

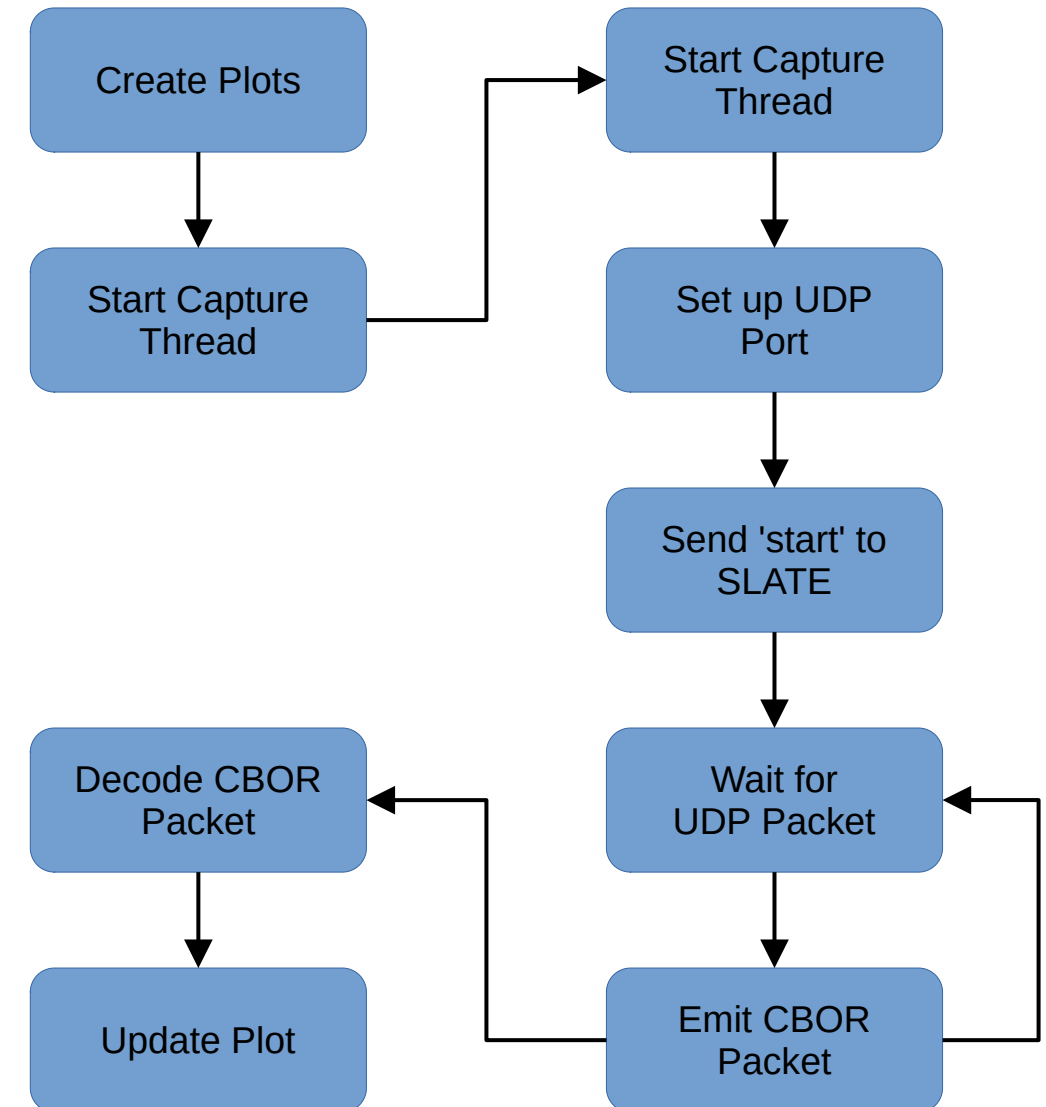
Plotting Data

Plotting

- A way to display data is with plots. Plots can be useful to provide a quick visual display of the data and also show any data trends such as battery voltage dropping over time.
- The GUI for Python that will be used is call PyQt. Version 6 will be used. pyqtgraph will also be used. It is a module that makes creating plots easier.
- Install both modules
 - **`pip3 install pyqt6`**
 - **`pip3 install pyqtgraph`**

Plotting

- This example will plot data from the air pressure sensor and humidity sensor
- The xmit2.py program will be reused on Thonny with no changes.
- The plotting program will use a Plot module you will create so it is easier to create multiple plots.
- A thread called Capture will set up the network connection to the SLATE and wait for data to be received.
- The CBOR data packet will be sent to the main program using a PYQT signal.
- The signal will trigger a function to be called that updates the plots.



