



Artifact Information					
Artifact ID		Artifact Title			
SRC-001		Software Source Code			
Capstone Team			Revision	Artifact Date	
Capstone Team 27 - Granustem			1.1	Apr 13, 2019	
Prepared by			Checked by		
Jonathan Meldrum			Tanner Gaskin		
Revision History					
Revision #	Date	Prepared by	Checked by	Description	Approved by
1.0	Mar 01, 2019	Jonathan Meldrum	Tanner Gaskin	Software files as of Mar 1, 2019 from the team's GitHub repository	Reese Bastian
1.1	Apr 13, 2019	Jonathan Meldrum	Tanner Gaskin	Software files as of Apr 13, 2019 from the team's GitHub Repository	Reese Bastian

```
1 from connections import *
2
3 def accl_test():
4     print("\n *****Beginning ACCL Test***** \n")
5     sleep_time = .1
6     for i in range(int(3.0/sleep_time)):
7         x, y, z = [value / adafruit_lis3dh.STANDARD_GRAVITY for value in \
8                     lis3dh.acceleration]
9         print("x = %0.3f G, y = %0.3f G, z = %0.3f G" % (x, y, z))
10
11         sleep(sleep_time)
12
13     print("\n *****Ending ACCL Test***** \n")
14
15 if __name__ == "__main__":
16     accl_test()
17
```

```
1 from connections import *
2
3 def adc_test():
4     print("\n *****Beginning ADC Test***** \n")
5     for i in range(4):
6         print("CHAN0 Sample ", i, " value: ", CHAN0.value, " voltage:", CHAN0.voltage)
7         print("CHAN1 Sample ", i, " value: ", CHAN1.value, " voltage:", CHAN1.voltage)
8         print("CHAN2 Sample ", i, " value: ", CHAN2.value, " voltage:", CHAN2.voltage)
9         print("CHAN3 Sample ", i, " value: ", CHAN3.value, " voltage:", CHAN3.voltage)
10        print("CHAN4 Sample ", i, " value: ", CHAN4.value, " voltage:", CHAN4.voltage)
11        print("CHAN5 Sample ", i, " value: ", CHAN5.value, " voltage:", CHAN5.voltage)
12        print("CHAN6 Sample ", i, " value: ", CHAN6.value, " voltage:", CHAN6.voltage)
13        print("CHAN7 Sample ", i, " value: ", CHAN7.value, " voltage:", CHAN7.voltage)
14        print()
15        sleep(.5)
16    print("\n *****Ending ADC Test***** \n")
17
18
19 if __name__ == "__main__":
20     adc_test()
21
```

```
1 from kivy.uix.screenmanager import Screen
2
3 class BaseScreen(Screen):
4     '''Keeps track of the screen history to allow users to move to the previous screen,
5     rather than having to specify which screen to move to each time.'''
6
7     screen_history = []
8
9     def move_to(self, screen_name):
10        '''Add the current screen to the screen history and move to a new screen.  If we
11        are moving to the previous screen, remove it from the screen history.'''
12        if self.screen_history and self.screen_history[-1] == screen_name:
13            # Make sure to pop the stack if we're moving back to the previous screen
14            self.back()
15        else:
16            self.screen_history.append(self.name)
17            self.manager.current = screen_name
18
19    def back(self):
20        '''Go to the previous screen.'''
21        self.manager.current = self.screen_history.pop()
22
```

```
1 <BreakHeightScreen>:
2     name: 'break_height_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Save'
7                 on_release: if root.save(): root.back()
8             GranuSideButton:
9                 text: 'Cancel'
10                on_release:
11                    root.back()
12            GranuNone:
13            GranuNone:
14        GranuContent:
15            GranuTitle:
16                text: 'Break Height'
17            GridLayout:
18                cols: 2
19                rows: 1
20            FloatInput:
21                id: break_height
22                font_size: 30
23                size_hint_max_y: 30+15
24            Label:
25                text: "cm"
26                size_hint_x: 0.2
27                font_size: 30
28
```

```
1  """
2  An input text box that, when selected, allows the user to type in the Break Height value of
... the last test via a touch screen number pad that will pop up. The value in the input text box
... when you first visit this view is whatever value for the Height setting is currently stored
... in our settings file.
3  """
4
5  from kivy.lang import Builder
6
7  import configurator as config
8  from view.BaseScreen import BaseScreen
9  from view.input.FloatInput import FloatInput
10
11  Builder.load_file('view/screens/main/testing/BreakHeightScreen.kv')
12
13  class BreakHeightScreen(BaseScreen):
14      def on_pre_enter(self):
15          """Before the Screen loads, read the configuration file to get the current
16             height."""
17          input = self.ids['break_height']
18          input.text = str(config.get('break_height', 0))
19          input.validate()
20
21      def on_enter(self):
22          """Once the Screen loads, focus the TextInput"""
23          input = self.ids['break_height']
24          input.focus = True
25
26      def save(self):
27          """Save button was pressed: save the new height in the configuration file.
28             Returns True if save was successful. False otherwise."""
29          input = self.ids['break_height']
30          valid = input.validate()
31          if valid:
32              config.set('break_height', input.text)
33              return True
34          else:
35              input.focus = True
36              return False
37
38
```

```
1 <CalibratingPopup>:
2     title: 'Calibrating...'
