```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

'''
Python GUI for Sensors and Motors Lab
'''

__author__ = "Heethesh Vhavle"
__version__ = "1.0.0"
__email__ = "heethesh@cmu.edu"

# Built-in modules
import os
import glob
import time

# External modules
import numpy as np
from tkinter import *
from tkinter import ttk
from tkinter.ttk import Separator, Progressbar, Notebook
from PIL import ImageTk, Image
import matplotlib

matplotlib.use('TkAgg')
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# Local modules
from utils import *
from packet import Packet

# Globals
PATH = os.path.dirname(os.path.abspath(__file__)) + '/'
HOME_PATH = os.path.expanduser('~') + '/'

app = None
GUI_CLOSED = False

# Arduino packet object
arduino = Packet()
last_time = time.time()

STATES = [
    'Reserved',
    'DC Motor Position (GUI)',
    'DC Motor Velocity (GUI)',
    'DC Motor Position (Sensor)',
    'DC Motor Velocity (Sensor)',
    'Stepper Motor (GUI)',
    'Stepper Motor (Sensor)',
    'Servo Motor (GUI)',
    'Servo Motor (Sensor)',
]

'''
Function wrapper to send data packet to arduino
'''
def arduino_send():
    global last_time
    if (time.time() - last_time) > 0.005 and arduino.is_open:
        last_time = time.time()
        arduino.send()

```

```
65
66  '''
67  Class to handle the sensor panel using a progress bar
68  '''
69  class SensorPanel:
70      def __init__(self, name, min_val, max_val, parent, row, column, padx):
71          # Sensor properties
72          self.name = name
73          self.min_val = min_val
74          self.max_val = max_val
75
76          # Append to Tk parent panel
77          self.panel = Frame(parent, width=170, height=10, pady=3)
78          if name:
79              self.name_label = Label(self.panel, text=name)
80              self.name_label.grid(row=0, column=0, columnspan=2, pady=5)
81          self.pb = Progressbar(
82              self.panel, orient=HORIZONTAL, mode="determinate", length=140
83          )
84          self.pb.grid(row=1, column=0, sticky=(W, E))
85          self.value_label = Entry(self.panel, width=7, justify=RIGHT)
86          self.value_label.grid(row=1, column=1, padx=5, sticky=(W, E))
87          self.panel.grid(row=row, column=column, padx=padx)
88          self.set_sensor_value(0)
89
90      def set_sensor_value(self, value):
91          self.pb['value'] = map_value(value, self.min_val, self.max_val)
92          self.value_label.delete(0, END)
93          self.value_label.insert(0, '%.2f' % value)
94
95
96  '''
97  Class to handle the slider panel using a slider
98  '''
99  class SliderPanel:
100     def __init__(self, name, min_val, max_val, parent, row, column, padx, func):
101         # Sensor properties
102         self.name = name
103         self.min_val = min_val
104         self.max_val = max_val
105
106         # Append to Tk parent panel
107         self.panel = Frame(parent, width=170, pady=3)
108         self.name_label = Label(self.panel, text=name)
109         self.name_label.grid(row=0, column=0, padx=6, sticky=(S))
110         self.slider = Scale(
111             self.panel,
112             from_=min_val,
113             to=max_val,
114             orient=HORIZONTAL,
115             length=140,
116             command=func,
117         )
118         self.slider.grid(row=0, column=1, sticky=(W, E))
119         self.panel.grid(row=row, column=column, padx=padx)
120
121
122 '''
123 Class to handle the titles used for the panels
124 '''
125 class SectionTitle:
126     def __init__(self, title, parent, row, column, width, top=10, bottom=10):
127         Separator(parent, orient=HORIZONTAL).grid(
128             row=row, column=column, pady=(top, bottom), sticky=(W, E)
129         )
130         self.panel = Frame(parent, width=width, height=10)
```

```
131         self.panel.grid(row=row, column=column)
132         self.label = Label(self.panel, text=" %s " % title)
133         self.label.grid(row=row, column=column, pady=(top, bottom))
134 
135 
136 '''
137 Class to handle the eight different state panels
138 '''
139 class StatePanel:
140     def __init__(self, state, parent, row, column, width, padx):
141         self.state_index = STATES.index(state)
142         self.panel = Frame(parent, width=width, pady=3)
143         self.raisedPanel = Frame(self.panel, width=width, pady=3,
144             bd=1, relief=RAISED)