Plotting Module

- A plotting module will be created to make it easier to create multiple plots.
- In IDLE, create a new file and enter the code to the right and save it to **plotline.py**.
- This class will let you create a line plot.
- It requires five arguments
 - length is the number of data points to plot at once
 - width is the horizontal size of the plot in pixels.
 - height is the vertical size in pixels.
 - Title is the name of the plot to be displayed in the plot.
 - Color is line plot color. It is a tuple with three values (red, green, blue). The range for each value is 0 to 255.

```
from PyQt6 import *
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *
import pyqtgraph as pg

# Plot widget
# length = number of data points to plot
# width is the X-size of the plot in pixels
# height is the Y-size of the plot in pixels
# title is a string for the plot title
# Color is the color of the data line in (r,g,b) tuple
class PlotLine(QWidget):
    def __init__(self, length, width, height, title, Color):
        super().__init__()
        self.Plt = pg.PlotWidget()
        self.data = [0.0] * length
        self.time = list(range(length))
        self.line = self.Plt.plot(self.time, self.data,
                                   pen=pg.mkPen(color=Color))
        self.Plt.setFixedSize(width, height)
        self.Plt.setTitle(title)
        layout = QVBoxLayout()
        layout.addWidget(self.Plt)
        self.setLayout(layout)

    def update(self, point):
        self.data = self.data[1:]
        self.data.append(point)
        self.line.setData(self.time, self.data)

    def setrange(self, high, low):
        self.Plt.setYRange(low, high)
```

Plotting Module

- First a plot widget is instantiated from the **pyqtgraph** module.
- Next, a variable is created with a list size set by the length argument. This will hold the Y-axis value.
- **self.time** is a list with values ranging from 0 to 199 for the x-axis value.
- A 2D plot is created with **self.Plt.plot()**. The X-axis and Y-axis lists are provided and the plot color is set.
- Next, the size of the plot is set.
- The title is then added.
- Lastly, the plot widget is placed in the widget layout. Since the plot is the only widget, the Vertical box layout is used.

```
from PyQt6 import *
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *
import pyqtgraph as pg

# Plot widget
# length = number of data points to plot
# width is the X-size of the plot in pixels
# height is the Y-size of the plot in pixels
# title is a string for the plot title
# Color is the color of the data line in (r,g,b) tuple
class PlotLine(QWidget):
    def __init__(self, length, width, height, title, Color):
        super().__init__()
        self.Plt = pg.PlotWidget()
        self.data = [0.0] * length
        self.time = list(range(length))
        self.line = self.Plt.plot(self.time, self.data,
                                   pen=pg.mkPen(color=Color))
        self.Plt.setFixedSize(width, height)
        self.Plt.setTitle(title)
        layout = QVBoxLayout()
        layout.addWidget(self.Plt)
        self.setLayout(layout)

    def update(self, point):
        self.data = self.data[1:]
        self.data.append(point)
        self.line.setData(self.time, self.data)

    def setrange(self, high, low):
        self.Plt.setYRange(low, high)
```

Plotting Module

- The **update()** function is called to add a data point to the plot.
- First, the list is copied to itself by taking the last length-1 values and saving it back itself. This removes the first data point in the list.
- Next, the new data point is appended to the end of the list.
- Lastly, the plot is updated with the new list values.

```
from PyQt6 import *
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *
import pyqtgraph as pg

# Plot widget
# length = number of data points to plot
# width is the X-size of the plot in pixels
# height is the Y-size of the plot in pixels
# title is a string for the plot title
# Color is the color of the data line in (r,g,b) tuple
class PlotLine(QWidget):
    def __init__(self, length, width, height, title, Color):
        super().__init__()
        self.Plt = pg.PlotWidget()
        self.data = [0.0] * length
        self.time = list(range(length))
        self.line = self.Plt.plot(self.time, self.data,
                                   pen=pg.mkPen(color=Color))
        self.Plt.setFixedSize(width, height)
        self.Plt.setTitle(title)
        layout = QVBoxLayout()
        layout.addWidget(self.Plt)
        self.setLayout(layout)

    def update(self, point):
        self.data = self.data[1:]
        self.data.append(point)
        self.line.setData(self.time, self.data)

    def setrange(self, high, low):
        self.Plt.setYRange(low, high)
```