3     title_align: 'center'
4     size_hint: (0.9, 0.9)
5     Label:
6         text: 'Reading ADC Data'
7
8 <CalibratePointScreen>:
9     name: 'calibrate_point_screen'
10    GranuContainer:
11        GranuSideArea:
12            GranuSideButton:
13                text: 'Add'
14                on_release:
15                    if root.add(): root.back()
16            GranuSideButton:
17                text: 'Cancel'
18                on_release:
19                    root.back()
20            GranuNone:
21            GranuSideButton:
22                text: 'Calibrate'
23                on_release:
24                    root.calibrate()
25        GranuContent:
26            GranuTitle:
27                text: 'Add Calibration Point'
28            GridLayout:
29                rows: 2
30                cols: 2
31                Label:
32                    text: 'ADC: '
33                FloatInput:
34                    id: adc
35                    font_size: 30
36                    multiline: False
37                    size_hint_max_y: 30+15
38                Label:
39                    text: 'Real: '
40                FloatInput:
41                    id: real
42                    font_size: 30
43                    multiline: False
44                    size_hint_max_y: 30+15
45
```

```
1 from kivy.lang import Builder
2 from kivy.clock import Clock
3 from kivy.uix.popup import Popup
4
5 import configurator as config
6 from Sensor import Sensor
7 from view.BaseScreen import BaseScreen
8 from view.input.StrInput import StrInput
9
10 import numpy
11
12 Builder.load_file('view/screens/settings/CalibratePointScreen.kv')
13
14 INTERVAL = .003
15 SECOND_CAP = 1/INTERVAL
16
17 class CalibratingPopup(Popup):
18     pass
19
20 class CalibratePointScreen(BaseScreen):
21     def __init__(self, **kwargs):
22         super(CalibratePointScreen, self).__init__(**kwargs)
23         self.sensors = Sensor()
24
25     def on_pre_enter(self):
26         adc_input = self.ids['adc']
27         adc_input.text = ''
28         real_input = self.ids['real']
29         real_input.text = ''
30
31     def add(self):
32         adc_input = self.ids['adc']
33         real_input = self.ids['real']
34         if adc_input.validate() and real_input.validate():
35             calib_screen = self.manager.get_screen('calibrate_screen')
36             calib_screen.add_point(float(adc_input.text), float(real_input.text))
37             return True
38         else:
39             return False
40
41     def calibrate(self):
42         calib_screen = self.manager.get_screen('calibrate_screen')
43         sensor = calib_screen.get_sensor()
44
45         self._popup = CalibratingPopup()
46         self._popup.open()
47
48         data_list = []
49         def update_data(dt):
50             data = self.sensors.get_all_data()
51             data_list.append(data[sensor])
52         event = Clock.schedule_interval(update_data, INTERVAL)
53         def calibrate_finish(dt):
54             event.cancel()
55             adc = numpy.float(numpy.average(data_list))
56             self.ids['adc'].text = str(adc)
57             self._popup.dismiss()
58         Clock.schedule_once(calibrate_finish, 1)
```



```
1 <PointDisplay>
2     canvas.before:
3         Color:
4             rgba: (.0, 0.9, .1, .3) if self.selected else (0, 0, 0, 1)
5         Rectangle:
6             pos: self.pos
7             size: self.size
8     text: '(' + str(root.adc) + ', ' + str(root.real) + ')'
9
10 <PointsList>
11     viewclass: 'PointDisplay'
12     SelectableRecycleBoxLayout:
13
14
15 <PointListTitle@Label>
16     size_hint_y: None
17     height: self.font_size + 5
18     font_size: 15
19
20 <CalibrateScreen>:
21     name: 'calibrate_screen'
22     GranuContainer:
23         GranuSideArea:
24             GranuSideButton:
25                 text: 'Save'
26                 on_release:
27                     if root.save(): root.back()
28             GranuSideButton:
29                 text: 'Add Point'
30                 on_release:
31                     root.move_to('calibrate_point_screen')
32             GranuNone:
33             GranuSideButton:
34                 text: 'Cancel'
35                 on_release:
36                     root.back()
37         GranuContent:
38             GranuTitle:
39                 text: root.sensor_name + ' Calibration'
40             GridLayout:
41                 cols: 2
42                 rows: 1
43                 size_hint_max_y: 20+15
44                 Label:
45                     text: "Units:"
46                     size_hint_x: 0.2
47                     font_size: 20
48                 StrInput:
49                     id: units
50                     text: root.units
51                     width: 20
52                     font_size: 20
53                     size_hint_max_y: 20+15
54             PointListTitle:
55                 text: 'Calibration Points'
56             PointListTitle:
57                 text: '(ADC, Real)'
58             PointsList:
59                 id: point_list
60                 list_data: root.points_list
61             Label:
62                 size_hint_y: None
63                 height: self.font_size + 5
64                 font_size: 20
65                 text: 'real = ' + "{:.3E}".format(root.slope) + '*adc + ' +
... "{:.3E}".format(root.intercept)
```

```
1 from kivy.lang import Builder
2 from kivy.clock import Clock
3 from kivy.uix.button import Button
4 from kivy.uix.label import Label
5 from kivy.uix.boxlayout import BoxLayout
6 from kivy.properties import StringProperty, ListProperty, NumericProperty
7
8 from view.BaseScreen import BaseScreen
9 from view.SelectableList import SelectableList, SelectableListBehavior,
... SelectableRecycleBoxLayout
10
11 import numpy
12
13 import configurator as config
14
15 Builder.load_file('view/screens/settings/CalibrateScreen.kv')
16
17 class PointDisplay(SelectableListBehavior, Label):
18     adc = NumericProperty()
19     real = NumericProperty()
20
21 class PointsList(SelectableList):
22     def update(self, k, val):
23         self.data = [{'adc': x[0], 'real': x[1]} for x in self.list_data]
24
25 class CalibrateScreen(BaseScreen):
26     sensor_name = StringProperty()
27     points_list = ListProperty()
28     slope = NumericProperty()
29     intercept = NumericProperty()
30     units = StringProperty()
31
32     def __init__(self, **kwargs):
33         super(CalibrateScreen, self).__init__(**kwargs)
34         self.config_data = {}
35
36     def set_sensor(self, name):
37         self.sensor_name = name
38         self.config_data = config.get('sensors', {})
39         if name in self.config_data:
40             self.points_list = self.config_data[name]['points_list']
41             self.slope = self.config_data[name]['slope']
42             self.intercept = self.config_data[name]['intercept']
43             self.ids['units'].text = self.config_data[name]['units']
44         else:
45             self.points_list = []
46             self.slope = 1
47             self.intercept = 0
48             self.ids['units'].text = ''
49
50     def get_sensor(self):
51         return self.sensor_name
52
53     def add_point(self, adc, real):
54         self.points_list.append((adc, real))
55         # Calculate line of best fit using Least Square Method
56         adc_points = [x[0] for x in self.points_list]
57         real_points = [x[1] for x in self.points_list]
58         if len(self.points_list) > 1:
59             poly = numpy.polyfit(adc_points, real_points, 1) # Linear regression
60             self.slope = numpy.float(poly[0])
61             self.intercept = numpy.float(poly[1])
62         else:
63             self.slope = 1.0
64             self.intercept = 0.0
65
66     def save(self):
67         self.config_data[self.sensor_name] = {
```

```
68         'slope': self.slope,  
69         'intercept': self.intercept,  
70         'points_list': self.points_list,  
71         'units': self.ids['units'].text  
72     }  
73     config.set('sensors', self.config_data)  
74     return True
```

```
1 <ColorsScreen>:
2     name: 'colors_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Back'
7                 on_release: root.back()
8     GranuContent:
9         GranuTitle:
10             text: 'Colors'
11         ColorPicker:
12             id: 'picker'
13
```

```
1 from kivy.lang import Builder
2
3 from view.BaseScreen import BaseScreen
4
5 Builder.load_file('view/screens/settings/ColorsScreen.kv')
6
7 class ColorsScreen(BaseScreen):
8     pass
9
```

```
1  """
2  The configurator module adds an interface to read to and write from the configuration
3  file ('config.json'). Reading and writing settings to the configuration file allows the
4  configuration to persist beyond the application lifecycle.
5  """
6
7  import os
8  import json
9
10 CONFIG_FILE = 'config.json'
11
12 data = {}
13
14 def load():
15     """Loads data from the configuration file, if it exists."""
16     global data
17     if os.path.isfile(CONFIG_FILE):
18         with open(CONFIG_FILE) as f:
19             data.update(json.load(f))
20     else:
21         data = {}
22
23 def load_from(filepath):
24     """Loads data from a specified configuration file. Overwrites CONFIG_FILE"""
25     global data
26     if os.path.isfile(filepath):
27         with open(filepath) as f:
28             data.update(json.load(f))
29             save()
30     else:
31         data = {}
32
33 def save():
34     """Saves data to the configuration file."""
