7/27/2020 : set up realtime plotting of the spectrum

step 1: add the plot function to the spectrum plot widget class

Add a plot function to the `SpectrumPlotWidget` class in `widgets.py`

before:

```
class SpectrumPlotWidget(pg.GraphicsLayoutWidget):
    def __init__(self, window_title='',parent=None):
        super().__init__(parent)
        self.plotWidget = self.addPlot(title = 'example')
        x = [1,2,3,4,5,6,7,8,9,10]
        y = [30,32,34,32,33,31,29,32,35,45]
        # plot data: x, y values
        self.plotWidget.plot(x,y)
```

after:

```
class SpectrumPlotWidget(pg.GraphicsLayoutWidget):
    def __init__(self, window_title='',parent=None):
        super().__init__(parent)
        self.plotWidget = self.addPlot(title = 'spectrum')

    def plot(x,y):
        # < add code here (hint: check the block above to see how to plot) >
```

step 2: add a spectrum extractor class

The class is used for instantiation of a spectrum extractor class, that handles extracting the spectrum and routing the spectrum for display. The class is located in `core.py`.

```
class SpectrumExtractor(QObject):

    packet_spectrum = Signal(np.array,np.array)

    def __init__(self):
        QObject.__init__(self)

    def extract_and_display_the_spectrum(self,raw_image):
        # < add the code for extracting the spectrum (get wavelength and intensity) >

        # send the spectrum for display
        packet_spectrum.emit(wavelength,intensity)
```

step 3: Instantiate an spectrum extractor object

Create a spectrum extractor object from the class just created in `gui_spectrometer.py`. See the last line of the code snippet below.

```
class OctopiGUI(QMainWindow):

  # variables
  fps_software_trigger = 100

  def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)

    # load objects
    self.camera = camera.Camera()
    self.microcontroller = microcontroller.Microcontroller_Simulation()

    self.streamHandler = core.StreamHandler()
    self.liveController = core.LiveController(self.camera,self.microcontroller)
    self.imageSaver = core.ImageSaver()
    self.imageDisplay = core.ImageDisplay()

    # instantiate an spectrum extractor object
    self.spectrumExtractor = core.SpectrumExtractor()
```

step 4: wire the new image to the spectrum extractor

Connect the `image_to_display` signal of `streamHandler` to `spectrumExtractor`'s `extract_and_display_the_spectrum` function in `gui_spectrometer.py`. See the last line of the code snippet below.

In the future we'll rename the `image_to_display` variable as it is not only for display.

```
    # make connections
    self.streamHandler.signal_new_frame_received.connect(self.liveController.on_new_frame)
    self.streamHandler.image_to_display.connect(self.imageDisplay.enqueue)
    self.streamHandler.packet_image_to_write.connect(self.imageSaver.enqueue)
    self.imageDisplay.image_to_display.connect(self.imageDisplayWindow.display_image) # may connect streamHandler directly to i
    # route the new image (once it has arrived) to the spectrumExtractor
    self.streamHandler.image_to_display.connect(self.spectrumExtractor.extract_and_display_the_spectrum)
```

step 5: wire the extracted spectrum for display

Connect the `packet_spectrum` signal of the `spectrumExtractor` to `spectrumDisplayWindow`'s `spectrumExtractor`'s `plotWidget`'s `plot` function. See the last line of the code snippet below.

```
    # make connections
    self.streamHandler.signal_new_frame_received.connect(self.liveController.on_new_frame)
    self.streamHandler.image_to_display.connect(self.imageDisplay.enqueue)
    self.streamHandler.packet_image_to_write.connect(self.imageSaver.enqueue)
    self.imageDisplay.image_to_display.connect(self.imageDisplayWindow.display_image) # may connect streamHandler directly to i
    self.streamHandler.image_to_display.connect(self.spectrumExtractor.extract_and_display_the_spectrum)
    self.spectrumExtractor.packet_spectrum.connect(self.spectrumDisplayWindow.plotWidget.plot)
```