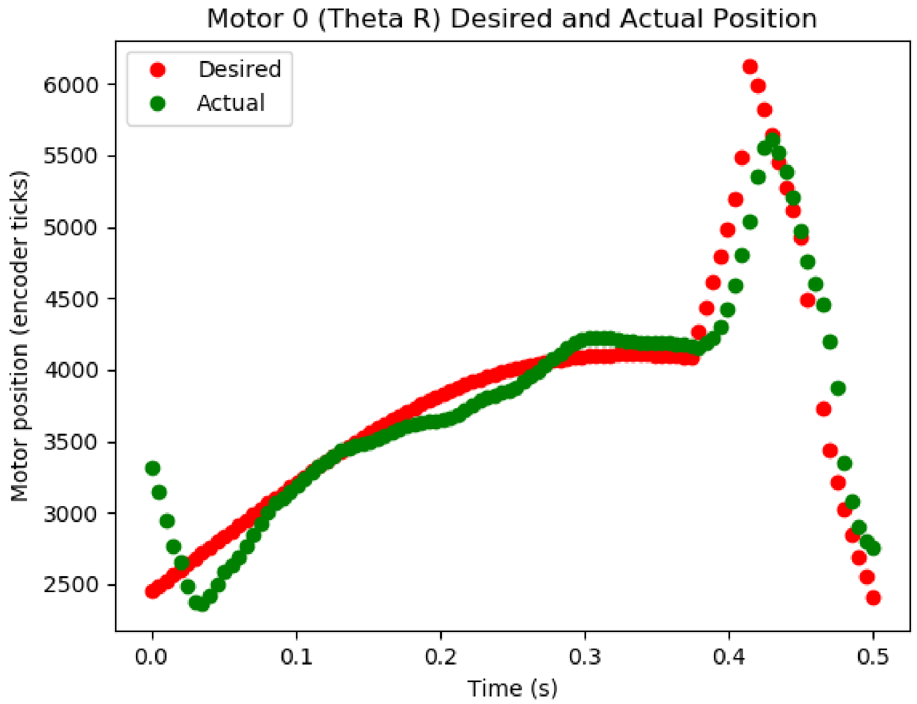


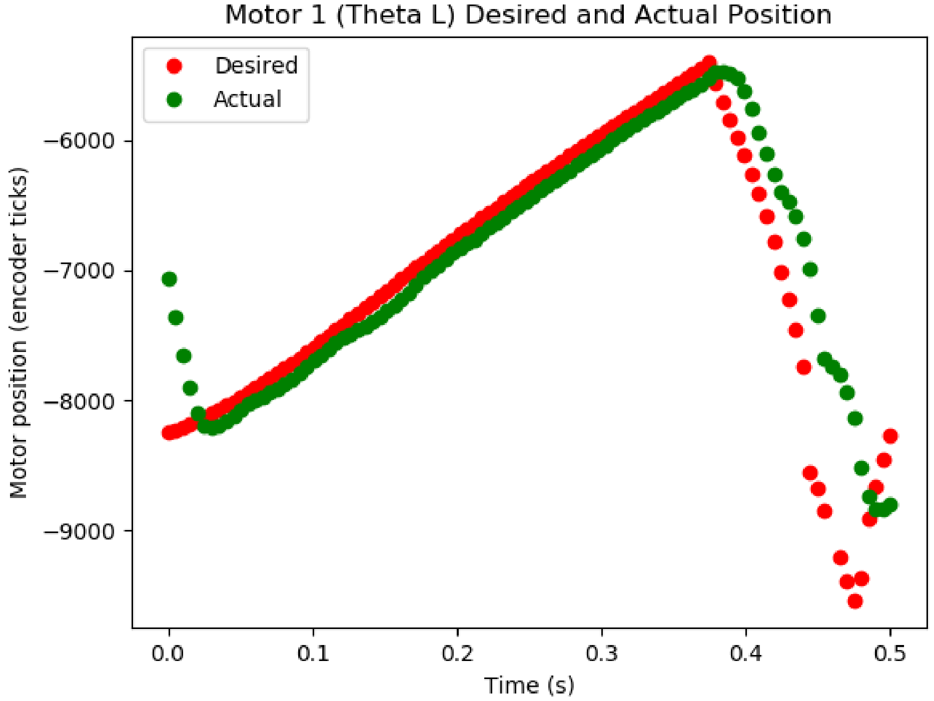




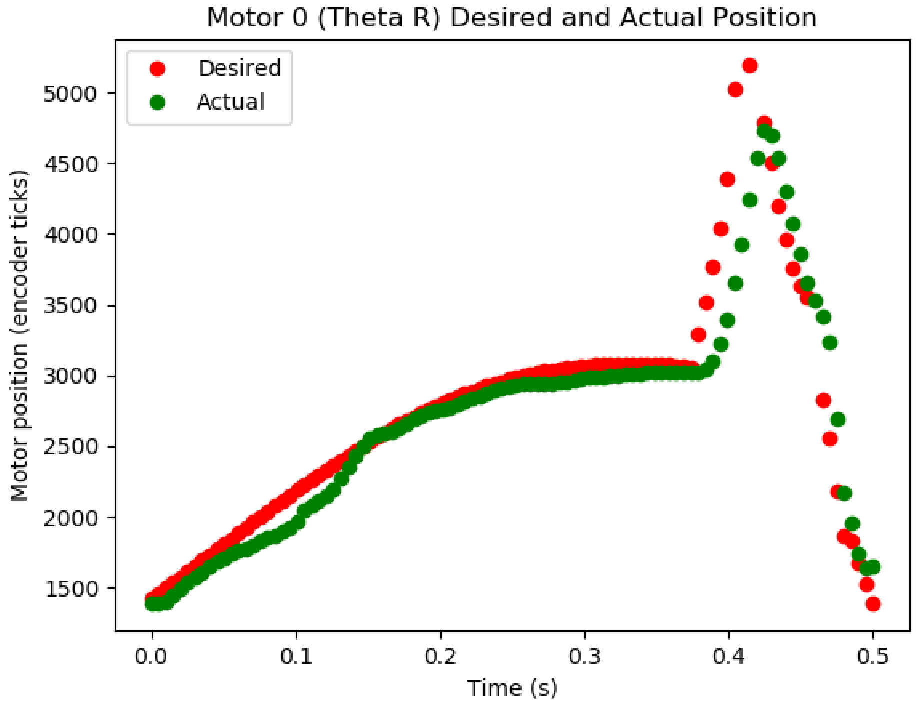
Hands On Study

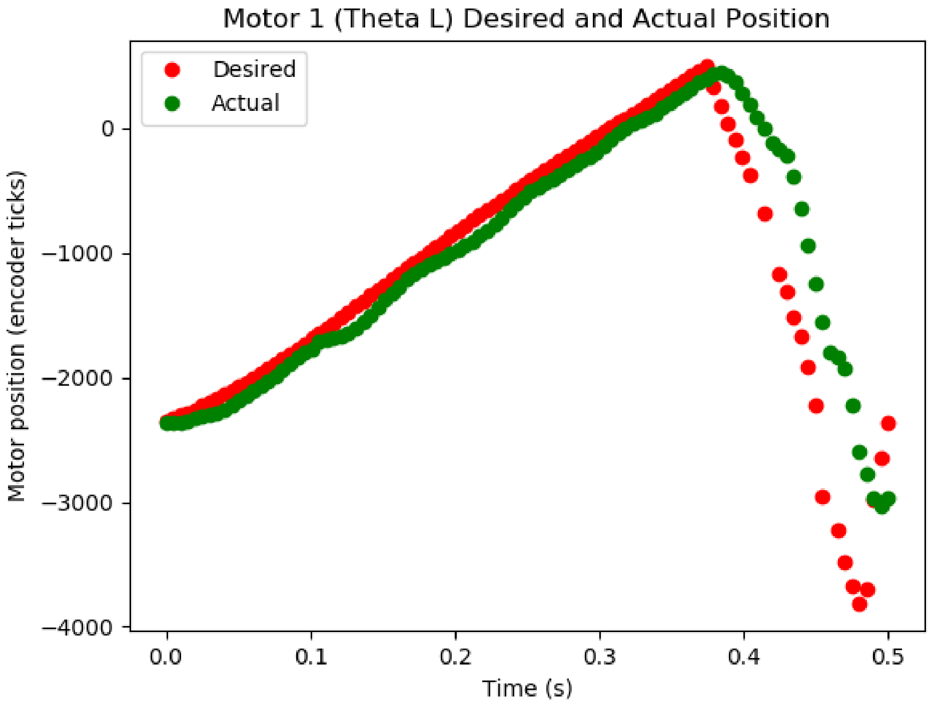
Motor Positions Gait 1



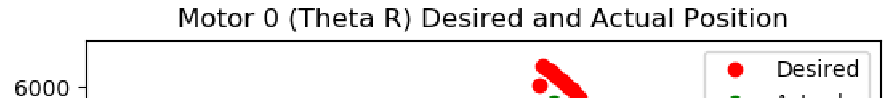


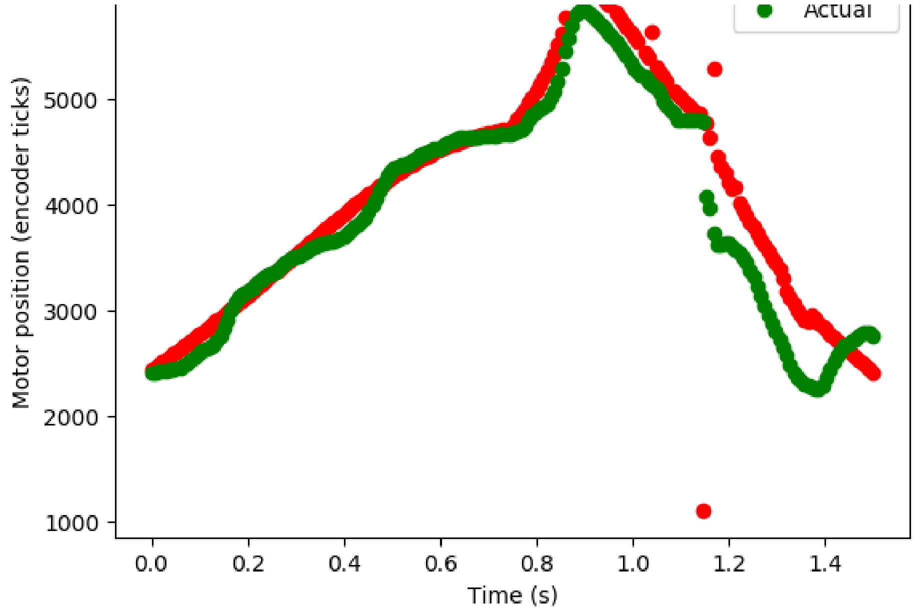
Motor Positions Gait 2

