Project: Part 5

24-677 Special Topics: Modern Control - Theory and Design

Ryan Wu (ID: weihuanw)

Due: Dec 8, 2023, 11:59 pm

Exercise 1.

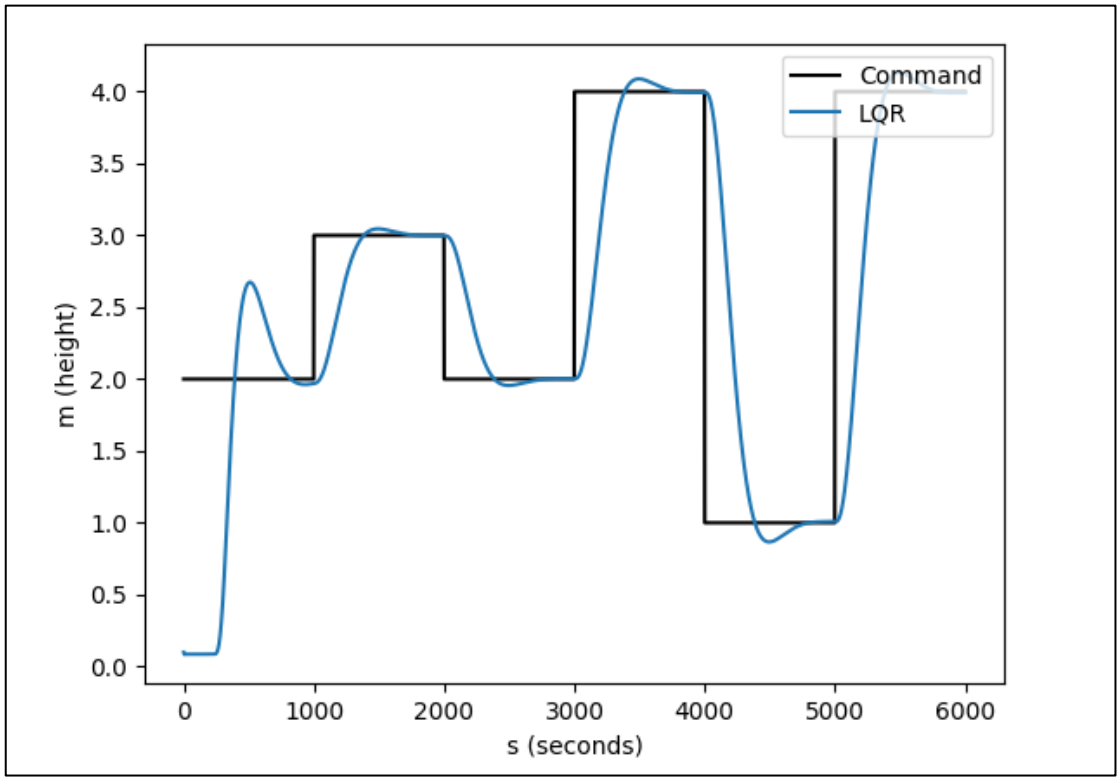

Figure 1. The completed simulation plot showing a stable system.

```
Time: 59.99
[69.34037917 69.45462241 68.93786665 69.08206459]
Time: 60.0
==============YOUR RESULT==============
ERROR: 0.179
SCORE: 50.000
INFO: 'ex1_controller' controller exited successfully.
```

Figure 2. The Webots terminal output for error (0.179) and score (50).

Exercise 2.

Exercise 2.1 (50% loss of thrust in one motor)