35     with open(CONFIG_FILE, 'w') as outfile:
36         json.dump(data, outfile, indent=4)
37
38 def save_as(filepath):
39     """Saves data to the specified file."""
40     with open(filepath, 'w') as outfile:
41         json.dump(data, outfile, indent=4)
42
43 def set(key, value):
44     """Set a key to value in the configuration JSON file."""
45     # Set key to value
46     data[key] = value
47     save()
48
49 def get(key, default):
50     """Get a value from the configuration JSON file using a key. If the value does not
51     exist, save the default value into the JSON file and return the default."""
52     if key in data:
53         return data.get(key)
54     else:
55         set(key, default)
56         return default
57
58 if __name__ == "__main__":
59     """If the configuration module is run as the main program, test the configuration
60     module. These tests ensure the module returns and saves the default value if a key
61     is not defined, returns the saved value if a key is defined, and that the
62     configuration file contains the values saved.
63
64     WARNING: This will override the configuration file."""
65     assert get('a', 3) == 3
66     set('b', 5)
67     assert get('b', 1) == 5
68     set('a', 9)
```

```
69     save()
70     get('a', 33)
71     get('b', 33)
72     load()
73     assert get('a', 2) == 9
74     get('c', 21) # The file should now contain 'c': 21
75     print("Check that the configuration file contains the key-value pair 'c': 21")
76
```

```
1 import time
2 import board
3 import busio
4 import digitalio
5 from time import sleep
6 import RPi.GPIO as GPIO
7 import serial
8 import adafruit_gps
9 import adafruit_lis3dh
10 import adafruit_am2320
11 import adafruit_ads1x15.ads1015 as ADS
12 from adafruit_ads1x15.analog_in import AnalogIn
13
14 GPIO.setmode(GPIO.BCM)
15 i2c = busio.I2C(board.SCL, board.SDA, 115200)
16 uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=3000)
17
18 # Temperature and Humidity sensor, off of the I2C pins on bottom right of board
19 am = adafruit_am2320.AM2320(i2c)
20
21 # Accelerometer, top middle of board
22 lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c)
23 lis3dh.range = adafruit_lis3dh.RANGE_2_G
24
25
26 # GPIO, right edge of board
27 GPIO1 = 4 #BOARD 7, BCM 4
28 GPIO2 = 17 #BOARD 11, BCM 17
29 GPIO3 = 27 #BOARD 13, BCM 27
30 GPIO4 = 22 #BOARD 15, BCM 22
31
32 # SPI/GPIO, top edge of board
33 SPI_CE1 = 7 #BOARD 26, BCM 7
34 SPI_CE0 = 8 #BOARD 24, BCM 8
35 SPI_SCLK = 11 #BOARD 23, BCM 11
36 SPI_MISO = 9 #BOARD 21, BCM 9
37 SPI_MOSI = 10 #BOARD 19, BCM 10
38
39 GPIO_PINS = [GPIO1, GPIO2, GPIO3, GPIO4, SPI_CE1, SPI_CE0,\
40             SPI_SCLK, SPI_MISO, SPI_MOSI]
41
42 for pin in GPIO_PINS:
43     GPIO.setup(pin, GPIO.OUT)
44
45
46 # ADC
47 ads1 = ADS.ADS1015(i2c, address=0x49, data_rate = 3300, mode=0)
48 ads2 = ADS.ADS1015(i2c, address=0x48, data_rate = 3300, mode=0)
49 CHAN0 = AnalogIn(ads1, ADS.P0)
50 CHAN1 = AnalogIn(ads1, ADS.P1)
51 CHAN2 = AnalogIn(ads1, ADS.P2)
52 CHAN3 = AnalogIn(ads1, ADS.P3)
53 CHAN4 = AnalogIn(ads2, ADS.P0, ADS.P1)
54 # CHAN5 = AnalogIn(ads2, ADS.P1)
55 CHAN6 = AnalogIn(ads2, ADS.P3)
56 CHAN7 = AnalogIn(ads2, ADS.P2)
57
58 # Channels for the pot and force sensors
59 POT_CHAN = CHAN3
60 X_LOAD_CHAN = CHAN4
61 Y_LOAD_CHAN = CHAN2
62
63 # Scaling factor for the force sensor
64 FORCE_SENSOR_SCALING = 3556.1878
65
66 # GPS
67 gps = adafruit_gps.GPS(uart, debug=False)
68 gps.send_command(b'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
```



```
69 gps.send_command(b'PMTK220,1000')
70 try:
71     gps.update()
72 except:
73     print("GPS may have a problem. Try Rebooting")
74
75 # MOTORS
76 PWMA = 16 #BOARD 36, BCM 16
77 AIN1 = 25 #BOARD 22, BCM 25
78 AIN2 = 20 #BOARD 38, BCM 20
79
80 PWMB = 21 #BOARD 40, BCM 21
81 BIN1 = 5 #BOARD 29, BCM 5
82 BIN2 = 12 #BOARD 32, BCM 12
83
84 PWMC = 13 #BOARD 33, BCM 13
85 CIN1 = 26 #BOARD 37, BCM 26
86 CIN2 = 19 #BOARD 35, BCM 19
87
88 PWMD = 24 #BOARD 18, BCM 24
89 DIN1 = 18 #BOARD 12, BCM 18
90 DIN2 = 23 #BOARD 16, BCM 23
91
92 MOTORS = ['A', 'B', 'C', 'D']
93 IN1 = [AIN1, BIN1, CIN1, DIN1]
94 IN2 = [AIN2, BIN2, CIN2, DIN2]
95 PWM = [PWMA, PWMB, PWMC, PWMD]
96
97 for in1, in2, pwm in zip(IN1, IN2, PWM):
98     GPIO.setup(in1, GPIO.OUT)
99     GPIO.setup(in2, GPIO.OUT)
100     GPIO.setup(pwm, GPIO.OUT)
101
```

```
1 import datetime
2
3 class Dataset:
4
5     def __init__(self, timestamp, x_load, y_load, pot_angle, imu_angle, data_rate):
6         self.timestamp = timestamp
7         self.x_load = x_load
8         self.y_load = y_load
9         self.pot_angle = pot_angle
10        self.imu_angle = imu_angle
11
```

```
1 <GranuContainer>
2     orientation: 'horizontal'
3     padding: 10
4
5 <GranuSideArea>
6     rows: 4
7     spacing: [0, 10]
8     row_default_height: self.height/4 - (30/4.)
