

```

1 import numpy as np
2
3 def integrate(f, xt, dt):
4     """
5     This function takes in an initial condition x(t) and a timestep dt,
6     as well as a dynamical system f(x) that outputs a vector of the
7     same dimension as x(t). It outputs a vector x(t+dt) at the future
8     time step.
9
10    Parameters
11    =====
12    dyn: Python function
13         derivate of the system at a given step x(t),
14         it can considered as  $\dot{x}(t) = \text{func}(x(t))$ 
15    xt: NumPy array
16         current step x(t)
17    dt:
18         step size for integration
19
20    Return
21    =====
22    new_xt:
23         value of x(t+dt) integrated from x(t)
24    """
25    k1 = dt * f(xt)
26    k2 = dt * f(xt+k1/2.)
27    k3 = dt * f(xt+k2/2.)
28    k4 = dt * f(xt+k3)
29    new_xt = xt + (1/6.) * (k1+2.0*k2+2.0*k3+k4)
30    return new_xt
31
32 def simulate(f, x0, tspan, dt, integrate):
33     """
34     This function takes in an initial condition x0, a timestep dt,
35     a time span tspan consisting of a list [min_time, max_time],
36     as well as a dynamical system f(x) that outputs a vector of the
37     same dimension as x0. It outputs a full trajectory simulated
38     over the time span of dimensions (xvec_size, time_vec_size).
39
40    Parameters
41    =====
42    f: Python function
43         derivate of the system at a given step x(t),
44         it can considered as  $\dot{x}(t) = \text{func}(x(t))$ 
45    x0: NumPy array
46         initial conditions
47    tspan: Python list
48         tspan = [min_time, max_time], it defines the start and end
49         time of simulation
50    dt:
51         time step for numerical integration
52    integrate: Python function
53         numerical integration method used in this simulation
54
55    Return
56    =====

```

```

57 x_traj:
58     simulated trajectory of x(t) from t=0 to tf
59     """
60     N = int((max(tspan)-min(tspan))/dt)
61     x = np.copy(x0)
62     tvec = np.linspace(min(tspan),max(tspan),N)
63     xtraj = np.zeros((len(x0),N))
64     for i in range(N):
65         xtraj[:,i]=integrate(f,x,dt)
66         x = np.copy(xtraj[:,i])
67     return xtraj
68
69 def animate_double_pend(theta_array,L1=1,L2=1,T=10):
70     """
71     Function to generate web-based animation of double-pendulum system
72
73     Parameters:
74     =====
75     theta_array:
76         trajectory of theta1 and theta2, should be a NumPy array with
77         shape of (2,N)
78     L1:
79         length of the first pendulum
80     L2:
81         length of the second pendulum
82     T:
83         length/seconds of animation duration
84
85     Returns: None
86     """
87
88     #####
89     # Imports required for animation.
90     from plotly.offline import init_notebook_mode, iplot
91     from IPython.display import display, HTML
92     import plotly.graph_objects as go
93
94     #####
95     # Browser configuration.
96     def configure_plotly_browser_state():
97         import IPython
98         display(IPython.core.display.HTML('''
99             <script src="/static/components/requirejs/require.js"></script>
100             <script>
101                 requirejs.config({
102                     paths: {
103                         base: '/static/base',
104                         plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
105                     },
106                 });
107             </script>
108             '''))
109     configure_plotly_browser_state()
110     init_notebook_mode(connected=False)
111
112     #####
113     # Getting data from pendulum angle trajectories.
114     xx1=L1*np.sin(theta_array[0])

```

```

115 yy1=-L1*np.cos(theta_array[0])
116 xx2=xx1+L2*np.sin(theta_array[0]+theta_array[1])
117 yy2=yy1-L2*np.cos(theta_array[0]+theta_array[1])
118 N = len(theta_array[0]) # Need this for specifying length of simulation
119
120 #####
121 # Using these to specify axis limits.
122 xm=np.min(xx1)-0.5
123 xM=np.max(xx1)+0.5
124 ym=np.min(yy1)-2.5
125 yM=np.max(yy1)+1.5
126
127 #####
128 # Defining data dictionary.
129 # Trajectories are here.
130 data=[dict(x=xx1, y=yy1,
131           mode='lines', name='Arm',
132           line=dict(width=2, color='blue')
133           ),
134       dict(x=xx1, y=yy1,
135           mode='lines', name='Mass 1',
136           line=dict(width=2, color='purple')
137           ),
138       dict(x=xx2, y=yy2,
139           mode='lines', name='Mass 2',
140           line=dict(width=2, color='green')
141           ),
142       dict(x=xx1, y=yy1,
143           mode='markers', name='Pendulum 1 Traj',
144           marker=dict(color="purple", size=2)
145           ),
146       dict(x=xx2, y=yy2,
147           mode='markers', name='Pendulum 2 Traj',
148           marker=dict(color="green", size=2)
149           ),
150       ]
151
152 #####
153 # Preparing simulation layout.
154 # Title and axis ranges are here.
155 layout=dict(xaxis=dict(range=[xm, xM], autorange=False, zeroline=False,dtick=1),
156            yaxis=dict(range=[ym, yM], autorange=False, zeroline=False,scaleanchor = "x",dtick=1),
157            title='Double Pendulum Simulation',
158            hovermode='closest',
159            updatemenus= [{'type': 'buttons',
160                          'buttons': [{'label': 'Play','method': 'animate',
161                                     'args': [None, {'frame': {'duration': T, 'redraw': False}}]},
162                                     {'args': [[None], {'frame': {'duration': T, 'redraw': False}, 'mode': 'immediate',
163                                     'transition': {'duration': 0}}], 'label': 'Pause','method': 'animate'}
164                          ]
165            })
166
167
168 #####
169 # Defining the frames of the simulation.
170 # This is what draws the lines from
171 # joint to joint of the pendulum.
172 frames=[dict(data=[dict(x=[0,xx1[k],xx2[k]],

```

```

173         y=[0,yy1[k],yy2[k]],
174         mode='lines',
175         line=dict(color='red', width=3)
176     ),
177     go.Scatter(
178         x=[xx1[k]],
179         y=[yy1[k]],
180         mode="markers",
181         marker=dict(color="blue", size=12)),
182     go.Scatter(
183         x=[xx2[k]],
184         y=[yy2[k]],
185         mode="markers",
186         marker=dict(color="blue", size=12)),
187     ]) for k in range(N)]
188
189 #####
190 # Putting it all together and plotting.
191 figure1=dict(data=data, layout=layout, frames=frames)
192 iplot(figure1)

```

▼ PARAMS

```

1 ##WEAPON BAR##
2 m1 = 0.5
3 l1 = 1
4 w1 = 0.25
5 inertia1 = (m1/12)*(l1**2 + w1**2)
6
7 m2 = 3
8 l2 = 1
9 w2 = 1
10 inertia2 =(m2/12)*(l2**2 + w2**2)
11

```

▼ SETUP AND LAG AND EL

```

1 import sympy as sym
2 from sympy import symbols, Matrix ,Function, solve,cos,sin,simplify,Eq,transpose,zeros,eye
3 from sympy.abc import x, y, z,t,g,r,R,m,M,p
4 from sympy import sin, cos
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 theta1 = Function(r"theta1")(t)
10 theta2 = Function(r"theta2")(t)
11 x1 = Function(r"x_1")(t)
12 x2 = Function(r"x_2")(t)
13 y1 = Function(r"y_1")(t)
14 y2 = Function(r"y_2")(t)