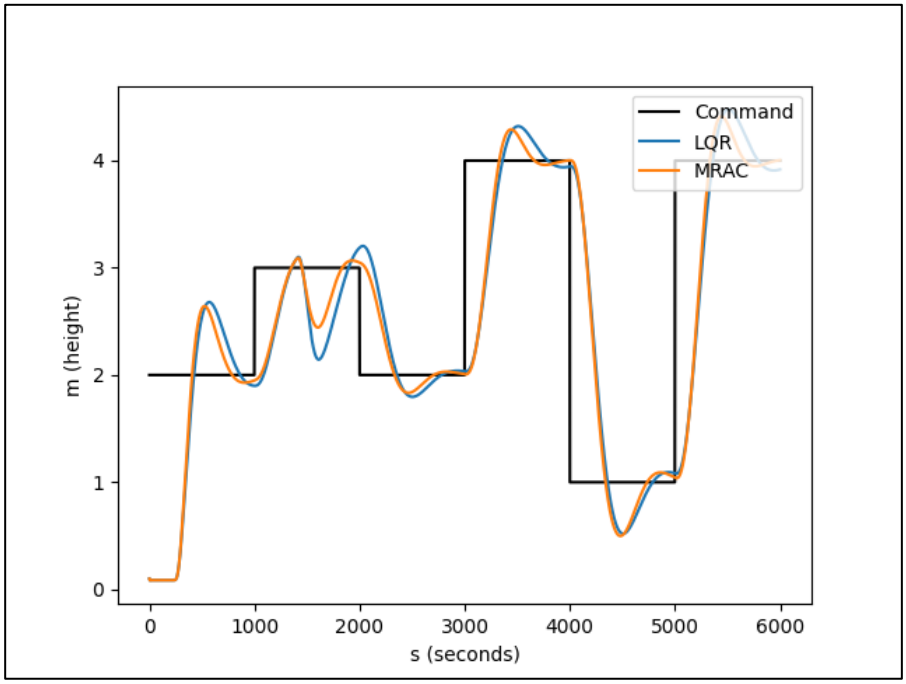

Figure 3. The LQR and MRAC result plot with 50% loss of motor thrust.

Exercise 2.2 (67% loss of thrust in one motor, LQR - failed, MRAC - maintained tracking)