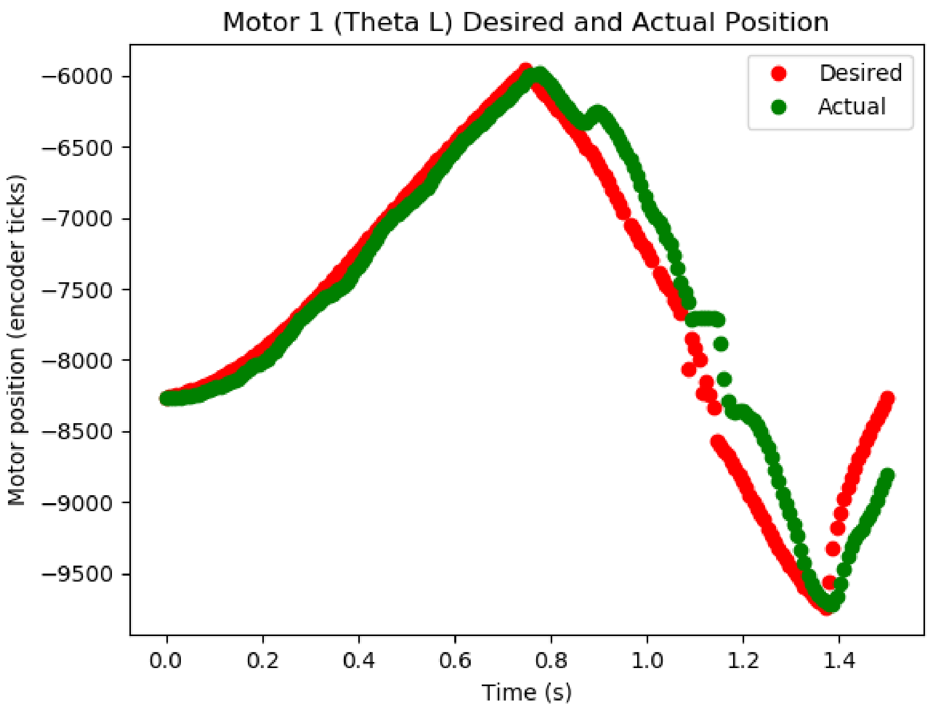




Motor Positions Gait 3







Code below used for controlling motors - Partner for project is James Salem

In [ ]:

```
import numpy as np

import matplotlib as mpl
mpl.use('Qt5Agg')

import matplotlib.pyplot as plt
plt.ion()

# for the symbolic manipulation of jacobian
import sympy as sp
# from sympy import symbols
# from sympy import sin, cos, asin, acos, pi, atan2, sqrt
from sympy.utilities.lambdify import lambdify
# from sympy import Matrix

from scipy.optimize import minimize
from scipy.optimize import fsolve
```

```
import time

import odrive
import odrive.utils
import odrive.enums
```