9     row_force_default: True
10    size_hint_x: 0.37
11
12 <GranuSideButton>
13     font_size: 40
14     halign: 'center'
15     valign: 'middle'
16
17 <GranuContent>
18     orientation: 'vertical'
19     padding: [10, 0, 0, 0]
20     spacing: 10
21
22 <GranuTitle>
23     size_hint_max_y: 50
24     font_size: 40
25
26
```

```
1 from kivy.lang import Builder
2
3 from kivy.uix.boxlayout import BoxLayout
4 from kivy.uix.gridlayout import GridLayout
5 from kivy.uix.button import Button
6 from kivy.uix.widget import Widget
7 from kivy.uix.label import Label
8
9 Builder.load_file('view/elements.kv')
10 class GranuContainer(BoxLayout):
11     pass
12
13 class GranuSideArea(GridLayout):
14     pass
15
16 class GranuSideButton(Button):
17     pass
18
19 class GranuNone(Widget):
20     pass
21
22 class GranuContent(BoxLayout):
23     pass
24
25 class GranuTitle(Label):
26     pass
27
```

```
1 <ExitScreen>
2     name: 'exit_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Cancel'
7                 on_release: root.move_to('main_screen')
8             GranuSideButton:
9                 text: 'Exit'
10                on_release: app.stop()
11            GranuSideButton:
12                text: 'Restart'
13                on_release:
14                    app.stop()
15                    app.run()
16            GranuSideButton:
17                text: 'Shutdown'
18                on_release: root.move_to('main_screen')
19        GranuContent:
20
21
```

```
1  """
2  Four buttons to select from: Back, Exit, Restart, and Shut Down
3  """
4
5  from kivy.lang import Builder
6
7  from view.BaseScreen import BaseScreen
8
9  Builder.load_file('view/screens/main/ExitScreen.kv')
10
11  class ExitScreen(BaseScreen):
12      pass
13
```

```
1 from kivy.clock import Clock
2 from kivy.uix.textinput import TextInput
3
4 import view.keyboard_man as km
5
6 class FloatInput(TextInput):
7     def __init__(self, **kwargs):
8         '''Floats do not need multiple lines to input, set multiline property to
9         false.'''
10        super(FloatInput, self).__init__(**kwargs)
11        self.multiline = False
12
13    def validate(self):
14        '''Check that the input can be cast as an int.'''
15        test = False
16        try:
17            fl = float(self.text)
18            test = True
19        except:
20            test = False
21        if test:
22            self.background_color = (1, 1, 1, 1)
23        else:
24            self.background_color = (1, .7, .7, 1)
25        return test
26
27    def on_text_validate(self):
28        '''Called when enter is pressed.'''
29        self.validate()
30        Clock.schedule_once(self.focus_and_select)
31
32    def on_focus(self, instance, value):
33        '''When the FloatInput is focused, show a numeric keyboard.'''
34        if value:
35            km.show_keyboard(self, 'numeric')
36
37    def focus_and_select(self, *args):
38        '''Focus the TextInput and select all of its text.'''
39        self.focus = True
40        self.select_all()
41
```

```
1 from connections import *
2
3 def gpio_test():
4     print("\n *****Beginning GPIO Test***** \n")
5
6     for i in range(3):
7         print("Blink: ", i)
8         for pin in GPIO_PINS:
9             GPIO.output(pin, GPIO.HIGH)
10            sleep(.5)
11            for pin in GPIO_PINS:
12                GPIO.output(pin, GPIO.LOW)
13                sleep(.5)
14
15    print("\n *****Ending GPIO Test***** \n")
16
17
18 if __name__ == "__main__":
19     gpio_test()
20
```



```
1 from connections import *
2
3 def gps_test():
4     print("\n *****Beginning GPS Test***** \n")
5
6
7
8     # Initialize the GPS module by changing what data it sends and at what rate.
9     # These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
10    # PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
11    # the GPS module behavior:
12    # https://cdn-shop.adafruit.com/datasheets/PMTK_All.pdf
13
14    # Turn on the basic GGA and RMC info (what you typically want)
15    # gps.send_command(b'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0')
16    # Turn on just minimum info (RMC only, location):
17    #gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
18    # Turn off everything:
19    #gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
20    # Tuen on everything (not all of it is parsed!)
21    #gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0')
22
23    # Set update rate to once a second (1hz) which is what you typically want.
24    # gps.send_command(b'PMTK220,1000')
25    # Or decrease to once every two seconds by doubling the millisecond value.
26    # Be sure to also increase your UART timeout above!
27    #gps.send_command(b'PMTK220,2000')
28    # You can also speed up the rate, but don't go too fast or else you can lose
29    # data during parsing. This would be twice a second (2hz, 500ms delay):
30    #gps.send_command(b'PMTK220,500')
31
32    # Main loop runs forever printing the location, etc. every second.
33    last_print = time.monotonic()
34    cnt = 0
35    while cnt < 4:
36        cnt = cnt + 1
37        # Make sure to call gps.update() every loop iteration and at least twice
38        # as fast as data comes from the GPS unit (usually every second).
39        # This returns a bool that's true if it parsed new data (you can ignore it
40        # though if you don't care and instead look at the has_fix property).
41        gps.update()
42        # Every second print out current location details if there's a fix.
43        current = time.monotonic()
44        if current - last_print >= 1.0:
45            last_print = current
46            if not gps.has_fix:
47                # Try again if we don't have a fix yet.
48                print('Waiting for fix...')
49                continue
50            # We have a fix! (gps.has_fix is true)
51            # Print out details about the fix like location, date, etc.
52            print('=' * 40) # Print a separator line.
53            print('Fix timestamp: {}/{}/{} {:02}:{:02}:{:02}'.format(
54                gps.timestamp_utc.tm_mon, # Grab parts of the time from the
55                gps.timestamp_utc.tm_mday, # struct_time object that holds
56                gps.timestamp_utc.tm_year, # the fix time. Note you might
57                gps.timestamp_utc.tm_hour, # not get all data like year, day,
58                gps.timestamp_utc.tm_min, # month!
59                gps.timestamp_utc.tm_sec))
60            print('Latitude: {0:.6f} degrees'.format(gps.latitude))
61            print('Longitude: {0:.6f} degrees'.format(gps.longitude))
62            print('Fix quality: {}'.format(gps.fix_quality))
63            # Some attributes beyond latitude, longitude and timestamp are optional
64            # and might not be present. Check if they're None before trying to use!