```

```

15 theta1dot = theta1.diff(t)
16 theta2dot = theta2.diff(t)
17 x1dot = x1.diff(t)
18 x2dot = x2.diff(t)
19 y1dot = y1.diff(t)
20 y2dot = y2.diff(t)
21
22
23
24 q = Matrix([x1,y1,theta1,x2,y2,theta2])
25 qdot = Matrix([x1dot,y1dot,theta1dot,x2dot,y2dot,theta2dot])
26
27 #Rbw = Matrix([[sym.cos(theta1),-1*sym.sin(theta1)],[sym.sin(theta1),sym.cos(theta1)]])\
28 #FORMAT: t_SpaceDescribingIn_thingBeingDescribed
29
30 #BATTLEBOT
31 t_s_a = Matrix([[1, 0, x1],
32                [0, 1, y1],
33                [0, 0, 1]])
34
35 t_a_b = Matrix([[sym.cos(theta1), -sym.sin(theta1), 0],
36                [sym.sin(theta1), sym.cos(theta1), 0],
37                [0, 0, 1]])
38 t_s_b = t_s_a * t_a_b
39 #BLOCK
40 t_s_c = Matrix([[1, 0, x2],
41                [0, 1, y2],
42                [0, 0, 1]])
43
44 t_c_d = Matrix([[sym.cos(theta2), -sym.sin(theta2), 0],
45                [sym.sin(theta2), sym.cos(theta2), 0],
46                [0, 0, 1]])
47 t_s_d = t_s_c * t_c_d
48 display(t_s_d)
49 #####
50
51 homo = Matrix([0,0,1])
52 tcubedot = t_s_d.diff(t)
53 tweapdot = t_s_b.diff(t)
54
55 v1bar = tcubedot * homo
56 v2bar = tweapdot * homo
57 display(v1bar)
58 InertiaMat1 = Matrix([[1, 0, 0],[0, 1, 0],[0, 0, inertia1]])
59 InertiaMat2 = Matrix([[1, 0, 0],[0, 1, 0],[0, 0, inertia2]])
60
61 rotKE = 1/2 * inertia1 * theta1dot**2 + 1/2 * inertia2 * theta2dot**2
62 display(rotKE)
63 #rotKE = 1/2 * InertiaMat1 * theta1dot**2 + 1/2 * IntertiaMat2 * theta2dot**2
64 #rotKE = 0
65 #KE = 1/2 * m1 * v1bar.T* InertiaMat1 *v1bar + 1/2 * m2 * v2bar.T* InertiaMat2*v2bar
66 KE = 1/2 * m1 * v1bar.T*v1bar + 1/2 * m2 * v2bar.T*v2bar
67 KE = KE[0] + rotKE
68 display(KE)
69 PE = 0
70
71 Lag = KE - PE
72 #Calc Lagrange

```

```
73 print("Lagrangian is:")
74 display(sym.simplify(Lag))
```

$$\begin{bmatrix} \cos(\theta_2(t)) & -\sin(\theta_2(t)) & x_2(t) \\ \sin(\theta_2(t)) & \cos(\theta_2(t)) & y_2(t) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{d}{dt} x_2(t) \\ \frac{d}{dt} y_2(t) \\ 0 \end{bmatrix}$$

$$0.02213541666666667 \left(\frac{d}{dt} \theta_1(t) \right)^2 + 0.25 \left(\frac{d}{dt} \theta_2(t) \right)^2$$

$$0.02213541666666667 \left(\frac{d}{dt} \theta_1(t) \right)^2 + 0.25 \left(\frac{d}{dt} \theta_2(t) \right)^2 + 1.5 \left(\frac{d}{dt} x_1(t) \right)^2 + 0.25 \left(\frac{d}{dt} x_2(t) \right)^2 + 1.5 \left(\frac{d}{dt} y_1(t) \right)^2 + 0.25 \left(\frac{d}{dt} y_2(t) \right)^2$$

Lagrangian is:

$$0.02213541666666667 \left(\frac{d}{dt} \theta_1(t) \right)^2 + 0.25 \left(\frac{d}{dt} \theta_2(t) \right)^2 + 1.5 \left(\frac{d}{dt} x_1(t) \right)^2 + 0.25 \left(\frac{d}{dt} x_2(t) \right)^2 + 1.5 \left(\frac{d}{dt} y_1(t) \right)^2 + 0.25 \left(\frac{d}{dt} y_2(t) \right)^2$$

```
1 J = Lag
2
3 # compute derivative wrt a vector, method 1
4 # wrap the expression into a SymPy Matrix
5 J_mat = Matrix([J])
6 # SymPy Matrix has built-in method for Jacobian
7 term_one = J_mat.jacobian(q)
8 term_two = J_mat.jacobian(qdot)
9 #print('\n\033[1mfirst term:')
10 #display(term_one) # note that dJdq is 1-by-2 vector, not 2-by-1 as q
11 #print('\n\033[1msecond term:')
12 #display(term_two) # note that dJdq is 1-by-2 vector, not 2-by-1 as q
13
14
15
16 term_one = Lag.diff(q)
17 term_two = Lag.diff(qdot)
18 term_two_real = term_two.diff(t)
19 EL = term_one - term_two_real
20
21
22 dLdq = J_mat.jacobian(q).T
23 dLdqdot = J_mat.jacobian(qdot).T
24 d_dLdqdot_dt = dLdqdot.diff(t)
25 EL = sym.simplify(d_dLdqdot_dt - dLdq)
26
27
28 EL = sym.simplify(EL)
29 display(EL)
```

$$\begin{bmatrix} 3.0 \frac{d^2}{dt^2} x_1(t) \\ 3.0 \frac{d^2}{dt^2} y_1(t) \\ 0.04427083333333333 \frac{d^2}{dt^2} \theta_1(t) \\ 0.5 \frac{d^2}{dt^2} x_2(t) \\ 0.5 \frac{d^2}{dt^2} y_2(t) \\ 0.5 \frac{d^2}{dt^2} \theta_2(t) \end{bmatrix}$$

```

1 #Simulate
2
3 lhs = Matrix([EL[0],EL[1],EL[2],EL[3],EL[4],EL[5]])
4 Force = 0
5 # define right hand side as a Matrix
6 rhs = Matrix([0,0,Force,0,0,0])#
7
8 # define the equations
9 eqn = Eq(lhs, rhs)
10
11 theta1ddot = theta1dot.diff(t)
12 theta2ddot = theta2dot.diff(t)
13 x1ddot = x1dot.diff(t)
14 x2ddot = x2dot.diff(t)
15 y1ddot = y1dot.diff(t)
16 y2ddot = y2dot.diff(t)
17
18
19 q = Matrix([x1,y1,theta1,x2,y2,theta2])
20 qdot = Matrix([x1dot,y1dot,theta1dot,x2dot,y2dot,theta2dot])
21 qddot = Matrix([x1ddot,y1ddot,theta1ddot,x2ddot,y2ddot,theta2ddot])
22
23 #solve for theta1ddot and theta2ddot
24 soln = solve(eqn, [x1ddot,y1ddot,theta1ddot,x2ddot,y2ddot,theta2ddot])
25
26 display(soln)
27
28 x1sol = soln[x1ddot]
29 x2sol = soln[x2ddot]
30 y1sol = soln[y1ddot]
31 y2sol = soln[y2ddot]
32 theta1sol = soln[theta1ddot]
33 theta2sol = soln[theta2ddot]
34
35
36 x1dub = sym.lambdify([x1,y1,theta1,x2,y2,theta2,x1dot,y1dot,x2dot,y2dot,theta1dot,theta2dot],x1sol,modules=sym)
37 x2dub = sym.lambdify([x1,y1,theta1,x2,y2,theta2,x1dot,y1dot,x2dot,y2dot,theta1dot,theta2dot],x2sol,modules=sym)
38 y1dub = sym.lambdify([x1,y1,theta1,x2,y2,theta2,x1dot,y1dot,x2dot,y2dot,theta1dot,theta2dot],y1sol,modules=sym)
39 y2dub = sym.lambdify([x1,y1,theta1,x2,y2,theta2,x1dot,y1dot,x2dot,y2dot,theta1dot,theta2dot],y2sol,modules=sym)
40 theta1dub = sym.lambdify([x1,y1,theta1,x2,y2,theta2,x1dot,y1dot,x2dot,y2dot,theta1dot,theta2dot],theta1sol,modules=sym)
41 theta2dub = sym.lambdify([x1,y1,theta1,x2,y2,theta2,x1dot,y1dot,x2dot,y2dot,theta1dot,theta2dot],theta2sol,modules=sym)
42