145         SectionTitle(state, self.panel, row=0, column=0, width=170)
146 
147         # DC motor position GUI
148         if self.state_index == 1:
149             self.slider1 = SliderPanel(
150                 'Angle',
151                 -360,
152                 360,
153                 self.raisedPanel,
154                 row=0,
155                 column=0,
156                 padx=9,
157                 func=self.control_dcmotor_pos,
158             )
159             self.sensor1 = SensorPanel(
160                 'Encoder Position (Ticks)',
161                 -10000,
162                 10000,
163                 self.raisedPanel,
164                 row=1,
165                 column=0,
166                 padx=9,
167             )
168 
169         # DC motor velocity GUI
170         elif self.state_index == 2:
171             self.slider1 = SliderPanel(
172                 'Velocity',
173                 0,
174                 110,
175                 self.raisedPanel,
176                 row=0,
177                 column=0,
178                 padx=9,
179                 func=self.control_dcmotor_vel,
180             )
181             self.sensor1 = SensorPanel(
182                 'Encoder Velocity (RPM)',
183                 -110,
184                 110,
185                 self.raisedPanel,
186                 row=1,
187                 column=0,
188                 padx=9,
189             )
190 
191         # DC motor position sensor
192         elif self.state_index == 3:
193             self.sensor1 = SensorPanel(
194                 'Encoder Position (Ticks)',
195                 -1000000,
196                 1000000,
```

```
197                 self.raisedPanel,
198                 row=0,
199                 column=0,
200                 padx=9,
201             )
202             self.sensor2 = SensorPanel(
203                 'Temperature Sensor (C)',
204                 0,
205                 200,
206                 self.raisedPanel,
207                 row=1,
208                 column=0,
209                 padx=9,
210             )
211 
212         # DC motor velocity sensor
213         elif self.state_index == 4:
214             self.sensor1 = SensorPanel(
215                 'Encoder Velocity (RPM)',
216                 -110,
217                 110,
218                 self.raisedPanel,
219                 row=0,
220                 column=0,
221                 padx=9,
222             )
223             self.sensor2 = SensorPanel(
224                 'Ultrasonic Sensor (inches)',
225                 6,
226                 24,
227                 self.raisedPanel,
228                 row=1,
229                 column=0,
230                 padx=9,
231             )
232 
233         # Stepper motor GUI control
234         elif self.state_index == 5:
235             self.slider1 = SliderPanel(
236                 'Angle',
237                 0,
238                 360,
239                 self.raisedPanel,
240                 row=0,
241                 column=0,
242                 padx=9,
243                 func=self.control_stepper_pos,
244             )
245             self.slider2 = SliderPanel(
246                 'Direction',
247                 0,
248                 1,
249                 self.raisedPanel,
250                 row=1,
251                 column=0,
252                 padx=9,
253                 func=self.control_stepper_dir,
254             )
255             self.sensor1 = SensorPanel(
256                 'Slot Encoder', 0, 1, self.raisedPanel, row=2, column=0, padx=9
257             )
258             self.button = Button(
259                 self.raisedPanel,
260                 text="Set Angle",
261                 command=self.set_stepper_flag,
262                 pady=4,
```

```
263                 width=24,
264             )
265             self.button.grid(row=3, column=0, pady=3)
266 
267         # Stepper motor sensor control
268         elif self.state_index == 6:
269             self.sensor1 = SensorPanel(
270                 'Slot Encoder', 0, 1, self.raisedPanel, row=1, column=0, padx=9
271             )
272 
273         # Servo motor GUI control
274         elif self.state_index == 7:
275             self.slider1 = SliderPanel(
276                 'Angle',
277                 0,
278                 90,