Plotting Module

- The `setrange()` function lets you set the Y-axis range. The function requires the high range and the low range.
- The plot will adjust to just plot the range specified.

```
from PyQt6 import *
from PyQt6.QtWidgets import *
from PyQt6.QtCore import *
from PyQt6.QtGui import *
import pyqtgraph as pg

# Plot widget
# length = number of data points to plot
# width is the X-size of the plot in pixels
# height is the Y-size of the plot in pixels
# title is a string for the plot title
# Color is the color of the data line in (r,g,b) tuple
class PlotLine(QWidget):
    def __init__(self, length, width, height, title, Color):
        super().__init__()
        self.Plt = pg.PlotWidget()
        self.data = [0.0] * length
        self.time = list(range(length))
        self.line = self.Plt.plot(self.time, self.data,
                                   pen=pg.mkPen(color=Color))
        self.Plt.setFixedSize(width, height)
        self.Plt.setTitle(title)
        layout = QVBoxLayout()
        layout.addWidget(self.Plt)
        self.setLayout(layout)

    def update(self, point):
        self.data = self.data[1:]
        self.data.append(point)
        self.line.setData(self.time, self.data)

    def setrange(self, high, low):
        self.Plt.setYRange(low, high)
```

Plotting

- Open a new file on the laptop in IDLE and call it **plot1.py**.
- The first part of the plotting program is the Capture thread that will wait for data and send it to the main program for plotting.
- The Capture thread is a class of type QThread which inherits the features of a thread. A thread runs in parallel with the main program.
- First a signal is create called data. The data type needs to be defined and it is defined as bytes. The CBOR data is in binary format.

```
import sys
import socket
from PyQt6.QtWidgets import *
from PyQt6.QtCore import Qt, QTimer
import pyqtgraph as pg
from cbor2 import loads
from plotline import PlotLine

class Capture(QThread):
    data = pyqtSignal(bytes)

    def __init__(self):
        super().__init__()
        self.is_running = True

    @pyqtSlot()
    def run(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.bind(('192.168.4.16', 10000))
        sock.sendto(b'start', ('192.168.4.1', 10000))
        while self.is_running:
            d, a = sock.recvfrom(128)
            self.data.emit(d)

    def close(self):
        self.is_running = False
```

Plotting

- The `__init__()` function create the variable **`self.is_running`** and sets it to **`True`**. This is used in the **`run()`** function to know when to stop looping.
- The **`@pyqtSlot()`** is a decorator for the **`run()`** function and identifies the function as a QT slot which allows for sending and receiving signals.
- The `run()` function is automatically started after `__init__()` is completed.
- The network socket is created and the 10000 port is bounded for receiving UDP packets.
- The loop waits for a data packet and signals the main program with the **`emit()`** function sending the data received.

```
import sys
import socket
from PyQt6.QtWidgets import *
from PyQt6.QtCore import Qt, QTimer
import pyqtgraph as pg
from cbor2 import loads
from plotline import PlotLine

class Capture(QThread):
    data = pyqtSignal(bytes)

    def __init__(self):
        super().__init__()
        self.is_running = True

    @pyqtSlot()
    def run(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.bind(('192.168.4.16', 10000))
        sock.sendto(b'start', ('192.168.4.1', 10000))
        while self.is_running:
            d, a = sock.recvfrom(128)
            self.data.emit(d)

    def close(self):
        self.is_running = False
```

Plotting

- The **close()** function is called by the main program when the program is to be terminated. It sets the variable **self.is_running** to **False** so the while loop will exit and the **run()** function terminates.

```
import sys
import socket
from PyQt6.QtWidgets import *
from PyQt6.QtCore import Qt, QTimer
import pyqtgraph as pg
from cbor2 import loads
from plotline import PlotLine

class Capture(QThread):
    data = pyqtSignal(bytes)

    def __init__(self):
        super().__init__()
        self.is_running = True

    @pyqtSlot()
    def run(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.bind(('192.168.4.16', 10000))
        sock.sendto(b'start', ('192.168.4.1', 10000))
        while self.is_running:
            d, a = sock.recvfrom(128)
            self.data.emit(d)

    def close(self):
        self.is_running = False
```