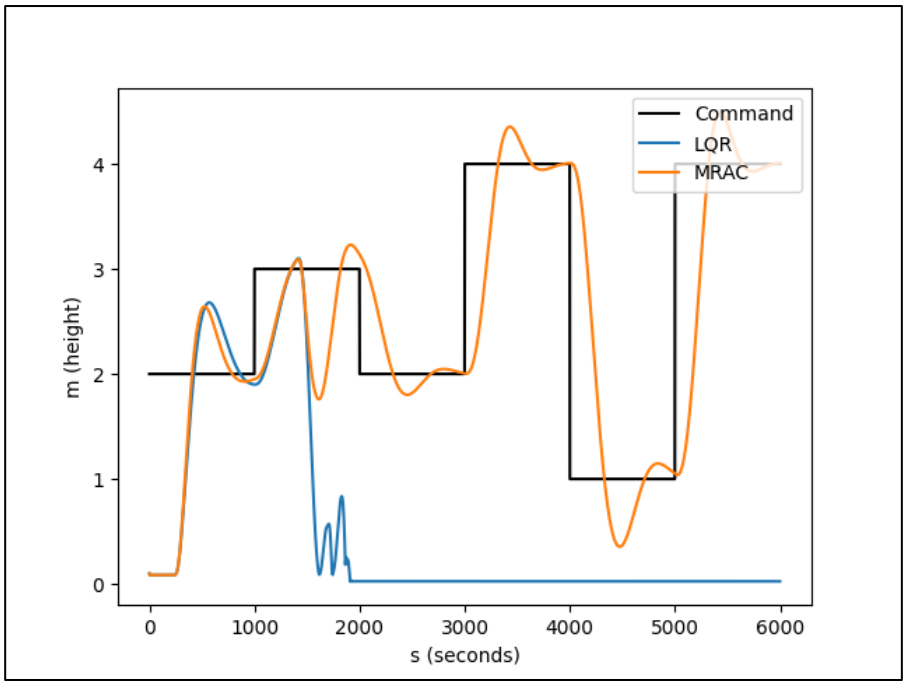

Figure 4. The LQR and MRAC result plot with 67% loss of motor thrust.

```python
# LQR optimal controller

# Import libraries
import numpy as np
from base_controller import BaseController
from lqr_solver import dlqr, lqr
from scipy.linalg import solve_continuous_lyapunov, solve_lyapunov, solve_discrete_lyapunov
from math import cos, sin
import numpy as np
from scipy import signal

class LQRController(BaseController):
    """ The LQR controller class.

    """

    def __init__(self, robot, lossOfThurst=0):
        """ LQR controller __init__ method.

        Initialize parameters here.

        Args:
            robot (webots controller object): Controller for the drone.
            lossOfThrust (float): percent lost of thrust.

        """
        super().__init__(robot, lossOfThurst)

        # define integral error
        self.int_e1 = 0
        self.int_e2 = 0
        self.int_e3 = 0
        self.int_e4 = 0

        # define K matrix
        self.K = None

    def initializeGainMatrix(self):
```

```
39          """ Calculate the gain matrix.
40
41          """
42
43          # ---------------|LQR Controller
    |-------------------------
44          # Use the results of linearization to
    create a state-space model
45
46          # Given parameters
47          n_p = 12 # number of states
48          m = 4 # number of integral error terms
49
50          # robot parameter
51          self.m = 0.4
52          self.d1x = 0.1122
53          self.d1y = 0.1515
54          self.d2x = 0.11709
55          self.d2y = 0.128
56          self.Ix = 0.000913855
57          self.Iy = 0.00236242
58          self.Iz = 0.00279965
59
60          # constants
61          self.g = 9.81
62          self.ct = 0.00026
63          self.ctau = 5.2e-06
64          self.U1_max = 10
65          self.pi = 3.1415926535
66
67          # ----------------- Your Code Here
     ----------------- #
68          # Compute the discretized A_d, B_d, C_d,
    D_d, for the computation of LQR gain
69
70          # Matrix A logic
71          # Initialize A matrix with zeros ( 16 x 16
     )
72          A = np.zeros((n_p+m, n_p+m))
73          A[0, 6] = 1; A[1, 7] = 1; A[2, 8] = 1; A[3
    , 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
 79          # Matrix B logic
 80          # Initialize B matrix with zeros ( 16 x 4
     )
 81          B = np.zeros((n_p+m, m))
 82          B[8, 0] = 1/self.m; B[9, 1] = 1/self.Ix; B
    [10, 2] = 1/self.Iy; B[11, 3] = 1/self.Iz
 83
 84          # Matrix C logic
 85          # Initialize C matrix with zeros ( 4 x 16
     )
 86          C = np.zeros ((m, n_p+m))
 87          C[0, 0] = 1; C[1, 1] = 1; C[2, 2] = 1; C[3
    , 3] = 1
 88
 89          # Matrix D logic
 90          # Zero matrix ( 4 x 4 )
 91          D = np.zeros((m, m))
 92
 93          # Discretize the system
 94          sys_discrete = signal.cont2discrete((A, B
    , C, D), self.delT, method='foh')
 95
 96          # Extract A_d, B_d, C_d, D_d
 97          A_d = sys_discrete[0]
 98          B_d = sys_discrete[1]
 99          C_d = sys_discrete[2]
100          D_d = sys_discrete[3]
101
102          # ----------------- Your Code Ends Here
     ----------------- #
103
104          # -----------------    Example code
         ----------------- #
```