65            if gps.satellites is not None:
66                print('# satellites: {}'.format(gps.satellites))
67            if gps.altitude_m is not None:
68                print('Altitude: {} meters'.format(gps.altitude_m))
```

```
69         if gps.speed_knots is not None:
70             print('Speed: {} knots'.format(gps.speed_knots))
71         if gps.track_angle_deg is not None:
72             print('Track angle: {} degrees'.format(gps.track_angle_deg))
73         if gps.horizontal_dilution is not None:
74             print('Horizontal dilution: {}'.format(gps.horizontal_dilution))
75         if gps.height_geoid is not None:
76             print('Height geo ID: {} meters'.format(gps.height_geoid))
77
78         # print('{0:.6f}, {1:.6f}, sample_{2:d}, #FF0000'.format(gps.latitude,
... gps.longitude, cnt))
79         # print('{0:.6f}, {1:.6f}'.format(gps.latitude, gps.longitude))
80         print("\n *****Ending GPS Test***** \n")
81
82
83 if __name__ == "__main__":
84     gps_test()
85
```

```
1 from math import sin
2 from kivy.garden.graph import Graph, MeshLinePlot
3 graph = Graph(xlabel='X', ylabel='Y', x_ticks_minor=5,
4 x_ticks_major=25, y_ticks_major=1,
5 y_grid_label=True, x_grid_label=True, padding=5,
6 x_grid=True, y_grid=True, xmin=-0, xmax=100, ymin=-1, ymax=1)
7 plot = MeshLinePlot(color=[1, 0, 0, 1])
8 plot.points = [(x, sin(x / 10.)) for x in range(0, 101)]
9 graph.add_plot(plot)
```

```
1 <HeightScreen>:
2     name: 'height_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Save'
7                 on_release: if root.save(): root.back()
8             GranuSideButton:
9                 text: 'Cancel'
10                on_release:
11                    root.back()
12            GranuNone:
13            GranuNone:
14        GranuContent:
15            GranuTitle:
16                text: 'Height'
17            GridLayout:
18                cols: 2
19                rows: 1
20            FloatInput:
21                id: height
22                font_size: 30
23                size_hint_max_y: 30+15
24            Label:
25                text: "cm"
26                size_hint_x: 0.2
27                font_size: 30
28
```

```
1  """
2  An input text box that, when selected, allows the user to type in the current Height
3  setting via a touch screen number pad that will pop up. The value in the input text box
4  when you first visit this view is whatever value for the Height setting is currently
5  stored in our settings file.
6  """
7
8  from kivy.lang import Builder
9
10 import configurator as config
11 from view.BaseScreen import BaseScreen
12 from view.input.FloatInput import FloatInput
13
14 Builder.load_file('view/screens/settings/HeightScreen.kv')
15
16 class HeightScreen(BaseScreen):
17     def on_pre_enter(self):
18         """Before the Screen loads, read the configuration file to get the current
19         height."""
20         input = self.ids['height']
21         input.text = str(config.get('height', 0))
22         input.validate()
23
24     def on_enter(self):
25         """Once the Screen loads, focus the TextInput"""
26         input = self.ids['height']
27         input.focus = True
28         input.select_all()
29
30     def save(self):
31         """Save button was pressed: save the new height in the configuration file.
32         Returns True if save was successful. False otherwise."""
33         input = self.ids['height']
34         valid = input.validate()
35         if valid:
36             config.set('height', float(input.text))
37             return True
38         else:
39             input.focus = True
40             return False
41
42
```

```
1 from .connections import *
2
3 class Humidity:
4
5     def __init__(self):
6         self.hum = 0.0
7
8     def get_data(self):
9         try:
10             self.hum = am.relative_humidity
11             return self.hum
12         except:
13             return self.hum
14
```

```
1 from .connections import *
2
3 class IMU:
4
5     def __init__(self):
6         pass
7
8     def get_data(self):
9         x, y, z = [value / adafruit_lis3dh.STANDARD_GRAVITY for value in \
10                    lis3dh.acceleration]
11         # return "x = {0:0.3f} G \n y = {1:0.3f} G \n z = {2:0.3f} G".format(x, y, z)
12         return x
13
```

```
1 {
2   "title" : "Integer",
3   "description" : "An integer-only keypad",
4   "cols" : 3,
5   "rows" : 4,
6   "normal_1": [
7     ["7", "7", "7", 1],
8     ["8", "8", "8", 1],
9     ["9", "9", "9", 1]],
10  "normal_2": [
11    ["4", "4", "4", 1],
12    ["5", "5", "5", 1],
13    ["6", "6", "6", 1]],
14  "normal_3": [
15    ["1", "1", "1", 1],
16    ["2", "2", "2", 1],
17    ["3", "3", "3", 1]],
18  "normal_4": [
19    ["0", "0", "0", 1],
20    [" ", null, "", 1],
21    ["\u232b", null, "backspace", 1]],
22  "shift_1": [
23    ["7", "7", "7", 1],
24    ["8", "8", "8", 1],
25    ["9", "9", "9", 1]],
26  "shift_2": [
27    ["4", "4", "4", 1],
28    ["5", "5", "5", 1],
29    ["6", "6", "6", 1]],
30  "shift_3": [
31    ["1", "1", "1", 1],
32    ["2", "2", "2", 1],
33    ["3", "3", "3", 1]],
34  "shift_4": [
35    ["0", "0", "0", 1],
36    [" ", null, "", 1],
37    ["\u232b", null, "backspace", 1]]
38 }
39
```



```
1 from kivy.clock import Clock
2 from kivy.uix.textinput import TextInput
3
4 import view.keyboard_man as km
5
6 class IntInput(TextInput):
7     def __init__(self, **kwargs):
8         '''Ints do not need multiple lines to input, set multiline property to
9         false.'''
10        super(IntInput, self).__init__(**kwargs)
11        self.multiline = False
12
13    def validate(self):
14        '''Check that the input can be cast as an int.'''
15        test = False
16        try:
17            fl = int(self.text)
18            test = True
19        except:
20            test = False
21        if test:
22            self.background_color = (1, 1, 1, 1)
23        else:
24            self.background_color = (1, .7, .7, 1)
25        return test
26
27    def on_text_validate(self):
28        '''Called when enter is pressed.'''
29        self.validate()
30        Clock.schedule_once(self.focus_and_select)
31
32    def on_focus(self, instance, value):
33        '''When the IntInput is focused, show a numeric keyboard.'''
34        if value:
35            km.show_keyboard(self, 'integer')
36
37    def focus_and_select(self, *args):
38        '''Focus the TextInput and select all of its text.'''
39        self.focus = True
40        self.select_all()
41
```

```
1  """
2  The keyboard manager adds the ability to change keyboard layouts depending on the current
3  screen. For example, the plot number screen uses a "numeric" keyboard (number pad),
4  while the operator uses a "text" keyboard (qwerty).
5  """
6
7  from kivy.core.window import Window
8
9  keyboard = None
10
11 def show_keyboard(caller, layout):
12     """Shows a keyboard with the specified layout.
13
14     The folder view/keyboard_layouts contains keyboard layouts used in this software,
15     including 'numeric' (number pad) and 'text' (qwerty). These are custom keyboard
16     layouts. Kivy contains its own keyboard layouts; however, Kivy's layouts are
17     designed for multi-touch screens."""
18     kb = Window.request_keyboard(_close_keyboard, caller)
19     if kb.widget:
20         keyboard = kb.widget
21         if layout=='numeric':
22             keyboard.layout = "view/keyboard_layouts/numeric.json"
23             keyboard.margin_hint = [0.05, 0.2, 0.05, 0.2]
24         elif layout=='integer':
25             keyboard.layout = "view/keyboard_layouts/integer.json"
26             keyboard.margin_hint = [0.05, 0.2, 0.05, 0.2]
27         elif layout=='text':
28             keyboard.layout = "view/keyboard_layouts/text.json"
29             keyboard.margin_hint = [0.05, 0.06, 0.05, 0.06]
30         else:
31             keyboard.layout = layout
32             keyboard.margin_hint = [0.05, 0.06, 0.05, 0.06]
33
34         # Using internal members is probably not the best way to do this. But...