279                 self.raisedPanel,
280                 row=0,
281                 column=0,
282                 padx=9,
283                 func=self.control_servo,
284             )
285             self.sensor1 = SensorPanel(
286                 'Flex Sensor', 0, 1024, self.raisedPanel, row=1, column=0, padx=9
287             )
288 
289         # Servo motor sensor control
290         elif self.state_index == 8:
291             self.sensor1 = SensorPanel(
292                 'Flex Sensor', 0, 1024, self.raisedPanel, row=1, column=0, padx=9
293             )
294 
295         self.raisedPanel.grid(row=1, column=0, padx=padx)
296         self.panel.grid(row=row, column=column, padx=padx)
297         self.configure_state(self.raisedPanel, state=DISABLED)
298 
299     '''Callback functions for all states'''
300 
301     def control_servo(self, value):
302         arduino.rx_servo_angle = int(value)
303         arduino_send()
304 
305     def control_stepper_pos(self, value):
306         arduino.rx_stepper_value = int(value)
307 
308     def control_stepper_dir(self, value):
309         arduino.rx_stepper_dir = int(value)
310 
311     def set_stepper_flag(self):
312         arduino.rx_stepper_flag = 1
313         arduino_send()
314 
315     def control_dcmotor_pos(self, value):
316         arduino.rx_motor_angle = int(value)
317         arduino_send()
318 
319     def control_dcmotor_vel(self, value):
320         arduino.rx_motor_velocity = int(value)
321         arduino_send()
322 
323     def configure_state(self, frame, state):
324         for child in frame.winfo_children():
325             if type(child) == Frame:
326                 self.configure_state(child, state)
327             elif type(child) == Progressbar:
328                 continue
```

```
            else:
                child.configure(state=state)


'''
Main class to handle the GUI
'''
class GUI(object):
    def __init__(self, master):
        # Main Window
        self.width = 820
        self.height = 645
        master.title("Sensors and Motors Lab  |  Delta Autonomy")
        master.geometry('%dx%d' % (self.width, self.height))
        master.resizable(width=False, height=False)

        self.ICON_PATH = PATH + 'images/da_logo_resize.gif'
        self.imgicon = PhotoImage(file=self.ICON_PATH)
        master.tk.call('wm', 'iconphoto', master._w, self.imgicon)

        ########################################################################

        # Master Panel
        self.mpanel = Frame(
            master, width=self.width, height=self.height, padx=5, pady=4
        )
        self.mpanel.pack()

        ########################################################################

        # Left Panel
        self.lpanel = Frame(self.mpanel, width=170, height=self.height, pady=3)
        self.lpanel.grid(row=0, column=0, rowspan=4, padx=10)

        ########################################################################

        # Info Panel
        self.raisedFrame = Frame(self.lpanel, bd=3, relief=GROOVE)
        self.raisedFrame.grid(row=0, column=0)

        self.logo = ImageTk.PhotoImage(Image.open(PATH + 'images/da_logo_resize.gif'))
        self.logolabel = Label(self.raisedFrame, image=self.logo, width=205)
        self.logolabel.grid(row=0, column=0)

        self.infolabel3 = Label(self.raisedFrame, text=" ", font='"Consolas" 2')
        self.infolabel3.grid(row=1, column=0)

        self.infolabel = Label(
            self.raisedFrame, text="Delta Autonomy", font='"Consolas" 12 bold'
        )
        self.infolabel.grid(row=2, column=0)

        self.infolabel2 = Label(
            self.raisedFrame,
            text="Sensors and Motors Lab GUI\nVersion %s" % __version__,
            font='"Consolas" 10',
        )
        self.infolabel2.grid(row=3, column=0)

        self.infolabel5 = Label(self.raisedFrame, text=" ", font='"Consolas" 4')
        self.infolabel5.grid(row=4, column=0)

        self.infolabel6 = Label(
            self.lpanel, text="\nInstructions", font='"Consolas" 11 bold'
        )
        self.infolabel6.grid(row=3, column=0)
```

```
395
396         self.infotext = "Select the COM port and open it. Select any one of the eight
    states and click start demo to enable the corresponding state panel. You can now visualize
    the sensors data and control the actuators."