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

```python
# LQR optimal controller

# Import libraries
import numpy as np
from base_controller import BaseController
from lqr_solver import dlqr, lqr
from scipy.linalg import solve_continuous_lyapunov
, solve_lyapunov, solve_discrete_lyapunov
from math import cos, sin
import numpy as np
from scipy import signal

class LQRController(BaseController):
    """ The LQR controller class.

    """

    def __init__(self, robot, lossOfThurst=0):
        """ LQR controller __init__ method.

        Initialize parameters here.

        Args:
            robot (webots controller object):
Controller for the drone.
            lossOfThrust (float): percent lost of
thrust.

        """
        super().__init__(robot, lossOfThurst)

        # define integral error
        self.int_e1 = 0
        self.int_e2 = 0
        self.int_e3 = 0
        self.int_e4 = 0

        # define K matrix
        self.K = None

    def initializeGainMatrix(self):
```

```
39          """ Calculate the gain matrix.
40
41          """
42
43          # ---------------|LQR Controller
    |-------------------------
44          # Use the results of linearization to
    create a state-space model
45
46          # Given parameters
47          n_p = 12 # number of states
48          m = 4 # number of integral error terms
49
50          # robot parameter
51          self.m = 0.4
52          self.d1x = 0.1122
53          self.d1y = 0.1515
54          self.d2x = 0.11709
55          self.d2y = 0.128
56          self.Ix = 0.000913855
57          self.Iy = 0.00236242
58          self.Iz = 0.00279965
59
60          # constants
61          self.g = 9.81
62          self.ct = 0.00026
63          self.ctau = 5.2e-06
64          self.U1_max = 10
65          self.pi = 3.1415926535
66
67          # ----------------- Your Code Here
     ----------------- #
68          # Compute the discretized A_d, B_d, C_d,
    D_d, for the computation of LQR gain
69
70          # Matrix A logic
71          # Initialize A matrix with zeros ( 16 x 16
     )
72          A = np.zeros((n_p+m, n_p+m))
73          A[0, 6] = 1; A[1, 7] = 1; A[2, 8] = 1; A[3
    , 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.
    1 * max_pos, max_pos,
111         #                        max_ang, max_ang,
    max_ang,
112         #                        0.5 * max_vel, 0.5
     * max_vel, max_vel,
113         #                        max_rate, max_rate,
    max_rate,
114         #                        0.1 * max_eyI, 0.1
     * max_eyI, 1 * max_eyI, 0.1 * max_eyI])
115 
116         # max_inputs = np.array([0.2 * self.U1_max
    , self.U1_max, self.U1_max, self.U1_max])
117 
118         # Q = np.diag(1/max_states**2)
119         # R = np.diag(1/max_inputs**2)
120         # -----------------  Example code Ends
     ----------------- #
121         # ----------------- Your Code Here
     ----------------- #
122         # Come up with reasonable values for Q and
     R (state and control weights)
123         # The example code above is a good
    starting point, feel free to use them or write you
     own.
124         # Tune them to get the better performance
125 
126         # referencing the example code above
127         max_pos = 15.0
128         max_ang = 0.2 * self.pi
129         max_vel = 6.0
130         max_rate = 0.015 * self.pi
131         # max_eyI = 0.75
132         max_eyI = 3.0
133 
134         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
```

```python
# MRAC Adaptive controller

# Import libraries
import numpy as np
from base_controller import BaseController
from lqr_solver import dlqr, lqr
from scipy.linalg import solve_continuous_lyapunov, solve_lyapunov, solve_discrete_lyapunov
from math import cos, sin
import numpy as np
from scipy import signal

class AdaptiveController(BaseController):
    """ The LQR controller class.

    """

    def __init__(self, robot, lossOfThurst):
        """ MRAC adaptive controller __init__ method.

        Initialize parameters here.

        Args:
            robot (webots controller object): Controller for the drone.
            lossOfThrust (float): percent lost of thrust.

        """

        super().__init__(robot, lossOfThurst)

        # define integral error
        self.int_e1 = 0
        self.int_e2 = 0
        self.int_e3 = 0
        self.int_e4 = 0

        # flag for initializing adaptive controller
        self.have_initialized_adaptive = False
```

```
38
39          # reference model
40          self.x_m = None
41
42          # baseline LQR controller gain
43          self.Kbl = None
44
45          # Saved matrix for adaptive law computation
46          self.A_d = None
47          self.B_d = None
48          self.Bc_d = None
49
50          self.B = None
51          self.Gamma = None
52          self.P = None
53
54          # adaptive gain
55          self.K_ad = None
56
57      def initializeGainMatrix(self):
58          """ Calculate the LQR gain matrix and
    matrices for adaptive controller.
59
60          """
61
62          # ---------------|LQR Controller