```

```

{Derivative(theta1(t), (t, 2)): 0.0,
 Derivative(theta2(t), (t, 2)): 0.0,
 Derivative(x_1(t), (t, 2)): 0.0,
 Derivative(x_2(t), (t, 2)): 0.0,
 Derivative(y_1(t), (t, 2)): 0.0,
 Derivative(y_2(t), (t, 2)): 0.0}

```

```

1 def line_point_dist(point1,point2,point3):
2   xz = symbols(r"x_z")
3   yz = symbols(r"y_z")
4
5   if point2[0] == point1[0]:
6     d = ((point3[0]-point2[0])**2)**1/2

```

```

7 elif point2[1] == point1[1]:
8     d = ((point3[1]-point2[1])**2)**1/2
9 else:
10    m = ((point2[1]-point1[1])/(point2[0]-point1[0]))
11    #display(m)
12    line = m * (xz - point1[0]) + point1[1] - yz
13    C = line.subs({xz:0,yz:0})
14    A = m
15    B = -1
16    d = (((A*point3[0] + B * point3[1] + C)**2)**1/2)/((A**2 + B**2)**1/2)
17    return d
18 display(line_point_dist([-0.50, 2.00000000000000],
19     [0.500000000000000, 2.00000000000000],
20     [-0.0758567746186340, 0.509775195301280]))
21 #display(line_point_dist([x1,y1],[3,4],[x2,y2]))

1.1103849842696691

```

Impact Stuff

▼ Min Distance

```

1 import math
2 def minDistance(A, B, E) :
3
4     # vector AB
5     AB = [None, None];
6     AB[0] = B[0] - A[0];
7     AB[1] = B[1] - A[1];
8
9     # vector BP
10    BE = [None, None];
11    BE[0] = E[0] - B[0];
12    BE[1] = E[1] - B[1];
13
14    # vector AP
15    AE = [None, None];
16    AE[0] = E[0] - A[0];
17    AE[1] = E[1] - A[1];
18
19    # Variables to store dot product
20
21    # Calculating the dot product
22    AB_BE = AB[0] * BE[0] + AB[1] * BE[1];
23    AB_AE = AB[0] * AE[0] + AB[1] * AE[1];
24
25    # Minimum distance from
26    # point E to the line segment
27    reqAns = 0;
28    #print(AB_BE)
29    # Case 1

```

```

30     if (AB_BE > 0) :
31
32         # Finding the magnitude
33         y = E[1] - B[1];
34         x = E[0] - B[0];
35         reqAns = math.sqrt(x * x + y * y);
36
37     # Case 2
38     elif (AB_AE < 0) :
39         y = E[1] - A[1];
40         x = E[0] - A[0];
41         reqAns = math.sqrt(x * x + y * y);
42
43     # Case 3
44     else:
45
46         # Finding the perpendicular distance
47         x1 = AB[0];
48         y1 = AB[1];
49         x2 = AE[0];
50         y2 = AE[1];
51         mod = math.sqrt(x1 * x1 + y1 * y1);
52         reqAns = abs(x1 * y2 - y1 * x2) / mod;
53
54     return reqAns;
55
56 A = Matrix([0,0])
57 B = Matrix([2,0])
58 C = Matrix([1,1])
59 print(minDistance(A,B,C))

```

1.0000000000000000

▼ Find Impact Eqs

```

1 x1_pre = symbols('x_1pr')
2 y1_pre = symbols('y_1pr')
3 theta1_pre = symbols('theta_1pr')
4 x2_pre = symbols('x_2pr')
5 y2_pre = symbols('y_2pr')
6 theta2_pre = symbols('theta_2pr')
7 x1_predot = symbols('x_1prd')
8 y1_predot = symbols('y_1prd')
9 theta1_predot = symbols('theta_1prd')
10 x2_predot = symbols('x_2prd')
11 y2_predot = symbols('y_2prd')
12 theta2_predot = symbols('theta_2prd')
13
14 x1_post = symbols('x_1po')
15 y1_post = symbols('y_1po')
16 theta1_post = symbols('theta_1po')
17 x2_post = symbols('x_2po')
18 y2_post = symbols('y_2po')
19 theta2_post = symbols('theta_2po')
20 x1_postdot = symbols('x_1pod')

```

```

21 y1_postdot = symbols('y_1pod')
22 theta1_postdot = symbols('theta_1pod')
23 x2_postdot = symbols('x_2pod')
24 y2_postdot = symbols('y_2pod')
25 theta2_postdot = symbols('theta_2pod')
26 lam = symbols(r'lam')
27
28
29 p = Lag.diff(qdot)
30 Hterm_one = p.transpose() * qdot
31 Hterm_two = Lag
32 #print("hterm one")
33 #display(Hterm_one)
34 #display(Hterm_one.subs(qsub_pre))
35 Hamil = Hterm_one[0] - Hterm_two
36 Hamil = sym.simplify(Hamil)
37
38
39 def find_impact_eqs(s):
40
41     z1 = symbols(r"z_1")
42     z2 = symbols(r"z_2")
43     z3 = symbols(r"z_3")
44     z4 = symbols(r"z_4")
45     z5 = symbols(r"z_5")
46     z6 = symbols(r"z_6")
47     subs_dict_q = {x1:s[0],y1:s[1],theta1:s[2],x2:s[3],y2:s[4],theta2:s[5]}
48     fake_dict = {x1dot:z1,y1dot:z2,theta1dot:z3,x2dot:z4,y2dot:z5,theta2dot:z6}
49     #subs_dict_qdot = {theta1dot:s[3],theta2dot:s[4],theta3dot:s[5]}
50     real_dict = {z1:s[6],z2:s[7],z3:s[8],z4:s[9],z5:s[10],z6:s[11]}
51
52     t_s_d_mod = t_s_d.subs(subs_dict_q).subs(fake_dict).subs(real_dict)
53     t_s_b_mod = t_s_b.subs(subs_dict_q).subs(fake_dict).subs(real_dict)
54
55
56
57
58     b1 = (t_s_d * Matrix([w2/2,-l2/2,1]))[0:2]
59     b2 = (t_s_d * Matrix([w2/2,-l2/2,1]))[0:2]
60     b3 = (t_s_d * Matrix([-w2/2,l2/2,1]))[0:2]
61     b4 = (t_s_d * Matrix([w2/2,l2/2,1]))[0:2]
62     wp1 = (t_s_b * Matrix([w1/2,l1/2,1]))[0:2]
63     wp2 = (t_s_b * Matrix([-w1/2,-l1/2,1]))[0:2]
64     wp3 = (t_s_b * Matrix([-w1/2,l1/2,1]))[0:2]
65     wp4 = (t_s_b * Matrix([w1/2,-l1/2,1]))[0:2]
66
67     block_p1 = (t_s_d_mod * Matrix([-w2/2,-l2/2,1]))[0:2]
68     block_p2 = (t_s_d_mod * Matrix([w2/2,-l2/2,1]))[0:2]
69     block_p3 = (t_s_d_mod * Matrix([-w2/2,l2/2,1]))[0:2]
70     block_p4 = (t_s_d_mod * Matrix([w2/2,l2/2,1]))[0:2]
71     weapon_p1 = (t_s_b_mod * Matrix([w1/2,l1/2,1]))[0:2]
72     weapon_p2 = (t_s_b_mod * Matrix([-w1/2,-l1/2,1]))[0:2]
73     weapon_p3 = (t_s_b_mod * Matrix([-w1/2,l1/2,1]))[0:2]
74     weapon_p4 = (t_s_b_mod * Matrix([w1/2,-l1/2,1]))[0:2]
75
76     full_dist = np.array([minDistance(block_p1,block_p2,weapon_p1),minDistance(block_p1,block_p2,weapon_p2),minDistance(block_p1,block_p2,weapon_p3),minDistance(block_p1,block_p2,weapon_p4),
77                             minDistance(block_p2,block_p3,weapon_p1),minDistance(block_p2,block_p3,weapon_p2),minDistance(block_p2,block_p3,weapon_p3),minDistance(block_p2,block_p3,weapon_p4),
78                             minDistance(block_p3,block_p4,weapon_p1),minDistance(block_p3,block_p4,weapon_p2),minDistance(block_p3,block_p4,weapon_p3),minDistance(block_p3,block_p4,weapon_p4),