35         # Turn off capslock each time a new keyboard is requested
36         keyboard.have_capslock = False
37         keyboard.active_keys.clear()
38         keyboard.refresh_active_keys_layer()
39     else:
40         keyboard = kb
41
42 def _close_keyboard():
43     """When the keyboard is closed, clear the keyboard global."""
44     global keyboard
45     if keyboard:
46         keyboard = None
47
```

```
1 <LiveFeedInfoBox@Label>:
2     font_size: 25
3     color: 0,0,0,1
4     halign: 'center'
5     valign: 'center'
6     canvas.before:
7         Color:
8             rgba: .74,.74,.74,1
9         Rectangle:
10             pos: self.pos
11             size: self.size
12
13 <LiveFeedScreen>:
14     name: 'live_feed_screen'
15     GranuContainer:
16         GranuSideArea:
17             GranuNone:
18             GranuNone:
19             GranuSideButton:
20                 text: root.transition_to_state
21                 on_release:
22                     root.transition()
23             GranuSideButton:
24                 text: 'Back'
25                 on_release:
26                     root.manager.transition.direction = 'right'
27                     root.move_to('main_screen') # Move to main screen
28     GranuContent:
29         GranuTitle:
30             text: 'Live Feed'
31         GridLayout:
32             cols: 3
33             spacing: 10
34             LiveFeedInfoBox:
35                 id: temperature
36                 text: root.temperature_label + ':\n' + root.temperature + u'\N{DEGREE
... SIGN}' + "C"
37             LiveFeedInfoBox:
38                 id: humidity
39                 text: root.humidity_label + ':\n' + root.humidity + "%"
40             LiveFeedInfoBox:
41                 id: location
42                 text: root.location_label + ':\n' + root.location
43             LiveFeedInfoBox:
44                 id: time
45                 text: root.time_label + ':\n' + root.time
46             LiveFeedInfoBox:
47                 id: x_load
48                 text: root.x_load_label + ':\n' + root.x_load
49             LiveFeedInfoBox:
50                 id: y_load
51                 text: root.y_load_label + ':\n' + root.y_load
52             LiveFeedInfoBox:
53                 id: pot_angle
54                 text: root.pot_angle_label + ':\n' + root.pot_angle
55             LiveFeedInfoBox:
56                 id: imu_angle
57                 text: root.imu_angle_label + ':\n' + root.imu_angle + "G"
58             LiveFeedInfoBox:
59                 id: cpu_time
60                 text: root.data_rate_label + ':\n' + root.data_rate + " Hz"
61
```

```
1  """
2  Shows all data: Temperature, Humidity, Location, Time, and all Sensor data
3  """
4
5  from kivy.lang import Builder
6  from kivy.properties import NumericProperty
7  from kivy.properties import StringProperty
8  from kivy.properties import ListProperty
9  from kivy.clock import Clock
10 from Sensor import Sensor
11 import datetime
12
13 from view.BaseScreen import BaseScreen
14 from view.elements import *
15
16
17 Builder.load_file('view/screens/main/LiveFeedScreen.kv')
18
19 INTERVAL = .004
20 SECOND_CAP = 1/INTERVAL
21
22 class LiveFeedScreen(BaseScreen):
23     sensor = Sensor()
24
25     run_count = 0
26     transition_to_state = StringProperty("Pause")
27     #self.keys = ListProperty()
28     #self.values =
29     #sensor_keys = self.sensor.get_sensor_keys()
30     #for key in sensor_keys:
31     #    self.keys.append(keys)
32     #sensor_data = self.sensor.get_sensor_data()
33     #for i in range(0,len(sensor_data)):
34
35
36
37     temperature_label = StringProperty("Temperature")
38     humidity_label = StringProperty("Humidity")
39     location_label = StringProperty("Location")
40     time_label = StringProperty("Time")
41     x_load_label = StringProperty("X Load")
42     y_load_label = StringProperty("Y Load")
43     pot_angle_label = StringProperty("Pot Angle")
44     imu_angle_label = StringProperty("IMU Angle")
45     data_rate_label = StringProperty("Data Rate")
46
47     temperature = StringProperty("0")
48     humidity = StringProperty("0")
49     location = StringProperty("0.00, 0.00")
50     time = StringProperty("00:00:00 AM")
51     x_load = StringProperty("0.00")
52     y_load = StringProperty("0.00")
53     pot_angle = StringProperty("0")
54     imu_angle = StringProperty("0")
55     data_rate = StringProperty("0")
56     old_time = 0
57
58     def on_pre_enter(self):
59         self.event = Clock.schedule_interval(self.update_values, INTERVAL)
60         self.transition_to_state = "Pause"
61
62     def update_values(self, obj):
63
64         if self.run_count >= SECOND_CAP:
65             self.sensor.get_header_data()
66             sensor_data = self.sensor.get_sensor_data()
67             self.temperature = str(sensor_data["Temperature"])
68             self.humidity = str(sensor_data["Humidity"])
```

```
69         self.location = str(sensor_data["Location"])
70         self.time = datetime.datetime.now().strftime("%H:%M:%S %p")
71         self.x_load = str(sensor_data["X Load"])
72         self.y_load = str(sensor_data["Y Load"])
73         self.pot_angle = str(sensor_data["Pot Angle"])
74         self.imu_angle = str(sensor_data["IMU Angle"])
75         # Calculate Data Acquisition Rate
76         now = datetime.datetime.now()
77         new_time = (int(now.strftime("%M")) * 60) + int(now.strftime("%S")) +
... (int(now.strftime("%f"))/1000000)
78         time_dif = new_time - self.old_time
79         self.data_rate = str(round(SECOND_CAP/time_dif,2))
80         self.old_time = new_time
81         # Reset run_count
82         self.run_count = 0
83     else:
84         sensor_data = self.sensor.get_sensor_data()
85         self.run_count = self.run_count + 1
86
87     def on_leave(self):
88         self.event.cancel()
89
90     def transition(self):
91         if(self.transition_to_state == "Pause"):
92             self.event.cancel()
93             self.transition_to_state = "Resume"
94         else:
95             self.event = Clock.schedule_interval(self.update_values, INTERVAL)
96             self.transition_to_state = "Pause"
97
```

```
1 from .connections import *
2
3 class Location:
4
5     def __init__(self):
6         self.lat = 0.0
7         self.long = 0.0
8         gps.update()
9         if gps.has_fix:
10             self.lat = gps.latitude
11             self.long = gps.longitude