    |-------------------------
63          # Use the results of linearization to
    create a state-space model
64
65          # Given parameters
66          n_p = 12  # number of states
67          m = 4  # number of integral error terms
68
69          # robot parameter
70          self.m = 0.4
71          self.d1x = 0.1122
72          self.d1y = 0.1515
73          self.d2x = 0.11709
74          self.d2y = 0.128
75          self.Ix = 0.000913855
```

```
 76          self.Iy = 0.00236242
 77          self.Iz = 0.00279965
 78
 79          # constants
 80          self.g = 9.81
 81          self.ct = 0.00026
 82          self.ctau = 5.2e-06
 83          self.U1_max = 10
 84          self.pi = 3.1415926535
 85
 86          # ----------------- Your Code Here
     ----------------- #
 87          # Compute the continuous A, B, Bc, C, D
    and
 88          # discretized A_d, B_d, Bc_d, C_d, D_d,
    for the computation of LQR gain
 89
 90          # Matrix A logic
 91          # Initialize A matrix with zeros ( 16 x 16
     )
 92          A = np.zeros((n_p + m, n_p + m))
 93          A[0, 6] = 1; A[1, 7] = 1; A[2, 8] = 1; A[3
    , 9] = 1; A[4, 10] = 1; A[5, 11] = 1
 94          A[6, 4] = self.g; A[7, 3] = -self.g
 95          A[12, 0] = 1; A[13, 1] = 1; A[14, 2] = 1;
    A[15, 5] = 1
 96          # 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
 97
 98          # Matrix B logic
 99          # Initialize B matrix with zeros ( 16 x 4
     )
100          B = np.zeros((n_p + m, m))
101          B[8, 0] = 1 / self.m; B[9, 1] = 1 / self.
    Ix; B[10, 2] = 1 / self.Iy; B[11, 3] = 1 / self.Iz
102
103          # Matrix Bc logic
104          # Initialize Bc matrix with zeros ( 16  x
    4 )
105          Bc = np.zeros((n_p + m, m))
```

```
106         Bc[12, 0] = -1; Bc[13, 1] = -1; Bc[14, 2
    ] = -1; Bc[15, 3] = -1
107
108         # Combine B and Bc into one matrix
109         combined_B = np.hstack((B, Bc))
110
111         # Matrix C logic
112         # Initialize C matrix with zeros ( 4 x 16
     )
113         C = np.zeros((m, n_p + m))
114         C[0, 0] = 1; C[1, 1] = 1; C[2, 2] = 1; C[3
    , 3] = 1
115
116         # Matrix D logic
117         # Zero matrix ( 4 x 4 )
118         D = np.zeros((m, m))
119
120         # Discretize the system
121         sys_discrete = signal.cont2discrete((A,
    combined_B, C, D), self.delT, method='zoh')
122
123         # Extract A_d, B_d, Bc_d, C_d, D_d
124         A_d = sys_discrete[0]
125         B_d = sys_discrete[1][:, :m]  # only take
    the first 4 columns
126         Bc_d = sys_discrete[1][:, m:]  # only take
     the last 4 columns
127         C_d = sys_discrete[2]
128         D_d = sys_discrete[3]
129
130
131         # ----------------- Your Code Ends Here
     ----------------- #
132
133         # Record the matrix for later use
134         self.B = B  # continuous version of B
135         self.A_d = A_d  # discrete version of A
136         self.B_d = B_d  # discrete version of B
137         self.Bc_d = Bc_d  # discrete version of Bc
138
139         # -----------------    Example code
```

```
139         ----------------- #
140             # max_pos = 15.0
141             # max_ang = 0.2 * self.pi
142             # max_vel = 6.0
143             # max_rate = 0.015 * self.pi
144             # max_eyI = 3.
145 
146             # max_states = np.array([0.1 * max_pos, 0.
    1 * max_pos, max_pos,
147             #                        max_ang, max_ang,
    max_ang,
148             #                        0.5 * max_vel, 0.5
     * max_vel, max_vel,
149             #                        max_rate, max_rate,
    max_rate,
150             #                        0.1 * max_eyI, 0.1
     * max_eyI, 1 * max_eyI, 0.1 * max_eyI])
151 
152             # max_inputs = np.array([0.2 * self.U1_max
    , self.U1_max, self.U1_max, self.U1_max])
153 
154             # Q = np.diag(1/max_states**2)
155             # R = np.diag(1/max_inputs**2)
156             # -----------------  Example code Ends
     ----------------- #
157             # ----------------- Your Code Here
     ----------------- #
158             # Come up with reasonable values for Q and
     R (state and control weights)
159             # The example code above is a good
    starting point, feel free to use them or write you
     own.
160             # Tune them to get the better performance
161 
162             # referencing the example code above
163             max_pos = 15.0
164             max_ang = 0.2 * self.pi
165             max_vel = 6.0
166             max_rate = 0.015 * self.pi
167             max_eyI = 3.0
168 
```

```
169         max_states = np.array([0.1 * max_pos, 0.1 * max_pos, max_pos,
170                                max_ang, max_ang, max_ang,
171                                0.5 * max_vel, 0.5 * max_vel, max_vel,
172                                max_rate, max_rate, max_rate,
173                                0.1 * max_eyI, 0.1 * max_eyI, 1 * max_eyI, 0.1 * max_eyI])
174 
175         max_inputs = np.array([0.2 * self.U1_max, self.U1_max, self.U1_max, self.U1_max])