397         self.infolabel4 = Label(
398             self.lpanel, text=self.infotext, wraplength=205, font='"Consolas" 9'
399         )
400         self.infolabel4.grid(row=4, column=0)
401
402         ######################################################################
403
404         # COM Port Panel
405         SectionTitle('Select COM Port', self.lpanel, 5, 0, 170)
406
407         self.comport = StringVar(master)
408         self.comports = self.get_com_ports()
409         if not self.comports:
410             self.comports = ['No Devices Available']
411         self.ddcom = OptionMenu(
412             self.lpanel, self.comport, *self.comports, command=self.comport_select
413         )
414         self.ddcom.config(width=22)
415         self.ddcom.grid(row=6, column=0)
416
417         self.b1 = Button(
418             self.lpanel, text="Open Port", command=self.b1_clicked,
419             pady=4, width=24
420         )
421         self.b1.grid(row=7, column=0)
422         self.b1.configure(state=DISABLED)
423
424         ######################################################################
425
426         # State Select Panel
427         SectionTitle('Select State', self.lpanel, 8, 0, 170)
428
429         self.state = StringVar(master)
430         self.states = STATES[1:]
431         self.ddstate = OptionMenu(
432             self.lpanel, self.state, *self.states, command=self.state_select
433         )
434         self.ddstate.config(width=22)
435         self.ddstate.grid(row=9, column=0)
436
437         self.b2 = Button(
438             self.lpanel, text="Start Demo", command=self.b2_clicked,
439             pady=4, width=24
440         )
441         self.b2.grid(row=10, column=0)
442         self.b2.configure(state=DISABLED)
443
444         ######################################################################
445
446         # Separator
447         Separator(self.mpanel, orient=VERTICAL).grid(
448             row=0, column=1, rowspan=20, sticky=(N, S), padx=6
449         )
450
451         # Right Panel
452         self.rpanel = Frame(self.mpanel, width=170, height=self.height, pady=3)
453         self.rpanel.grid(row=0, column=2, rowspan=4)
454
455         # State Panels
456         self.state1_panel = StatePanel(
457             STATES[1], self.rpanel, row=0, column=0, width=170, padx=9
458         )
```

```
459         self.state3_panel = StatePanel(
460             STATES[3], self.rpanel, row=1, column=0, width=170, padx=9
461         )
462         self.state2_panel = StatePanel(
463             STATES[2], self.rpanel, row=2, column=0, width=170, padx=9
464         )
465         self.state4_panel = StatePanel(
466             STATES[4], self.rpanel, row=3, column=0, width=170, padx=9
467         )
468 
469         ########################################################################
470 
471         # Separator
472         Separator(self.mpanel, orient=VERTICAL).grid(
473             row=0, column=3, rowspan=20, sticky=(N, S), padx=6
474         )
475 
476         # Right Panel
477         self.r2panel = Frame(self.mpanel, width=170, height=self.height, pady=3)
478         self.r2panel.grid(row=0, column=4, rowspan=4)
479 
480         # State Panels
481         self.state5_panel = StatePanel(
482             STATES[5], self.r2panel, row=0, column=0, width=170, padx=9
483         )
484         self.state6_panel = StatePanel(
485             STATES[6], self.r2panel, row=1, column=0, width=170, padx=9
486         )
487         self.state7_panel = StatePanel(
488             STATES[7], self.r2panel, row=2, column=0, width=170, padx=9
489         )
490         self.state8_panel = StatePanel(
491             STATES[8], self.r2panel, row=3, column=0, width=170, padx=9
492         )
493 
494         ########################################################################
495 
496         # Keep track all state panel objects
497         self.current_state = STATES[0]
498         self.state_panels = [
499             self.state1_panel,
500             self.state2_panel,
501             self.state3_panel,
502             self.state4_panel,
503             self.state5_panel,
504             self.state6_panel,
505             self.state7_panel,
506             self.state8_panel,
507         ]
508 
509         ########################################################################
510 
511         # Separator
512         Separator(self.mpanel, orient=VERTICAL).grid(
