

```

1  # LQR optimal controller
2
3  # Import libraries
4  import numpy as np
5  from base_controller import BaseController
6  from lqr_solver import dlqr, lqr
7  from scipy.linalg import solve_continuous_lyapunov
   , solve_lyapunov, solve_discrete_lyapunov
8  from math import cos, sin
9  import numpy as np
10 from scipy import signal
11
12 class LQRController(BaseController):
13     """ The LQR controller class.
14
15     """
16
17     def __init__(self, robot, lossOfThrust=0):
18         """ LQR controller __init__ method.
19
20         Initialize parameters here.
21
22         Args:
23             robot (webots controller object):
24                 Controller for the drone.
25             lossOfThrust (float): percent lost of
26                 thrust.
27
28         """
29         super().__init__(robot, lossOfThrust)
30
31         # define integral error
32         self.int_e1 = 0
33         self.int_e2 = 0
34         self.int_e3 = 0
35         self.int_e4 = 0
36
37         # define K matrix
38         self.K = None
39
40     def initializeGainMatrix(self):

```

```

39         """ Calculate the gain matrix.
40
41         """
42
43         # -----/LQR Controller
44         |-----
45         # Use the results of linearization to
46         create a state-space model
47
48         # Given parameters
49         n_p = 12 # number of states
50         m = 4 # number of integral error terms
51
52         # robot parameter
53         self.m = 0.4
54         self.d1x = 0.1122
55         self.d1y = 0.1515
56         self.d2x = 0.11709
57         self.d2y = 0.128
58         self.Ix = 0.000913855
59         self.Iy = 0.00236242
60         self.Iz = 0.00279965
61
62         # constants
63         self.g = 9.81
64         self.ct = 0.00026
65         self.ctau = 5.2e-06
66         self.U1_max = 10
67         self.pi = 3.1415926535
68
69         # ----- Your Code Here
70         ----- #
71         # Compute the discretized A_d, B_d, C_d,
72         D_d, for the computation of LQR gain
73
74         # Matrix A logic
75         # Initialize A matrix with zeros ( 16 x 16
76         )
77         A = np.zeros((n_p+m, n_p+m))
78         A[0, 6] = 1; A[1, 7] = 1; A[2, 8] = 1; A[3
79         , 9] = 1; A[4, 10] = 1; A[5, 11] = 1

```

```

74         A[6, 4] = self.g; A[7, 3] = -self.g
75         # A[12, 0] = 1; A[12, 12] = -1; A[13, 1
    ] = 1; A[13, 13] = -1; A[14, 2] = 1; A[14, 14] = -
    1; A[15, 5] = 1; A[15, 15] = -1
76         A[12, 0] = 1; A[13, 1] = 1; A[14, 2] = 1;
    A[15, 5] = 1
77
78         # Matrix B logic
79         # Initialize B matrix with zeros ( 16 x 4
    )
80         B = np.zeros((n_p+m, m))
81         B[8, 0] = 1/self.m; B[9, 1] = 1/self.Ix; B
    [10, 2] = 1/self.Iy; B[11, 3] = 1/self.Iz
82
83         # Matrix C logic
84         # Initialize C matrix with zeros ( 4 x 16
    )
85         C = np.zeros ((m, n_p+m))
86         C[0, 0] = 1; C[1, 1] = 1; C[2, 2] = 1; C[3
    , 3] = 1
87
88         # Matrix D logic
89         # Zero matrix ( 4 x 4 )
90         D = np.zeros((m, m))
91
92         # Discretize the system
93         sys_discrete = signal.cont2discrete((A, B
    , C, D), self.delT, method='foh')
94
95         # Extract A_d, B_d, C_d, D_d
96         A_d = sys_discrete[0]
97         B_d = sys_discrete[1]
98         C_d = sys_discrete[2]
99         D_d = sys_discrete[3]
100
101         # ----- Your Code Ends Here
    ----- #
102
103         # ----- Example code
    ----- #
104         # max_pos = 15.0

```

```

105         # max_ang = 0.2 * self.pi
106         # max_vel = 6.0
107         # max_rate = 0.015 * self.pi
108         # max_eyI = 3.
109
110         # max_states = np.array([0.1 * max_pos, 0.
111         1 * max_pos, max_pos,
112         #                                     max_ang, max_ang,
113         max_ang,
114         #                                     0.5 * max_vel, 0.5
115         * max_vel, max_vel,
116         #                                     max_rate, max_rate,
117         max_rate,
118         #                                     0.1 * max_eyI, 0.1
119         * max_eyI, 1 * max_eyI, 0.1 * max_eyI])
120
121         # max_inputs = np.array([0.2 * self.U1_max
122         , self.U1_max, self.U1_max, self.U1_max])
123
124         # Q = np.diag(1/max_states**2)
125         # R = np.diag(1/max_inputs**2)
126         # ----- Example code Ends
127         ----- #
128         # ----- Your Code Here
129         ----- #
130         # Come up with reasonable values for Q and
131         R (state and control weights)
132         # The example code above is a good
133         starting point, feel free to use them or write you
134         own.
135         # Tune them to get the better performance
136
137         # referencing the example code above
138         max_pos = 15.0
139         max_ang = 0.2 * self.pi
140         max_vel = 6.0
141         max_rate = 0.015 * self.pi
142         # max_eyI = 0.75
143         max_eyI = 3.0
144
145         max_states = np.array([0.1 * max_pos, 0.1

```

```

134 * max_pos, max_pos,
135                                     max_ang, max_ang,
    max_ang,
136                                     0.5 * max_vel, 0.5
    * max_vel, max_vel,
137                                     max_rate, max_rate
    , max_rate,
138                                     0.1 * max_eyI, 0.1
    * max_eyI, 1 * max_eyI, 0.1 * max_eyI])
139
140     max_inputs = np.array([0.2 * self.U1_max,
    self.U1_max, self.U1_max, self.U1_max])
141
142     Q = np.diag(1 / max_states ** 2)
143     R = np.diag(1 / max_inputs ** 2)
144
145     # ----- Your Code Ends Here
    ----- #
146
147     # solve for LQR gains
148     [K, _, _] = dlqr(A_d, B_d, Q, R)
149
150     self.K = -K
151
152     def update(self, r):
153         """ Get current states and calculate
    desired control input.
154
155         Args:
156             r (np.array): reference trajectory.
157
158         Returns:
159             np.array: states. information of the
    16 states.
160             np.array: U. desired control input.
161
162         """
163
164         # Fetch the states from the BaseController
    method
165         x_t = super().getStates()

```

```
166
167     # update integral term
168     self.int_e1 += float((x_t[0]-r[0])*(self.
delT))
169     self.int_e2 += float((x_t[1]-r[1])*(self.
delT))
170     self.int_e3 += float((x_t[2]-r[2])*(self.
delT))
171     self.int_e4 += float((x_t[5]-r[3])*(self.
delT))
172
173     # Assemble error-based states into array
174     error_state = np.array([self.int_e1, self.
int_e2, self.int_e3, self.int_e4]).reshape((-1,1))
175     states = np.concatenate((x_t, error_state
))
176
177     # calculate control input
178     U = np.matmul(self.K, states)
179     U[0] += self.g * self.m
180
181     # Return all states and calculated control
inputs U
182     return states, U
```