176 
177         Q = np.diag(1 / max_states ** 2)
178         R = np.diag(1 / max_inputs ** 2)
179 
180         # ----------------- Your Code Ends Here ----------------- #
181 
182         # solve for LQR gains
183         [K, _, _] = dlqr(A_d, B_d, Q, R)
184         self.Kbl = -K
185 
186         [K_CT, _, _] = lqr(A, B, Q, R)
187         Kbl_CT = -K_CT
188 
189         # initialize adaptive controller gain to baseline LQR controller gain
190         self.K_ad = self.Kbl.T
191 
192         # -----------------    Example code      ----------------- #
193         # self.Gamma = 3e-3 * np.eye(16)
194 
195         # Q_lyap = np.copy(Q)
196         # Q_lyap[0:3,0:3] *= 30
197         # Q_lyap[6:9,6:9] *= 150
198         # Q_lyap[14,14] *= 2e-3
199         # -----------------  Example code Ends ----------------- #
```

```
200         # ----------------- Your Code Here
    ----------------- #
201         # Come up with reasonable value for Gamma
    matrix and Q_lyap
202         # The example code above is a good
    starting point, feel free to use them or write you
    own.
203         # Tune them to get the better performance

204
205         # referencing the example code above
206         self.Gamma = 3e-3 * np.eye(16)
207
208         Q_lyap = np.copy(Q)
209         Q_lyap[0:3,0:3] *= 30
210         Q_lyap[6:9,6:9] *= 150
211         Q_lyap[14,14] *= 2e-3
212         # ----------------- Your Code Ends Here
    ----------------- #
213
214         A_m = A + self.B @ Kbl_CT
215         self.P = solve_continuous_lyapunov(A_m.T
    , -Q_lyap)
216
217     def update(self, r):
218         """ Get current states and calculate
    desired control input.
219
220         Args:
221             r (np.array): reference trajectory.
222
223         Returns:
224             np.array: states. information of the
    16 states.
225             np.array: U. desired control input.
226
227         """
228
229         U = np.array([0.0, 0.0, 0.0, 0.0]).reshape
    (-1,1)
230
```

```
231         # Fetch the states from the BaseController
    method
232         x_t = super().getStates()
233 
234         # update integral term
235         self.int_e1 += float((x_t[0]-r[0])*(self.
    delT))
236         self.int_e2 += float((x_t[1]-r[1])*(self.
    delT))
237         self.int_e3 += float((x_t[2]-r[2])*(self.
    delT))
238         self.int_e4 += float((x_t[5]-r[3])*(self.
    delT))
239 
240         # Assemble error-based states into array
241         error_state = np.array([self.int_e1, self.
    int_e2, self.int_e3, self.int_e4]).reshape((-1,1))
242         states = np.concatenate((x_t, error_state
    ))
243 
244         # initialize adaptive controller
245         if self.have_initialized_adaptive == False
    :
246             print("Initialize adaptive controller"
    )
247             self.x_m = states
248             self.have_initialized_adaptive = True
249         else:
250             # ----------------- Your Code Here
     ----------------- #
251             # adaptive controller update law
252             # Update self.K_ad by first order
    approximation:
253             # self.K_ad = self.K_ad +
    rate_of_change * self.delT
254 
255             # error term
256             e = states - self.x_m
257 
258             # adaptive rate of the control gain
259             rate_of_change = -self.Gamma @ states
```

```
259  @ e.T @ self.P @ self.B
260              self.K_ad = self.K_ad + rate_of_change
     * self.delT
261
262              # ----------------- Your Code Ends
    Here ----------------- #
263
264              # compute x_m at k+1
265              self.x_m = self.A_d @ self.x_m + self.
    B_d @ self.Kbl @ self.x_m + self.Bc_d @ r
266              # Compute control input
267              U = self.K_ad.T @ states
268
269          # calculate control input
270          U[0] += self.g * self.m
271
272          # Return all states and calculated control
     inputs U
273          return states, U
```

```python
# This file should be set as the controller for the DJI Maverick node.
# Please do not alter this file - it may cause the simulation to fail.

# Import Webots libraries
from controller import Robot

import numpy as np
import pickle

# Import evalution functions
from eval import showPlots

# Import functions from other scripts in controller folder
from lqr_controller import LQRController
from adaptive_controller import AdaptiveController

# Instantiate dron driver supervisor
driver = Robot()

# Get the time step of the current world
timestep = int(driver.getBasicTimeStep())

# Set your percent loss of thrust
lossOfThust = 0.5  # Ex2.1 50% loss of thrust
# lossOfThust = 0.67  # Ex2.2 67% loss of thrust

# Instantiate controller and start sensors
# customController = LQRController(driver, lossOfThust)  # LQR controller
customController = AdaptiveController(driver, lossOfThust)  # Adaptive controller
customController.initializeMotors()
customController.startSensors(timestep)

# Initialize state storage vectors
stateHistory = []
referenceHistory = []