12
13     def get_data(self):
14         gps.update()
15         if gps.has_fix:
16             self.lat = gps.latitude
17             self.long = gps.longitude
18         return self.lat, self.long
19
```

```
1 # Include custom Kivy widgets for all screens
2 #:import * view.elements
3
4 #:import ExitScreen view.screens.main.ExitScreen
5 #:import LiveFeedScreen view.screens.main.LiveFeedScreen
6 #:import MainScreen view.screens.main.MainScreen
7 #:import SettingsScreen view.screens.main.SettingsScreen
8 #:import TestingScreen view.screens.main.TestingScreen
9
10 #:import ColorsScreen view.screens.settings.ColorsScreen
11 #:import HeightScreen view.screens.settings.HeightScreen
12 #:import NoteScreen view.screens.settings.NoteScreen
13 #:import OperatorScreen view.screens.settings.OperatorScreen
14 #:import PlotScreen view.screens.settings.PlotScreen
15 #:import SensorsScreen view.screens.settings.SensorsScreen
16 #:import UpdateScreen view.screens.settings.UpdateScreen
17
18 #:import NewNoteScreen view.screens.settings.NewNoteScreen
19
20 #:import CalibrateScreen view.screens.settings.CalibrateScreen
21 #:import CalibratePointScreen view.screens.settings.CalibratePointScreen
22
23 #:import TestInProgressScreen view.screens.main.testing.TestInProgressScreen
24 #:import TestingResultsScreen view.screens.main.testing.TestingResultsScreen
25 #:import BreakHeightScreen view.screens.main.testing.BreakHeightScreen
26 #:import TestsScreen view.screens.main.testing.TestsScreen
27
28 <GranuScreenManager>:
29     # Properties
30     current: 'main_screen' # Start with the main screen
31
32     # Root
33     ExitScreen:
34     LiveFeedScreen:
35     MainScreen:
36     SettingsScreen:
37     TestingScreen:
38
39     # Settings
40     ColorsScreen:
41     HeightScreen:
42     NoteScreen:
43     OperatorScreen:
44     PlotScreen:
45     SensorsScreen:
46     UpdateScreen:
47
48     # Settings - Notes
49     NewNoteScreen:
50
51     # Settings - Sensors
52     CalibrateScreen:
53     CalibratePointScreen:
54
55     # Testing
56     TestInProgressScreen:
57     TestingResultsScreen:
58     BreakHeightScreen:
59     TestsScreen:
60
```

```
1 from kivy.config import Config as KivyConfig
2 from kivy.app import App
3 from kivy.uix.screenmanager import ScreenManager, FadeTransition
4
5 import configurator as config
6
7 # Kivy Configuration
8 KivyConfig.set('kivy', 'desktop', 0) # Disable OS-specific features for testing
9 KivyConfig.set('kivy', 'keyboard_mode', 'systemanddock') # Allow barcode scanner and
10                                     # on screen keyboard
11 KivyConfig.set('graphics', 'height', 480) # Set window size to be the same as touchscreen
12 KivyConfig.set('graphics', 'width', 800) # (not used when fullscreen enabled)
13 CLOCK_TYPE = "default"
14 KivyConfig.set('kivy', 'kivy_clock', CLOCK_TYPE)
15 KivyConfig.set('graphics', 'maxfps', 250)
16 KivyConfig.write()
17
18 class GranuScreenManager(ScreenManager):
19     pass
20
21 class MainApp(App):
22     def build(self):
23         sm = GranuScreenManager(transition=FadeTransition(duration=0.1))
24         return sm
25
26 if __name__ == "__main__":
27     config.load() # Load our own app preferences
28     MainApp().run()
29
```



```
1 <MainScreenInfoBox@Label>:
2     font_size: 25
3     color: 0,0,0,1
4     halign: 'center'
5     valign: 'center'
6     canvas.before:
7         Color:
8             rgba: .74,.74,.74,1
9         Rectangle:
10             pos: self.pos
11             size: self.size
12
13 <MainScreen>:
14     name: 'main_screen'
15     GranuContainer:
16         GranuSideArea:
17             GranuSideButton:
18                 text: 'Settings'
19                 on_release:
20                     root.move_to('settings_screen') # Move to settings screen
21             GranuSideButton:
22                 text: 'Testing'
23                 on_release:
24                     root.move_to('testing_screen') # Move to testing screen
25             GranuSideButton:
26                 text: 'Live Feed'
27                 on_release:
28                     root.move_to('live_feed_screen') # Move to liveFeed screen
29             GranuSideButton:
30                 text: 'Exit'
31                 on_release:
32                     root.move_to('exit_screen') # Move to exit screen
33         GranuContent:
34             GranuTitle:
35                 text: 'Main Menu'
36             Label:
37                 id: device_name
38                 font_size: 40
39                 text: 'Stalk Strength\nDevice 2.0'
40                 halign: 'center'
41             Label:
42                 id: warning_text
43                 color: (0.8, 0, 0, 1)
44                 size_hint_max_y: 15
45                 font_size: 15
46             GridLayout:
47                 cols: 2
48                 spacing: 10
49                 MainScreenInfoBox:
50                     id: temperature
51                     text: 'Temperature: ' + root.temperature + u'\N{DEGREE SIGN}' + 'F'
52                 MainScreenInfoBox:
53                     id: humidity
54                     text: 'Humidity: ' + root.humidity + "%"
55                 MainScreenInfoBox:
56                     id: location
57                     text: 'Location: ' + root.location
58                 MainScreenInfoBox:
59                     id: time
60                     text: 'Time: ' + root.time
61
```

```
1  """
2  The main screen contains four buttons for navigation:
3  Settings, Testing, Live Feed, and Exit
4
5  It also shows environment data: Temperature, Humidity, Location, and Time.
6  """
7
8  from kivy.lang import Builder
9  from kivy.properties import StringProperty
10 from kivy.clock import Clock
11 from view.BaseScreen import BaseScreen
12 import datetime
13
14 from Sensor import Sensor
15
16 Builder.load_file('view/screens/main/MainScreen.kv')
17
18 INTERVAL = .004
19
20 class MainScreen(BaseScreen):
21     temperature = StringProperty("0")
22     humidity = StringProperty("0")
23     location = StringProperty("0.00,0.00")
24     time = StringProperty("0")
25     def on_pre_enter(self):
26         self.test_time = 0
27         self.event = Clock.schedule_interval(self.update_values, INTERVAL)
28         self.sensor_man = Sensor()
29         if self.sensor_man.REAL_DATA is False:
30             self.ids['warning_text'].text = 'WARNING: Using fake data. Check console for
... stack trace.'