```

```

79         minDistance(block_p4,block_p1,weapon_p1),minDistance(block_p4,block_p1,weapon_p2),minDistance(block_p4,block_p1,weapon_p3),minDistance(block_p4,block_p1,weapon_p4)
80
81     points_lookup = np.array([ [block_p1,block_p2,weapon_p1],[block_p1,block_p2,weapon_p2],[block_p1,block_p2,weapon_p3],[block_p1,block_p2,weapon_p4],
82                               [block_p2,block_p3,weapon_p1],[block_p2,block_p3,weapon_p2],[block_p2,block_p3,weapon_p3],[block_p2,block_p3,weapon_p4],
83                               [block_p3,block_p4,weapon_p1],[block_p3,block_p4,weapon_p2],[block_p3,block_p4,weapon_p3],[block_p3,block_p4,weapon_p4],
84                               [block_p4,block_p1,weapon_p1],[block_p4,block_p1,weapon_p2],[block_p4,block_p1,weapon_p3],[block_p4,block_p1,weapon_p4]])
85     var_lookup = np.array([ [b1,b2,wp1],[b1,b2,wp2],[b1,b2,wp3],[b1,b2,wp4],
86                             [b2,b3,wp1],[b2,b3,wp2],[b2,b3,wp3],[b2,b3,wp4],
87                             [b3,b4,wp1],[b3,b4,wp2],[b3,b4,wp3],[b3,b4,wp4],
88                             [b4,b1,wp1],[b4,b1,wp2],[b4,b1,wp3],[b4,b1,wp4]])
89
90     fullest_dist = np.argmin(full_dist)
91     print("fullest dist position")
92     display(fullest_dist)
93     fullest_points = points_lookup[fullest_dist]
94     print("closest point")
95     display(fullest_points)
96
97     phimat = full_dist[fullest_dist]
98     #print("PHIMAT")
99     #display(phimat)
100    fullest_vars = var_lookup[fullest_dist]
101    #print("fullest vars")
102    #display(fullest_vars[0][1])
103
104    phivars = line_point_dist(fullest_vars[0],fullest_vars[1],fullest_vars[2])
105    #display(phivars)
106    phidot = phivars.diff(t)
107    #display(phidot)
108    phidiff = phidot.subs(subs_dict_q).subs(fake_dict).subs(real_dict)
109    #print("phidot")
110    #display(phidot)
111
112    qsub_pre = {x1:x1_pre,y1:y1_pre,theta1:theta1_pre,x2:x2_pre,y2:y2_pre,theta2:theta2_pre, x1dot:x1_predot,y1dot:y1_predot,theta1dot:theta1_predot,x2dot:x2_predot,y2dot:y2_predot,theta2dot:theta2_predot}
113    qsub_post = {x1:x1_post,y1:y1_post,theta1:theta1_post,x2:x2_post,y2:y2_post,theta2:theta2_post, x1dot:x1_postdot,y1dot:y1_postdot,theta1dot:theta1_postdot,x2dot:x2_postdot,y2dot:y2_postdot,theta2dot:thet.
114    dLdqdot = Lag.diff(qdot)
115    dLdqdot_pre = dLdqdot.subs(qsub_pre)
116    dLdqdot_post = dLdqdot.subs(qsub_post)
117
118
119    #dphidq = phimat.jacobian(q)
120    #print("dphidq")
121    #display(dphidq)
122    dphidq = phivars.diff(q)
123    #print("dphidq")
124    #display(dphidq)
125    dphidq_pre = dphidq.subs(qsub_pre)
126    dphidq_post = dphidq.subs(qsub_post)
127
128    Hamil_pre = Hamil.subs(qsub_pre)
129    Hamil_post = Hamil.subs(qsub_post)
130    #print("\n Hamiltonian: ")
131    #display(Hamil_pre)
132    #display(Hamil_post)
133    #print(Hamil_pre)
134
135    from sympy import factor
136    #display(dLdqdot_post[0])

```

```

137 impact_rhs = Matrix([dLdqdot_post[0]-dLdqdot_pre[0],dLdqdot_post[1]-dLdqdot_pre[1],dLdqdot_post[2]-dLdqdot_pre[2],dLdqdot_post[3]-dLdqdot_pre[3],dLdqdot_post[4]-dLdqdot_pre[4],dLdqdot_post[5]-dLdqdot_pre
138
139 impact_lhs = Matrix([(lam * dphidq_pre[0]),(lam * dphidq_pre[1]),(lam * dphidq_pre[2]),(lam * dphidq_pre[3]),(lam * dphidq_pre[4]),(lam * dphidq_pre[5]),0])
140 #display(impact_rhs)
141 #display(impact_lhs)
142 #print("THER YA GO")
143 impact_eqs = sym.Eq(impact_lhs,impact_rhs)
144 #display(impact_eqs)
145 return impact_eqs,phivars,phimat,phidiff
146
147 s0 = np.array([0,0,np.pi/8,0,2.5,0,0,4,10,0,0,0])
148 impact_eqs,phivars,phimat,phidiff = find_impact_eqs(s0)
149 print(phidiff)
150
    fullest dist position
    0
    closest point
    array([[ -0.5000000000000000,  2.0000000000000000],
           [ 0.5000000000000000,  2.0000000000000000],
           [-0.0758567746186340,  0.509775195301280]], dtype=object)
    0.287928387309317*Derivative(0.0, t) + 0.293557499746779*Derivative(0.392699081698724, t)