Plotting

- Continuing with the plotting program, the class **Plotter(QWidget)** is created. This is the main program.
- The **__init__()** function sets the Window Title.
- The plot is created using **PlotLine()**.
- It is added to the window using the **QGridLayout**.
- The **Capture** thread is instantiated and started.
- Lastly, the data signal is connected to the **update_plot()** function which is next.

```
class Plotter(QWidget):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle("Plot Temperature")  
        self.tempP = PlotLine(200, 600, 200, "Temperature", (255, 0, 0))  
        layout = QGridLayout()  
        layout.addWidget(self.tempP, 0, 0)  
        self.setLayout(layout)  
        self.thread = Capture()  
        self.thread.start()  
        self.thread.data.connect(self.update_plot)
```

Plotting

- The next function is the **update_plot()** function receives the signal from the Capture thread in variable d.
- The data is in CBOR format and is converted to a dictionary object **telem** using the CBOR **loads()** function.
- The plot is updated using the temperature data.

```
def update_plot(self,d):  
    telem = loads(d)  
    self.tempP.update(telem['temp'])  
  
def closeEvent(self,event):  
    self.thread.close()  
    event.accept()  
  
app = QApplication(sys.argv)  
win = Plotter()  
win.show()  
sys.exit(app.exec())
```

Plotting

- The last function is the **closeEvent()**. It is called when the close button on the window is clicked.
- the function calls the **close()** function in the **Capture()** thread to stop the while loop.

```
def update_plot(self,d):  
    telem = loads(d)  
    self.tempP.update(telem['temp'])  
  
def closeEvent(self,event):  
    self.thread.close()  
    event.accept()  
  
app = QApplication(sys.argv)  
win = Plotter()  
win.show()  
sys.exit(app.exec())
```

Plotting

- This last part of the code sets up the PyQt application. The class is instantiated and assigned to the variable win.
- **win.show()** makes the window visible.
- The last line will exit when **app.exec()** exits.
- **app.exec()** is called in the **sys.exit()** function. This is the main loop running the program and plotting.

```
def update_plot(self,d):  
    telem = loads(d)  
    self.tempP.update(telem['temp'])  
  
def closeEvent(self,event):  
    self.thread.close()  
    event.accept()  
  
app = QApplication(sys.argv)  
win = Plotter()  
win.show()  
sys.exit(app.exec())
```

Plotting

- Start the xmit2.py program in Thonny.
- Connect the laptop WiFi to the SLATE.
- Run the plot1.py program in IDLE.
- The plot should start updating.

```
def update_plot(self,d):  
    telem = loads(d)  
    self.tempP.update(telem['temp'])  
  
def closeEvent(self,event):  
    self.thread.close()  
    event.accept()  
  
app = QApplication(sys.argv)  
win = Plotter()  
win.show()  
sys.exit(app.exec())
```

Two Plots

- Adding a second plot requires a few steps. Rename **plot1.py** to **plot2.py**.
- First, create a new PlotLine widget called **self.pressP** for the pressure data in the Plotter class.
- Add it to the layout as shown. The plot will be positioned below the temperature plot.

```
class Plotter(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Plot Temperature")
        self.tempP = PlotLine(200,600,200,"Temperature",(255,0,0))
        self.pressP = PlotLine(200,600,200,"Pressure",(0,255,0))
        layout = QGridLayout()
        layout.addWidget(self.tempP,0,0)
        layout.addWidget(self.pressP,1,0)
        self.setLayout(layout)
        self.thread = Capture()
        self.thread.start()
        self.thread.data.connect(self.update_plot)
```

Plotting

- The **update_plot()** function needs to be updated to update the second plot.
- The pressure data can be retrieved from the dictionary **telem**.
- The pressure plot is updated just as the temperature data.
- Try running the program again and two plots should appear.
- You can add a third plot for the humidity sensor.

```
def update_plot(self,d):  
    telem = loads(d)  
    self.tempP.update(telem['temp'])  
    self.presP.update(telem['press'])  
  
def closeEvent(self,event):  
    self.thread.close()  
    event.accept()  
  
app = QApplication(sys.argv)  
win = Plotter()  
win.show()  
sys.exit(app.exec())
```

Plotting

- Play with the plot2.py program. Try changing the size of the plots and the data ranges and see how the plots change.