```

```
37 # flag for motor failure
38 motor_failure = False
39
40 # calculate gain matrix for baseline LQR controller
    & adaptive controller
41 customController.initializeGainMatrix()
42
43 # start simulation for LQR controller
44 while driver.step(timestep) != -1:
45
46     current_time = driver.getTime()
47     print("Time:", current_time)
48
49     # motor failure after 14 s
50     if current_time > 14:
51         motor_failure = True
52
53     # reference trajectory
54     if current_time < 10:
55         r = np.array([0, 0, 2, 0]).reshape(-1,1)
56     elif current_time >= 10 and current_time < 20:
57         r = np.array([0, 0, 3, 0]).reshape(-1,1)
58     elif current_time >= 20 and current_time < 30:
59         r = np.array([0, 0, 2, 0]).reshape(-1,1)
60     elif current_time >= 30 and current_time < 40:
61         r = np.array([0, 0, 4, 0]).reshape(-1,1)
62     elif current_time >= 40 and current_time < 50:
63         r = np.array([0, 0, 1, 0]).reshape(-1,1)
64     elif current_time >= 50 and current_time < 60:
65         r = np.array([0, 0, 4, 0]).reshape(-1,1)
66     else:
67         # end simulation
68         break
69
70     # Call control update method
71     states, U = customController.update(r)
72
73     # Check failure
74     if (states[2] < 0 ):
75         print("="*15 + "Drone Crashed" + "="*15)
76         print("="*15 + "Your Controller Failed" +
```

```
 76 "="*15)
 77         break;
 78 
 79     # Convert control input to motoespeed
 80     rotorspeed = customController.
    convertUtoMotorSpeed(U)
 81 
 82     # set motor speed
 83     customController.setMotorsSpeed(rotorspeed,
    motor_failure)
 84 
 85     # collect state history for evaluation
 86     stateHistory.append(list(states.flatten()))
 87     referenceHistory.append(list(r.flatten()))
 88 
 89 # save data for evaluation
 90 # reference trajectory
 91 np.save("r_hist_ex2",referenceHistory)
 92 
 93 # COMMENT one of two following lines to correctly
    save te state data
 94 if type(customController).__name__ == '
    AdaptiveController':
 95     np.save("x_ad_hist_ex2",stateHistory) # states
     using adatpive controller
 96 elif type(customController).__name__ == '
    LQRController':
 97     np.save("x_lqr_hist_ex2",stateHistory) #
    states using lqr controller
 98 
 99 # simulation finished, draw plots
100 showPlots()
```