```

▼ Impact Update

```

1 def impact_update_battlebot(x1pre,y1pre,theta1pre,x1dotpre,y1dotpre,theta1dotpre,x2pre,y2pre,theta2pre,x2dotpre,y2dotpre,theta2dotpre):
2     #calls some thing to deterine which impact eqs based on s
3     s = [x1pre,y1pre,theta1pre,x1dotpre,y1dotpre,theta1dotpre,x2pre,y2pre,theta2pre,x2dotpre,y2dotpre,theta2dotpre]
4
5     impact_eqs,vareq,dumm1,dummy2 = find_impact_eqs(s)
6     print("Impact Equations")
7     display(impact_eqs)
8
9     first_dict = {x1_predot:x1dotpre,y1_predot:y1dotpre,theta1_predot:theta1dotpre,x2_predot:x2dotpre,"y_2prd":y2dotpre, "theta_2prd":theta2dotpre }
10    subs_dict = {x1_pre:x1pre,y1_pre:y1pre,theta1_pre:theta1pre, x2_pre:x2pre,"y_2pr":y2pre,theta2_pre:theta2pre ,x1_post:x1pre,y1_post:y1pre, theta1_post:theta1pre ,x2_post:x2pre,y2_post : y2pre, theta2_
11    cleaned = impact_eqs.subs(first_dict).subs(subs_dict)
12
13    #display(cleaned.rhs)
14    #display(cleaned.lhs)
15    #impact_test = np.array([s_test[0],test])
16    #qpost1,qpost2,qpost3,
17    #display(cleaned)
18    impact_sol = sym.solve(cleaned,[x1_postdot,y1_postdot,theta1_postdot,x2_postdot,y2_postdot,theta2_postdot,lam],dict = True)
19
20    #sol = [impact_sol[x1_postdot],impact_sol[y1_postdot],impact_sol[theta1_postdot],impact_sol[x2_postdot],impact_sol[y2_postdot],impact_sol[theta2_postdot]]
21
22    if len(impact_sol) == 1:
23        print("DISAPPOINTMENT")
24        return False
25    else:
26        for solu in impact_sol:
27            lamsol = solu[lam]
28            if abs(lamsol) < 0.001:
29                print("OH NO")
30
31

```

```

30
31     else:
32         print("yay")
33         impact_sol = solu
34         sol = [-impact_sol[x1_postdot], -impact_sol[y1_postdot], -impact_sol[theta1_postdot], impact_sol[x2_postdot], impact_sol[y2_postdot], impact_sol[theta2_postdot]]
35         return sol
36
37 s = [0,0,0,0,0,0,0,0,0.5,0,0,0,0]
38 print(impact_update_battlebot(s[0],s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10],s[11]))
39 #print(impact_update_triple_pend(0.00842, 1.52631626131906*10-1000 , -0.04365, -1.162, -0.440 , 4.973))

```

fullest dist position

1

closest point

```

array([[ -0.500000000000000, -0.500000000000000],
       [ 0.500000000000000, -0.500000000000000],
       [-0.125000000000000, -0.500000000000000]], dtype=object)

```

Impact Equations

$$\begin{bmatrix}
 \text{lam}(x_{1pr} - x_{2pr} + 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr})) \\
 0 \\
 \frac{\text{lam}(0.25 \sin(\theta_{1pr}) + \cos(\theta_{1pr}))(x_{1pr} - x_{2pr} + 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr}))}{2} \\
 \text{lam}(-x_{1pr} + x_{2pr} - 0.5 \sin(\theta_{1pr}) + 0.5 \sin(\theta_{2pr}) + 0.125 \cos(\theta_{1pr}) + 0.5 \cos(\theta_{2pr})) \\
 0 \\
 \frac{\text{lam}(\sin(\theta_{2pr}) - 1.0 \cos(\theta_{2pr}))(x_{1pr} - x_{2pr} + 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr}))}{2} \\
 0 \\
 3.0x_{1pod} - 3.0x_{1prd} \\
 3.0y_{1pod} - 3.0y_{1prd} \\
 0.0442708333333333\theta_{1pod} - 0.0442708333333333\theta_{1prd} \\
 0.5x_{2pod} - 0.5x_{2prd} \\
 0.5y_{2pod} - 0.5y_{2prd} \\
 0.5\theta_{2pod} - 0.5\theta_{2prd} \\
 0.0221354166666667\theta_{1pod}^2 - 0.0221354166666667\theta_{1prd}^2 + 0.25\theta_{2pod}^2 - 0.25\theta_{2prd}^2 + 1.5x_{1pod}^2 - 1.5x_{1prd}^2 + 0.25x_{2pod}^2 - 0.25x_{2prd}^2 + 1.5y_{1pod}^2 - 1.5y_{1prd}^2 + 0.25y_{2pod}^2 - 0.25y_{2prd}^2
 \end{bmatrix} =$$

DISAPPOINTMENT

False

▼ Impact Condition

```

1 import numpy as np
2
3 def impact_condition_battlebot(s):
4     '''
5     impact_eqs, vareq, que, qdawt = find_impact_eqs(s)
6     threshold = 0.04
7     # if que < threshold and qdawt > 0:
8     #print(qdawt)
9     if que < threshold:
10        print("UHOH")
11        return True
12
13    else:
14        return False
15
16    '''

```

```

17 z1 = symbols(r"z_1")
18 z2 = symbols(r"z_2")
19 z3 = symbols(r"z_3")
20 z4 = symbols(r"z_4")
21 z5 = symbols(r"z_5")
22 z6 = symbols(r"z_6")
23
24 subs_dict_q = {x1:s[0],y1:s[1],theta1:s[2],x2:s[3],y2:s[4],theta2:s[5]}
25 fake_dict = {x1dot:z1,y1dot:z2,theta1dot:z3,x2dot:z4,y2dot:z5,theta2dot:z6}
26 #subs_dict_qdot = {theta1dot:s[3],theta2dot:s[4],theta3dot:s[5]}
27 real_dict = {z1:s[6],z2:s[7],z3:s[8],z4:s[9],z5:s[10],z6:s[11]}
28
29
30
31 t_s_d_mod = t_s_d.subs(subs_dict_q).subs(fake_dict).subs(real_dict)
32 t_s_b_mod = t_s_b.subs(subs_dict_q).subs(fake_dict).subs(real_dict)
33
34 block_p1 = (t_s_d_mod * Matrix([-w2/2,-l2/2,1]))[0:2]
35
36 block_p2 = (t_s_d_mod * Matrix([w2/2,-l2/2,1]))[0:2]
37 block_p3 = (t_s_d_mod * Matrix([-w2/2,l2/2,1]))[0:2]
38 block_p4 = (t_s_d_mod * Matrix([w2/2,l2/2,1]))[0:2]
39 weapon_p1 = (t_s_b_mod * Matrix([w1/2,l1/2,1]))[0:2]
40 weapon_p2 = (t_s_b_mod * Matrix([-w1/2,-l1/2,1]))[0:2]
41 weapon_p3 = (t_s_b_mod * Matrix([-w1/2,l1/2,1]))[0:2]
42 weapon_p4 = (t_s_b_mod * Matrix([w1/2,-l1/2,1]))[0:2]
43 threshold = 0.05
44
45 points_lookup = np.array([ [block_p1,block_p2,weapon_p1],[block_p1,block_p2,weapon_p2],[block_p1,block_p2,weapon_p3],[block_p1,block_p2,weapon_p4],
46                           [block_p2,block_p3,weapon_p1],[block_p2,block_p3,weapon_p2],[block_p2,block_p3,weapon_p3],[block_p2,block_p3,weapon_p4],
47                           [block_p3,block_p4,weapon_p1],[block_p3,block_p4,weapon_p2],[block_p3,block_p4,weapon_p3],[block_p3,block_p4,weapon_p4],
48                           [block_p4,block_p1,weapon_p1],[block_p4,block_p1,weapon_p2],[block_p4,block_p1,weapon_p3],[block_p4,block_p1,weapon_p4]])
49 full_dist = np.array([minDistance(block_p1,block_p2,weapon_p1),minDistance(block_p1,block_p2,weapon_p2),minDistance(block_p1,block_p2,weapon_p3),minDistance(block_p1,block_p2,weapon_p4),
50                       minDistance(block_p2,block_p3,weapon_p1),minDistance(block_p2,block_p3,weapon_p2),minDistance(block_p2,block_p3,weapon_p3),minDistance(block_p2,block_p3,weapon_p4),
51                       minDistance(block_p3,block_p4,weapon_p1),minDistance(block_p3,block_p4,weapon_p2),minDistance(block_p3,block_p4,weapon_p3),minDistance(block_p3,block_p4,weapon_p4),
52                       minDistance(block_p4,block_p1,weapon_p1),minDistance(block_p4,block_p1,weapon_p2),minDistance(block_p4,block_p1,weapon_p3),minDistance(block_p4,block_p1,weapon_p4)])
53 numtrues = 0
54
55 looking = np.zeros((16,1))
56 #print(looking[4])
57 for i in range(15):
58     looking[i] = full_dist[i]
59     if full_dist[i] < threshold:
60         numtrues = numtrues + 1
61
62
63 smol = np.argmin(looking)
64 look = points_lookup[smol]
65 #print(np.shape(look))
66 #print(look)
67 if numtrues == 0:
68     return False,look
69 else:
70     return True,look
71
72 '''
73 if minDistance(block_p1,block_p2,weapon_p1) < threshold:
74     return True