In [ ]:

```
odrv0 = odrive.find_any()
if odrv0 is not None:
    print('Connected!')
    print('Odrive serial {}'.format(odrv0.serial_number))

    m0 = odrv0.axis0.motor.is_calibrated
    m1 = odrv0.axis1.motor.is_calibrated

    print('Motor 0 calibrated: {}'.format(m0))
    print('Motor 1 calibrated: {}'.format(m1))

else:
    print('Not connected')
```

In [ ]:

```
odrv0.axis0.controller.config.vel_limit = 200000
odrv0.axis1.controller.config.vel_limit = 200000
```

In [ ]:

```
odrv0.axis0.requested_state = odrive.enums.AXIS_STATE_FULL_CALIBRATION_SEQUENCE
odrv0.axis1.requested_state = odrive.enums.AXIS_STATE_FULL_CALIBRATION_SEQUENCE

time.sleep(15)

print('\t Motor 0 calibration result: {} \r\n'.format(odrv0.axis0.motor.is_calibrated),
      '\t Motor 1 calibration result: {}'.format(odrv0.axis1.motor.is_calibrated))
```

In [ ]:

```
odrv0.axis0.requested_state = odrive.enums.AXIS_STATE_CLOSED_LOOP_CONTROL
odrv0.axis1.requested_state = odrive.enums.AXIS_STATE_CLOSED_LOOP_CONTROL

odrv0.axis0.controller.set_pos_setpoint(1.2343835830688477,0,0)
odrv0.axis1.controller.set_pos_setpoint(-3207.765625,0,0)
```

In [ ]:

```
odrv0.axis0.requested_state = odrive.enums.AXIS_STATE_IDLE
odrv0.axis1.requested_state = odrive.enums.AXIS_STATE_IDLE
```

In [ ]:

```
# adjust joint angles

adjusted_joint_angles = np.pi - joint_angles
```

In [ ]:

```
motor_cpr = (odrv0.axis0.encoder.config.cpr,
             odrv0.axis1.encoder.config.cpr)

def convert_joints(angles, cpr=motor_cpr, zero_pos = (2774.234375-4096, -1452.25)):
    encoder_vals = (angles * cpr[0] / (2 * np.pi)) + zero_pos
    return encoder_vals
```

In [ ]:

```
motor_cpr = (odrv0.axis0.encoder.config.cpr,
             odrv0.axis1.encoder.config.cpr)

def get_joints(odrv, cpr = motor_cpr, zero_pos = (0,0)):
    m0 = 2*np.pi*(odrv.axis0.encoder.pos_estimate - zero_pos[0])/motor_cpr[0]
    m1 = 2*np.pi*(odrv.axis1.encoder.pos_estimate - zero_pos[1])/motor_cpr[1]

    return (m0, m1)

joints = get_joints(odrv0)

print('encoder0: ', odrv0.axis0.encoder.pos_estimate)
print('encoder1: ', odrv0.axis1.encoder.pos_estimate)
print('angles in rad: ',joints)
print('converted to encoder: ', convert_joints(np.array(list(joints))))

print('test: ', convert_joints(np.array([0,0])))
```

In [ ]:

```
# get encoder values
encoder_vals = convert_joints(adjusted_joint_angles)
```

In [ ]:

```
import math
T = 0.5
N = 100
time_per_tick = T / N

odrv0.axis0.requested_state = odrive.enums.AXIS_STATE_CLOSED_LOOP_CONTROL
odrv0.axis1.requested_state = odrive.enums.AXIS_STATE_CLOSED_LOOP_CONTROL

actual_motor_pos0 = []
actual_motor_pos1 = []

for j in range(5):
    for i in range(encoder_vals.shape[0]):
        start = time.time()
        thetaR = encoder_vals[i,0]
        thetaL = encoder_vals[i,1]
        if math.isnan(thetaR):
            print("nan")
        else:

            odrv0.axis0.controller.set_pos_setpoint(thetaR,0,0)
            odrv0.axis1.controller.set_pos_setpoint(thetaL,0,0)

        M0 = odrv0.axis0.encoder.pos_estimate
        M1 = odrv0.axis1.encoder.pos_estimate

        actual_motor_pos0.append(M0)
        actual_motor_pos1.append(M1)

        end = time.time()

        time.sleep(time_per_tick - (start - end))
```