513             row=0, column=3, rowspan=20, sticky=(N, S), padx=6
514         )
515 
516         ########################################################################
517 
518     '''Callback functions'''
519 
520     def get_com_ports(self):
521         # Linux
522         return glob.glob('tty[AU]*')
523 
524     def comport_select(self, port):
```

```
525             print('Port changed:', port)
526             self.b1.configure(state=NORMAL)
527 
528         def state_select(self, state):
529             print('State changed:', STATES.index(state), state)
530             self.b2.configure(state=NORMAL)
531             self.current_state = state
532 
533         def b1_clicked(self):
534             print('B1 Clicked')
535             if self.b1['text'] == 'Close Port':
536                 self.b1['text'] = 'Open Port'
537                 self.ddcom.configure(state=NORMAL)
538                 # arduino.close()
539 
540             elif self.b1['text'] == 'Open Port':
541                 self.b1['text'] = 'Close Port'
542                 self.ddcom.configure(state=DISABLED)
543                 print('Opening Port:', str(self.comport.get()))
544                 # arduino.start(str(self.comport.get()))
545 
546         def b2_clicked(self):
547             print('B2 Clicked')
548             if self.b2['text'] == 'Stop Demo':
549                 self.b2['text'] = 'Start Demo'
550                 self.ddstate.configure(state=NORMAL)
551                 state_index = STATES.index(self.current_state)
552                 if state_index:
553                     self.state_panels[state_index - 1].configure_state(
554                         self.state_panels[state_index - 1].raisedPanel, state=DISABLED
555                     )
556                 arduino.rx_global_switch = 0
557                 arduino.rx_state = state_index
558                 arduino_send()
559 
560             elif self.b2['text'] == 'Start Demo':
561                 self.b2['text'] = 'Stop Demo'
562                 self.ddstate.configure(state=DISABLED)
563                 state_index = STATES.index(self.current_state)
564                 if state_index:
565                     self.state_panels[state_index - 1].configure_state(
566                         self.state_panels[state_index - 1].raisedPanel, state=NORMAL
567                     )
568                 arduino.rx_global_switch = 1
569                 arduino.rx_state = state_index
570                 arduino_send()
571 
572         '''
573         Function to update data on the GUI from serial packets
574         '''
575         def update_data(self):
576             self.state1_panel.sensor1.set_sensor_value(arduino.tx_encoder['encoder_count'])
577             self.state2_panel.sensor1.set_sensor_value(
578                 arduino.tx_encoder['encoder_velocity']
579             )
580             self.state3_panel.sensor1.set_sensor_value(arduino.tx_encoder['encoder_count'])
581             self.state3_panel.sensor2.set_sensor_value(arduino.tx_temperature)
582             self.state4_panel.sensor1.set_sensor_value(
583                 arduino.tx_encoder['encoder_velocity']
584             )
585             self.state4_panel.sensor2.set_sensor_value(arduino.tx_ultrasonic_distance)
586             self.state5_panel.sensor1.set_sensor_value(int(bool(arduino.tx_slot_encoder)))
587             self.state6_panel.sensor1.set_sensor_value(int(bool(arduino.tx_slot_encoder)))
588             self.state7_panel.sensor1.set_sensor_value(arduino.tx_flex_sensor)
589             self.state8_panel.sensor1.set_sensor_value(arduino.tx_flex_sensor)
590 
```

```
591
592 '''
593 Threaded function to keep listening to packets over serial
594 '''
595 def packet_listener():
596     global app
597     time.sleep(2)
598
599     # Keep running thread till GUI is open
600     while not GUI_CLOSED:
601         if arduino.is_open and arduino.recieve():
602             app.update_data()
603         else:
604             time.sleep(0.1)
605
606
607 if __name__ == '__main__':
608     # Start thread
609     packet_listener_t = StoppableThread(target=packet_listener)
610     packet_listener_t.start()
611
612     # Create GUI and start GUI thread
613     root = Tk()
614     root.wm_attributes('-type', 'splash')
615     app = GUI(root)
616     root.mainloop()
617
618     # Cleanup
619     GUI_CLOSED = True
620     packet_listener_t.stop()
621     if arduino.is_open: arduino.close()
```