```

```
75 elif minDistance(block_p1,block_p2,weapon_p2) < threshold:
76     return True
77 elif minDistance(block_p1,block_p2,weapon_p3) < threshold:
78     return True
79 elif minDistance(block_p1,block_p2,weapon_p4) < threshold:
80     return True
81 elif minDistance(block_p2,block_p3,weapon_p1) < threshold:
82     return True
83 elif minDistance(block_p2,block_p3,weapon_p2) < threshold:
84     return True
85 elif minDistance(block_p2,block_p3,weapon_p3) < threshold:
86     return True
87 elif minDistance(block_p2,block_p3,weapon_p4) < threshold:
88     return True
89 elif minDistance(block_p3,block_p4,weapon_p1) < threshold:
90     return True
91 elif minDistance(block_p3,block_p4,weapon_p2) < threshold:
92     return True
93 elif minDistance(block_p3,block_p4,weapon_p3) < threshold:
94     return True
95 elif minDistance(block_p3,block_p4,weapon_p4) < threshold:
96     return True
97 elif minDistance(block_p4,block_p1,weapon_p1) < threshold:
98     return True
99 elif minDistance(block_p4,block_p1,weapon_p2) < threshold:
100    return True
101 elif minDistance(block_p4,block_p1,weapon_p3) < threshold:
102    return True
103 elif minDistance(block_p4,block_p1,weapon_p4) < threshold:
104    return True
105 else:
106     return False
107
108 '''
109 '''
110 phi_clean = phimat.subs(subs_dict_q)
111
112 phi_clean = phi_clean[0]
113 phidot_clean = phi_clean.subs(fake_dict)
114 phidot_clean = phidot_clean.subs(subs_dict_q)
115 phidot_clean = phidot_clean.subs(real_dict)
116 qdawt = phidot_clean
117 phidot = phimat.diff(t)
118 using = phidot
119 #using = x3dot
120 #use dphidq here?
121 phidot_clean = using.subs(fake_dict)
122
123 phidot_clean = phidot_clean.subs(subs_dict_q)
124 phidot_clean = phidot_clean.subs(real_dict)
125 '''
126 #que = phi_clean
127
128
129 #print(que)
130 #print(qdawt)
131
132 #print(qdawt<0)
```

```

133 #impact = phi.subs(theta,que)
134
135
136

```

```

1

```

```

1
2 def integrate(f, xt, dt):
3     k1 = dt * f(xt)
4     k2 = dt * f(xt+k1/2.)
5     k3 = dt * f(xt+k2/2.)
6     k4 = dt * f(xt+k3)
7     new_xt = xt + (1/6.) * (k1+2.0*k2+2.0*k3+k4)
8     return new_xt

```

▼ Impact Simulation Func

```

1 from sympy import evalf
2 def simulate_impact(f, x0, tspan, dt, integrate):
3     """
4     This function takes in an initial condition x0, a timestep dt,
5     a time span tspan consisting of a list [min_time, max_time],
6     as well as a dynamical system f(x) that outputs a vector of the
7     same dimension as x0. It outputs a full trajectory simulated
8     over the time span of dimensions (xvec_size, time_vec_size).
9
10    Parameters
11    =====
12    f: Python function
13        derivate of the system at a given step x(t),
14        it can considered as  $\dot{x}(t) = \text{func}(x(t))$ 
15    x0: NumPy array
16        initial conditions
17    tspan: Python list
18        tspan = [min_time, max_time], it defines the start and end
19        time of simulation
20    dt:
21        time step for numerical integration
22    integrate: Python function
23        numerical integration method used in this simulation
24
25    Return
26    =====
27    x_traj:
28        simulated trajectory of x(t) from t=0 to tf
29    """
30    N = int((max(tspan)-min(tspan))/dt)
31    x = np.copy(x0)
32    tvec = np.linspace(min(tspan),max(tspan),N)
33    xtraj = np.zeros((len(x0),N))
34

```

```

35     for i in range(N-1):
36         if i % 10 == 0:
37             print(i)
38         '''
39
40     xtraj[:,i]=integrate(f,x,dt)
41     x = np.copy(xtraj[:,i])
42     '''
43
44     check = integrate(f,x,dt)
45     check_imp, dummy = impact_condition_battlebot(check)
46     Looking_Points[i] = dummy
47
48     Hamilclean = Hamil.subs({x1dot:check[6],y1dot:check[7],theta1dot:check[8],x2dot:check[9],y2dot:check[10],theta2dot:check[11]})
49     Hamilray[:,i] = [i,Hamilclean.evalf()]
50
51     if check_imp == True:
52
53         s_test = check
54         print("IMPACT")
55         #print(check)
56         test = impact_update_battlebot(check[0],check[1],check[2],check[3],check[4],check[5],check[6],check[7],check[8],check[9],check[10],check[11])
57         print(test)
58         if test == None:
59             print("Rats")
60         elif test != False:
61
62             #print("TEST")
63             #rint(test)
64             impact_test = np.array([s_test[0],s_test[1],s_test[2],s_test[3],s_test[4],s_test[5],test[0],test[1],test[2],test[3],test[4],test[5]])
65             check = impact_test
66
67         #print(check)
68         xtraj[:,i] = check
69         x = np.copy(xtraj[:,i])
70
71     return xtraj

```

▼ Simulate it

```

1
2
3 #test ddots at initial conditions
4 #print('theta1ddot = ', theta1dub(-1.57,-1.57,0,0))
5 #print('theta2ddot = ', theta2dub(-1.57,-1.57,0,0))
6
7 def dyn(s):
8
9     thing1 = x1dub(s[0],s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10] , s[11])
10    thing2 = y1dub(s[0],s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10] , s[11])
11    thing3 = theta1dub(s[0],s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10] , s[11])
12    thing4 = x2dub(s[0],s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10] , s[11])
13    thing5 = y2dub(s[0],s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10] , s[11])
14    thing6 = theta2dub(s[0],s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10] , s[11])
15

```

```
16     thing = np.array([ s[6],s[7],s[8],s[9],s[10] , s[11] ,thing1,thing2,thing3,thing4,thing5,thing6])
17     return thing
18
19 #s0 = np.array([0,0,np.pi/3,4,4,0,2,2,1,0,0,0]) # spinny case,angle strange
20 #s0 = np.array([1,0,np.pi/8,0,2.5,np.pi/8,-0.33,2,15,0,0,0]) # head on certical
21 #s0 = np.array([0,0,np.pi/8,0.5,2.5,0,.5,2.5,10,0,0,0])###FORMAT: XYTH1 XYTH2 XYTH_DOT_1 XYTH_DOT_2
22 s0 = np.array([0,0,0,1.8,0,np.pi/6,0.8,0,12,0,0,12])# sideways spinner
23
24
25
26 timestep = 0.01
27 steps_sec = 1/timestep
28 end_time = 3
29 steps = end_time/timestep
30
31 Hamilray = np.zeros((2,int(steps)))
32
33 Looking_Points = np.zeros((int(steps),3,2))
34
35 traj = simulate_impact(dyn, s0, [0, end_time], timestep, integrate)
36 #plt.plot(np.arange(steps)*timestep,traj[2].T)
37 #plt.plot(np.arange(steps)*timestep,traj[5].T)
38 plt.plot(Hamilray.T)
39 #print(Hamilray[0])
40 #print(Hamilray[1][0:int(steps)])
41 #plt.plot((Hamil[0][0:int(steps)]),(Hamil[1][0:int(steps)]))
42 plt.title("Simulated Trajectories")
43 plt.xlabel("time")
44 plt.ylabel("Angle")
45 plt.legend(('theta1(t)'),loc= 'upper right')
46 #plt.show()
47 print("Formatting")
```