Code below used to plot configurations to ensure simulation angles are correct before using motors

In [7]:

```
l1 = 0.09;                    # m
l2 = 0.16;                    # m
```

```python
w = 0.07;                     # m

def internal_angles(thetaR, thetaL, l1 = l1, l2 = l2, w = w):

    #
    # system_of_equations = @(x) sum(abs([w + l1*(cos(b1)-cos(a1)) + l2*(cos(x(2))-cos(x(1)));
    #                                     l1*(sin(b1)-sin(a1)) + l2*(sin(x(2))-sin(x(1)))]));
    #
    # x_guess = [a1, b1 + pi/2];
    # % [x,fval,exitflag,output] = fminsearch(system_of_equations, x_guess);
    # [x,fval,exitflag,output] = fmincon(system_of_equations, ...
    #                                     x_guess, ...
    #                                     [], [], [], [], ...
    #                                     [-pi, -pi], [pi, pi]);
    #
    # a2 = x(1);
    # b2 = x(2);
    #
    # x = w/2 + l1*cos(b1) + l2*cos(b2);
    # y = l1*sin(b1) + l2*sin(b2);

    def sys(x):
        return (w + l1*np.cos(thetaR) + l2*np.cos(x[0]) - l1*np.cos(thetaL) - l2*np.cos(x[1]),
                l1*np.sin(thetaR) + l2*np.sin(x[0]) - l1*np.sin(thetaL) - l2*np.sin(x[1]))

    alphaR, alphaL = fsolve(sys, (np.pi/2, np.pi/2))

    alphaR = alphaR % (2*np.pi)
    alphaL = alphaL % (2*np.pi)

    # Copmute FK for checking
    x = w/2 + l1*np.cos(thetaR) + l2*np.cos(alphaR);
    y = l1*np.sin(thetaR) + l2*np.sin(alphaR);

    return (alphaR, alphaL, x, y)

def draw_robot(thetaR, thetaL, link1 = l1, link2 = l2, width = w, ax = False):
    #plt.ion()
    # Solve for internal angles
    (alphaR, alphaL, x, y) = internal_angles(thetaR, thetaL)

    def pol2cart(rho, phi):
        x = rho * np.cos(phi)
        y = rho * np.sin(phi)
        return(x, y)



    plt.plot(-width/2, 0, 'ok')
    plt.plot(width/2, 0, 'ok')

    plt.plot([-width/2, 0], [0, 0], 'k')
    plt.plot([width/2, 0], [0, 0], 'k')

    plt.plot(-width/2 + np.array([0, link1*np.cos(thetaL)]), [0, link1*np.sin(thetaL)], 'k')
    plt.plot(width/2 + np.array([0, link1*np.cos(thetaR)]), [0, link1*np.sin(thetaR)], 'k')

    plt.plot(-width/2 + link1*np.cos(thetaL) + np.array([0, link2*np.cos(alphaL)]), \
             link1*np.sin(thetaL) + np.array([0, link2*np.sin(alphaL)]), 'k');

    plt.plot(width/2 + link1*np.cos(thetaR) + np.array([0, link2*np.cos(alphaR)]), \
             np.array(link1*np.sin(thetaR) + np.array([0, link2*np.sin(alphaR)])), 'k');

    plt.plot(x, y, 'ro');
    plt.xlim((-0.5,0.5))
    plt.ylim((-0.2,0.2))
    plt.show()
#     plt.plot(x_end, y_end, 'go');

    #ax.axis([-.3,.3,-.1,.3])
```

In [14]:

```python
thetaR_total = thetaR_stance
thetaR_total.extend(thetaR_swing)
```

```
thetaR_total.extend(thetaR_swing)
thetaL_total = thetaL_stance
thetaL_total.extend(thetaL_swing)
for i in range(len(thetaR_total)):
    thetaR = thetaR_total[i]
    thetaL = thetaL_total[i]
    draw_robot(thetaR, thetaL)
```