

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  '''
5  Serial packet handler module
6  '''
7
8  __author__ = "Heethesh Vhavle"
9  __version__ = "1.0.0"
10 __email__ = "heethesh@cmu.edu"
11
12 import time
13 import serial
14 import struct
15 import traceback
16
17
18 class Packet:
19     def __init__(self):
20         self.is_open = False
21
22         # TX data
23         self.tx_slot_encoder = False
24         self.tx_encoder = {'encoder_count': 0, 'encoder_velocity': 0}
25         self.tx_temperature = 0
26         self.tx_ultrasonic_distance = 0
27         self.tx_flex_sensor = 0
28         self.tx_servo_angle = 0
29
30         # RX data
31         self.rx_global_switch = False
32         self.rx_state = 0
33         self.rx_servo_angle = 0
34         self.rx_motor_angle = 0
35         self.rx_motor_velocity = 0
36         self.rx_stepper_value = 0
37         self.rx_stepper_dir = 0
38         self.rx_stepper_flag = False
39
40     def start(self, com_port, baud=115200, timeout=0):
41         # Configure serial port
42         self.ser = serial.Serial()
43         self.ser.port = com_port
44         self.ser.baudrate = baud
45         self.ser.timeout = timeout
46
47         # Time to wait until the board becomes operational
48         wakeup = 2
49         try:
50             self.ser.open()
51             print("\n>>> Opening COM Port: " + self.ser.port)
52             for i in range(1, wakeup):
53                 time.sleep(1)
54         except Exception as error:
55             traceback.print_tb(error.__traceback__)
56             self.ser.close()
57             self.is_open = False
58
59         # Clear buffer
60         self.ser.flushInput()
61         self.ser.flushOutput()
62         self.is_open = True
63
64     def close(self):

```

```

65         try:
66             self.ser.close()
67         except AttributeError as e:
68             print(e)
69         self.is_open = False
70
71     def generate_frame(self, data, data_format, mode):
72         checksum = 0
73         frame = ''
74         direc = {'tx': '<', 'rx': '>'}
75
76         # Pack data into bytes
77         header = struct.pack('cc', '$'.encode('utf-8'), direc[mode].encode('utf-8'))
78         payload = struct.pack(data_format, *data)
79         data_length = struct.pack('B', len(payload))
80
81         # Calculate checksum
82         for byte in payload:
83             checksum ^= byte
84         checksum = struct.pack('B', checksum)
85
86         # Complete frame
87         frame = header + data_length + payload + checksum
88         return frame
89
90     def send_packet(self, data, data_format, mode='tx'):
91         # Make frame
92         tx_frame = self.generate_frame(data, data_format, mode)
93
94         # Send data
95         try:
96             self.ser.write(tx_frame)
97
98         except Exception as error:
99             print(error)
100             traceback.print_tb(error.__traceback__)
101
102         # Clear buffer
103         self.ser.flushInput()
104         self.ser.flushOutput()
105
106     def recieve_packet(self, data_format, data_length):
107         checksum = 0
108         calcsun = 0
109         payload = ''
110         rx_data = []
111
112         # Recieve data
113         try:
114             if self.ser.inWaiting() >= (data_length + 4):
115                 # Verify header
116                 if self.ser.read(1).decode('utf-8') != '$':
117                     return None
118                 if self.ser.read(1).decode('utf-8') != '>':
119                     return None
120
121                 # Verify data length
122                 data = int(ord(self.ser.read(1).decode('utf-8')))
123                 if data != data_length:
124                     return None
125
126                 payload = self.ser.read(data_length)
127                 checksum = self.ser.read(1)
128
129                 # Clear buffer
130                 self.ser.flushInput()

```

```

131         self.ser.flushOutput()
132
133         # Verify checksum
134         for byte in payload:
135             calcsun ^= byte
136         if calcsun != ord(checksum):
137             return None
138
139         # Unpack data
140         rx_data = list(struct.unpack(data_format, payload))
141         return rx_data
142
143     except Exception as error:
144         traceback.print_tb(error.__traceback__)
145
146     # Clear buffer
147     self.ser.flushInput()
148     self.ser.flushOutput()
149
150     return None
151
152 def send(self):
153     data = [
154         self.rx_global_switch,
155         self.rx_state,
156         self.rx_servo_angle,
157         self.rx_motor_angle,
158         self.rx_motor_velocity,
159         self.rx_stepper_value,
160         self.rx_stepper_dir,
161         self.rx_stepper_flag,
162     ]
163
164     self.send_packet(data, '<BBBhhHBB')
165
166 def recieve(self, delay=0.2, max_retries=5):
167     data = None
168     retries = 0
169     while not data:
170         if retries > max_retries:
171             return False
172         time.sleep(delay)
173         retries += 1
174         data = self.recieve_packet('<BifBHHB', 15)
175
176     self.parse_data(data)
177     # self.display()
178     return True
179
180 def parse_data(self, data):
181     # Boolean data
182     self.tx_slot_encoder = data[0]
183
184     # Encoder data
185     self.tx_encoder['encoder_count'] = data[1]
186     self.tx_encoder['encoder_velocity'] = data[2]
187
188     # Sensors data
189     self.tx_temperature = data[3]
190     self.tx_ultrasonic_distance = data[4]
191     self.tx_flex_sensor = data[5]
192
193     # Servo angle
194     self.tx_servo_angle = data[6]
195
196 def display(self):

```

```

197         print('tx_slot_encoder:', self.tx_slot_encoder)
198         print('tx_encoder:', self.tx_encoder)
199         print('tx_temperature:', self.tx_temperature)
200         print('tx_ultrasonic_distance:', self.tx_ultrasonic_distance)
201         print('tx_flex_sensor', self.tx_flex_sensor)
202         print('tx_servo_angle:', self.tx_servo_angle)
203         print()
204
205
206     if __name__ == '__main__':
207         packet = Packet()
208         packet.start('/dev/ttyACM0')
209
210         # Recieve test
211         while True:
212             packet.recieve()
213
214         # Send test
215         # packet.rx_global_switch = False
216         # packet.rx_state = 10
217         # packet.rx_servo_angle = 90
218         # while True:
219         #     packet.send()
220         #     time.sleep(1)
221
222         packet.close()

```