0
10
20
30
40
50
60
70
80

IMPACT

fullest dist position

2

closest point

```
array([[1.16557293635718, 0.312253584314967],
       [1.48774641568503, -0.634427063642824],
       [1.23475935334016, 0.0254520755306376]], dtype=object)
```

Impact Equations

$$\begin{bmatrix} lam(x_{1pr} - x_{2pr} - 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr})) \\ 0 \\ \frac{lam(0.25 \sin(\theta_{1pr}) - 1.0 \cos(\theta_{1pr}))(x_{1pr} - x_{2pr} - 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr}))}{2} \\ lam(-x_{1pr} + x_{2pr} + 0.5 \sin(\theta_{1pr}) + 0.5 \sin(\theta_{2pr}) + 0.125 \cos(\theta_{1pr}) + 0.5 \cos(\theta_{2pr})) \\ 0 \\ \frac{lam(\sin(\theta_{2pr}) - 1.0 \cos(\theta_{2pr}))(x_{1pr} - x_{2pr} - 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr}))}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 3.0x_{1pod} - 3.0x_{1prd} \\ 3.0y_{1pod} - 3.0y_{1prd} \\ 0.0442708333333333\theta_{1pod} - 0.0442708333333333\theta_{1prd} \\ 0.5x_{2pod} - 0.5x_{2prd} \\ 0.5y_{2pod} - 0.5y_{2prd} \\ 0.5\theta_{2pod} - 0.5\theta_{2prd} \\ 0.0221354166666667\theta_{1pod}^2 - 0.0221354166666667\theta_{1prd}^2 + 0.25\theta_{2pod}^2 - 0.25\theta_{2prd}^2 + 1.5x_{1pod}^2 - 1.5x_{1prd}^2 + 0.25x_{2pod}^2 - 0.25x_{2prd}^2 + 1.5y_{1pod}^2 - 1.5y_{1prd}^2 + 0.25y_{2pod}^2 - 0.25y_{2prd}^2 \end{bmatrix}$$

OH NO

yay

```
[-3.16751937439349, 0, -8.96496643556960, -8.20511624636092, 0.0, 6.33671850340271]
```

90

IMPACT

fullest dist position

2

closest point

```
array([[1.06502169555613, 0.271451924409251],
       [1.44649691312714, -0.652927141980263],
       [1.20329568296575, -0.0207363483010927]], dtype=object)
```

Impact Equations

$$\begin{bmatrix} lam(x_{1pr} - x_{2pr} - 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr})) \\ 0 \\ \frac{lam(0.25 \sin(\theta_{1pr}) - 1.0 \cos(\theta_{1pr}))(x_{1pr} - x_{2pr} - 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr}))}{2} \\ lam(-x_{1pr} + x_{2pr} + 0.5 \sin(\theta_{1pr}) + 0.5 \sin(\theta_{2pr}) + 0.125 \cos(\theta_{1pr}) + 0.5 \cos(\theta_{2pr})) \\ 0 \\ \frac{lam(\sin(\theta_{2pr}) - 1.0 \cos(\theta_{2pr}))(x_{1pr} - x_{2pr} - 0.5 \sin(\theta_{1pr}) - 0.5 \sin(\theta_{2pr}) - 0.125 \cos(\theta_{1pr}) - 0.5 \cos(\theta_{2pr}))}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 3.0x_{1pod} - 3.0x_{1prd} \\ 3.0y_{1pod} - 3.0y_{1prd} \\ 0.0442708333333333\theta_{1pod} - 0.0442708333333333\theta_{1prd} \\ 0.5x_{2pod} - 0.5x_{2prd} \\ 0.5y_{2pod} - 0.5y_{2prd} \end{bmatrix}$$

$$0.0221354166666667\theta_{1pod}^2 - 0.0221354166666667\theta_{1prd}^2 + 0.25\theta_{2pod}^2 - 0.25\theta_{2prd}^2 + 1.5x_{1pod}^2 - 1.5x_{1prd}^2 + 0.25x_{2pod}^2 - 0.25x_{2prd}^2 + 1.5y_{1pod}^2 - 1.5y_{1prd}^2 + 0.25y_{2pod}^2 - 0.25y_{2prd}^2$$

yay
 [1.44243666109684, 0, -6.94601582274295, 10.7571967454385, 0.0, 2.04785157392523]
 100
 110
 120
 130
 140
 150
 160
 170
 180
 190
 200
 210

ANIMATION

```

---
---
1 def animate_battlebot(theta_array,T=10,xraymode = False):
2     """
3     Function to generate web-based animation of double-pendulum system
4
5     Parameters:
6     =====
7     theta_array:
8         trajectory of theta1 and theta2, should be a NumPy array with
9         shape of (2,N)
10    L1:
11        length of the first pendulum
12    L2:
13        length of the second pendulum
14    T:
15        length/seconds of animation duration
16
17    Returns: None
18    """
19
20    #####
21    # Imports required for animation.
22    from plotly.offline import init_notebook_mode, iplot
23    from IPython.display import display, HTML
24    import plotly.graph_objects as go
25
26    #####
27    # Browser configuration.
28    def configure_plotly_browser_state():
29        import IPython
30        display(IPython.core.display.HTML('''
31        <script src="/static/components/requirejs/require.js"></script>
32        <script>
33            requirejs.config({
34                paths: {
35                    base: '/static/base',
36                    plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
37                },

```

```

38     });
39     </script>
40     '')
41     configure_plotly_browser_state()
42     init_notebook_mode(connected=False)
43
44     #####
45     # Getting data from pendulum angle trajectories.
46     xx1= theta_array[0]
47     yy1= theta_array[1]
48     thetat1 = theta_array[2]
49     xx2= theta_array[3]
50     yy2= theta_array[4]
51     thetat2 = theta_array[5]
52     N = len(theta_array[0]) # Need this for specifying length of simulation
53
54     #####
55     # Define arrays containing data for frame axes
56     # In each frame, the x and y axis are always fixed
57     x_axis = np.array([0.3, 0.0])
58     y_axis = np.array([0.0, 0.3])
59     bump = 0.001
60     b_points_x = np.array([-w1/2-bump,w1/2+bump,w1/2+bump,-w1/2-bump])
61     b_points_y = np.array([-l1/2-bump,-l1/2-bump,l1/2+bump,l1/2+bump])
62     d_points_x = np.array([-w2/2-bump,w2/2+bump,w2/2+bump,-w2/2-bump])
63     d_points_y = np.array([-l2/2-bump,-l2/2-bump,l2/2+bump,l2/2+bump])
64     # Use homogeneous tranformation to transfer these two axes/points
65     # back to the fixed frame
66
67
68     shape_b_x = np.zeros((4,N))
69     shape_b_y = np.zeros((4,N))
70     shape_d_x = np.zeros((4,N))
71     shape_d_y = np.zeros((4,N))
72
73
74     shape_imp_x = np.zeros((3,N))
75     shape_imp_y = np.zeros((3,N))
76
77     #sub_d1 = {theta1dot:theta_array[2][k],theta2dot:theta_array[3][k]}
78
79     for i in range(N): # iteration through each time step
80
81
82
83
84         sub_d2 = {theta1:theta_array[0][i],theta2:theta_array[1][i]}
85         t_s_a = np.array([[1, 0, 1* theta_array[0][i]],
86                         [0, 1, 1* theta_array[1][i]],
87                         [0, 0, 1]])
88
89         t_a_b = np.array([[np.cos(theta_array[2][i]), -np.sin(theta_array[2][i]), 0],
90                         [np.sin(theta_array[2][i]), np.cos(theta_array[2][i]), 0],
91                         [0, 0, 1]])
92         #t_s_b = t_s_a * t_a_b
93         t_s_b = np.dot(t_s_a,t_a_b)
94         #BLOCK
95         t_s_c = np.array([[1, 0, 1 * theta_array[3][i]],

```

```

96         [0, 1, 1 * theta_array[4][i]],
97         [0, 0, 1]])
98
99     t_c_d = np.array([[np.cos(theta_array[5][i]), -np.sin(theta_array[5][i]), 0],
100                    [np.sin(theta_array[5][i]), np.cos(theta_array[5][i]), 0],
101                    [0, 0, 1]])
102
103     t_s_d = np.dot(t_s_c,t_c_d)
104     #t_s_d = t_s_c * t_c_d
105     block_p1 = [-w2/2-bump,-l2/2-bump,1]
106     block_p2 = [w2/2+bump,-l2/2-bump,1]
107     Weapon_p1 = [w1/2+bump,l1/2+bump,1]
108
109     if xraymode == True:
110         point1 = Looking_Points[i][0]
111         point2 = Looking_Points[i][1]
112         point3 = Looking_Points[i][2]
113     else:
114
115         point1 = t_s_b.dot([Weapon_p1[0],Weapon_p1[1],Weapon_p1[2]])
116         point2 = t_s_d.dot([block_p1[0],block_p1[1],block_p1[2]])
117         point3 = t_s_d.dot([block_p2[0],block_p2[1],block_p2[2]])
118
119     shape_imp_x[:,i] = [point1[0],point2[0],point3[0]]
120     shape_imp_y[:,i] = [point1[1],point2[1],point3[1]]
121     #print(shape_imp_x[:,i])
122     for j in range(4):
123         points = t_s_b.dot([b_points_x[j],b_points_y[j] ,1])[0:2]
124         shape_b_x[j,i] = points[0]
125         shape_b_y[j,i] = points[1]
126
127     for j in range(4):
128         points = t_s_d.dot([d_points_x[j],d_points_y[j], 1])[0:2]
129         shape_d_x[j,i] = points[0]
130         shape_d_y[j,i] = points[1]
131
132     # transfer the x and y axes in body frame back to fixed frame at
133     # the current time step
134     '''
135     shape_b_x[:,i] = t_wb.dot([x_axis[0], x_axis[1], 1])[0:2]
136     frame_b_y_axis[:,i] = t_wb.dot([y_axis[0], y_axis[1], 1])[0:2]
137     frame_d_x_axis[:,i] = t_wd.dot([x_axis[0], x_axis[1], 1])[0:2]
138     frame_d_y_axis[:,i] = t_wd.dot([y_axis[0], y_axis[1], 1])[0:2]
139     '''
140     #print(t_wb.dot([y_axis[0], y_axis[1], 1])[0:2])
141
142
143     #####
144     # Using these to specify axis limits.
145     xm = -5 #np.min(xx1)-0.5
146     xM = 5 #np.max(xx1)+0.5
147     ym = -3 #np.min(yy1)-2.5
148     yM = 5#np.max(yy1)+1.5
149
150
151     print("FINALLY")
152
153

```

```

154 #####
155 # Defining data dictionary.
156 # Trajectories are here.
157 data=[
158     # note that except for the trajectory (which you don't need this time),
159     # you don't need to define entries other than "name". The items defined
160     # in this list will be related to the items defined in the "frames" list
161     # later in the same order. Therefore, these entries can be considered as
162     # labels for the components in each animation frame
163     dict(name='Weapon Bar'),
164     dict(name='Block'),
165     dict(name='Weapon Center'),
166     dict(name = 'Impact Points')
167
168
169     # You don't need to show trajectory this time,
170     # but if you want to show the whole trajectory in the animation (like what
171     # you did in previous homeworks), you will need to define entries other than
172     # "name", such as "x", "y". and "mode".
173
174     # dict(x=xx1, y=yy1,
175     #     mode='markers', name='Pendulum 1 Traj',
176     #     marker=dict(color="fuchsia", size=2)
177     # ),
178     # dict(x=xx2, y=yy2,
179     #     mode='markers', name='Pendulum 2 Traj',
180     #     marker=dict(color="purple", size=2)
181     # ),
182     ]
183
184 #####
185 # Preparing simulation layout.
186 # Title and axis ranges are here.
187 layout=dict(autosize=False, width=1000, height=1000,
188             xaxis=dict(range=[xm, xM], autorange=False, zeroline=False,dtick=1),
189             yaxis=dict(range=[ym, yM], autorange=False, zeroline=False,scaleanchor = "x",dtick=1),
190             title='Double Pendulum Simulation',
191             hovermode='closest',
192             updatemenus= [{'type': 'buttons',
193                           'buttons': [{'label': 'Play','method': 'animate',
194                                       'args': [None, {'frame': {'duration': T, 'redraw': False}}]},
195                                       {'args': [[None], {'frame': {'duration': T, 'redraw': False}, 'mode': 'immediate',
196                                       'transition': {'duration': 0}}], 'label': 'Pause','method': 'animate'}
197                           ]
198             })
199
200
201 #####
202 # Defining the frames of the simulation.
203 # This is what draws the lines from
204 # joint to joint of the pendulum.
205 #print(shape_b_y)
206 frames=[dict(data=[# first three objects correspond to the arms and two masses,
207                  # same order as in the "data" variable defined above (thus
208                  # they will be labeled in the same order)
209                  go.Scatter(
210                      x=[shape_b_x[0,k],shape_b_x[1,k],shape_b_x[2,k],shape_b_x[3,k]],
211                      y=[shape_b_y[0,k],shape_b_y[1,k],shape_b_y[2,k],shape_b_y[3,k]],

```

```
212         fill="toself",
213         marker=dict(color="blue", size=12)),
214     go.Scatter(
215         x=[xx2[k]],
216         y=[yy2[k]],
217         mode="markers",
218         marker=dict(color="orange", size=12)),
219     go.Scatter(
220         x=[shape_d_x[0,k],shape_d_x[1,k],shape_d_x[2,k],shape_d_x[3,k]],
221         y=[shape_d_y[0,k],shape_d_y[1,k],shape_d_y[2,k],shape_d_y[3,k]],
222         fill= "toself",
223         marker=dict(color="red", size=12)),
224     go.Scatter(
225         x=[shape_imp_x[0,k],shape_imp_x[1,k],shape_imp_x[2,k]],
226         y=[shape_imp_y[0,k],shape_imp_y[1,k],shape_imp_y[2,k]],
227         mode="markers",
228         marker=dict(color="green", size=12)),
229
230     ]) for k in range(N)]
231
232     #####
233     # Putting it all together and plotting.
234     figure1=dict(data=data, layout=layout, frames=frames)
235     iplot(figure1)
```

```
1
▶ 2 animate_battlebot(traj,3,True)
```

FINALLY

▶ Executing (17s) (> animat... > ... > return_figure_f... > ... > ... > valid... > ... > _s... > _set_co... > valid... > ... > _s... > _s... > _set_co... > valid... > ... > _s... > _set_co... > valid... > ... > _s... > _set_co... > valid... > ... > _s... > _set_co... > valid... > ... > _i... ⋮ ✕