31
32     def update_values(self, obj):
33         self.time = datetime.datetime.now().strftime("%I:%M:%S %p")
34
35     def on_leave(self):
36         self.event.cancel()
37
```

```
1 from connections import *
2
3 def motor_test():
4     print("\n *****Beginning Motor Test***** \n")
5     duration = 1
6     freq = 50
7
8     for motor, in1, in2, pwm in zip(MOTORS, IN1, IN2, PWM):
9         p = GPIO.PWM(pwm, freq)
10        p.start(0)
11
12        # Drive the motor clockwise
13        print("Driving Motor {} clockwise for {} seconds".format(motor, duration))
14        GPIO.output(in1, GPIO.HIGH)
15        GPIO.output(in2, GPIO.LOW)
16        p.ChangeDutyCycle(50)
17        sleep(duration)
18
19        # Drive the motor counterclockwise
20        print("Driving Motor {} counterclockwise for {} seconds".format(motor, duration))
21        GPIO.output(in1, GPIO.LOW)
22        GPIO.output(in2, GPIO.HIGH)
23        p.ChangeDutyCycle(100)
24        sleep(duration)
25
26        # Reset all the GPIO pins by setting them to LOW
27        GPIO.output(in1, GPIO.LOW)
28        GPIO.output(in2, GPIO.LOW)
29        p.stop()
30
31    print("\n *****Ending Motor Test***** \n")
32
33 if __name__ == "__main__":
34     motor_test()
35
```

```
1 <NewNoteScreen>:
2     name: 'new_note_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Save'
7                 on_release:
8                     if root.save(): root.back()
9             GranuSideButton:
10                text: 'Cancel'
11                on_release:
12                    root.back()
13     GranuContent:
14         GranuTitle:
15             text: 'New Note'
16         GridLayout:
17             rows: 1
18             cols: 1
19             StrInput:
20                 id: note
21                 font_size: 30
22                 multiline: False
23                 size_hint_max_y: 30+15
24
```

```
1  """
2  An input text box that, when selected, allows the user to type in a new note via a touch
3  screen keyboard that will pop up. The input text box will iniinputally be empty.
4  """
5
6  from kivy.lang import Builder
7
8  import configurator as config
9  from view.BaseScreen import BaseScreen
10 from view.input.StrInput import StrInput
11
12 Builder.load_file('view/screens/settings/NewNoteScreen.kv')
13
14 class NewNoteScreen(BaseScreen):
15     def on_pre_enter(self):
16         input = self.ids['note']
17         input.text = ''
18
19     def on_enter(self):
20         """Once the Screen loads, focus the Texinputnput"""
21         input = self.ids['note']
22         input.focus = True
23
24     def save(self):
25         notes = config.get('notes', {
26             "pretest": [],
27             "posttest": [],
28             "bank": []
29         })
30         input = self.ids['note']
31
32         note = input.text
33         valid = input.validate()
34         exists = (note in notes['pretest']) or (note in notes['posttest']) \
35             or (note in notes['bank'])
36
37         if valid and not exists:
38             notes['bank'].append(input.text)
39             config.set('notes', notes)
40             return True
41         else:
42             input.show_invalid()
43             input.focus = True
44             return False
45
```

```
1 <Note>
2     canvas.before:
3         Color:
4             rgba: (.0, 0.9, .1, .3) if self.selected else (0, 0, 0, 1)
5         Rectangle:
6             pos: self.pos
7             size: self.size
8
9 <NoteList>
10     viewclass: 'Note'
11     SelectableRecycleBoxLayout:
12
13 <ListTitle@Label>
14     size_hint_y: None
15     height: self.font_size
16     font_size: 15
17
18 <MoveButton@Button>
19     size_hint_y: None
20     height: self.font_size + 30
21     font_size: 15
22
23 <NoteScreen>:
24     name: 'note_screen'
25     GranuContainer:
26         GranuSideArea:
27             id: note_buttons
28         GranuContent:
29             GranuTitle:
30                 text: 'Notes'
31             BoxLayout:
32                 orientation: 'horizontal'
33                 size_hint_y: None
34                 height: self.minimum_height
35                 ListTitle:
36                     text: 'Pre-test Notes'
37                 ListTitle:
38                     text: 'Post-test Notes'
39                 ListTitle:
40                     text: 'Note Bank'
41             BoxLayout:
42                 orientation: 'horizontal'
43                 NoteList:
44                     id: pretest
45                     on_interaction:
46                         posttest.clear_selection()
47                         bank.clear_selection()
48                         root.test_buttons()
49                     on_deselect_all:
50                         root.default_buttons()
51                 NoteList:
52                     id: posttest
53                     on_interaction:
54                         pretest.clear_selection()
55                         bank.clear_selection()
56                         root.test_buttons()
57                     on_deselect_all:
58                         root.default_buttons()
59                 NoteList:
60                     canvas.after:
61                         Color:
62                             rgba: 0.5, 0.5, 0.5, 1
63                         Line:
64                             width: 1
65                             rectangle: self.x, self.y, 1, self.height
66                     id: bank
67                     on_interaction:
68                         pretest.clear_selection()
```

```
69         posttest.clear_selection()
70         root.bank_buttons()
71     on_deselect_all:
72         root.default_buttons()
73     BoxLayout:
74         orientation: 'horizontal'
75         size_hint_y: None
76         height: self.minimum_height
77     MoveButton:
78         text: 'Move Here'
79         on_release:
80             root.move_to_pretest()
81             root.default_buttons()
82     MoveButton:
83         text: 'Move Here'
84         on_release:
85             root.move_to_posttest()
86             root.default_buttons()
87     MoveButton:
88         text: 'Move Here'
89         on_release:
90             root.move_to_bank()
91             root.default_buttons()
92     AnchorLayout:
93         anchor_x: 'right'
94         anchor_y: 'center'
95         size_hint_y: None
96         height: clear_button.height - 6
97     Button:
98         id: clear_button
99         text: 'Clear Selection'
100        size_hint_y: None
101        height: self.font_size + 40
102        size_hint_x: 0.25
103        on_release:
104            pretest.clear_selection()
105            posttest.clear_selection()
106            bank.clear_selection()
107            root.default_buttons()
108
```

```
1 from kivy.lang import Builder
2 from kivy.properties import ListProperty
3
4 import configurator as config
5 from view.BaseScreen import BaseScreen
6 from view.SelectableList import SelectableList, SelectableListBehavior,
... SelectableRecycleBoxLayout
7 from view.elements import *
8
9 Builder.load_file('view/screens/settings/NoteScreen.kv')
10
11 class Note(SelectableListBehavior, Label):
12     pass
13
14 class NoteList(SelectableList):
15     def update(self, k, val):
16         self.data = [{'text': str(x)} for x in self.list_data]
17
18 class NoteScreen(BaseScreen):
19     '''Manages the Notes.
20
21     Be careful not to make a shallow copy of list_data for any SelectableList'''
22
23     # Setup!
24
25     def __init__(self, **kwargs):
26         '''Creates Kivy Buttons to be able to dynamically change the sidebar actions
27         based on interaction with the lists.'''
28         super(BaseScreen, self).__init__(**kwargs)
29
30         self.save_button = GranuSideButton(text = 'Save')
31         self.save_button.bind(on_release = self.save)
32         self.new_button = GranuSideButton(text = 'New')
33         self.new_button.bind(on_release = self.new)
34         self.remove_button = GranuSideButton(text = 'Remove')
35         self.remove_button.bind(on_release = self.remove)
36         self.delete_button = GranuSideButton(text = 'Delete')
37         self.delete_button.bind(on_release = self.delete)
38
39     def on_pre_enter(self):
40         """Before the Screen loads, read the configuration file to get the current
41         list of notes. Show the default buttons."""
42
43         # Get notes from config file
44         notes = config.get('notes', {
45             "pretest": [],
46             "posttest": [],
47             "bank": []
48         })
49         # Set the data
50         self.ids['pretest'].list_data = notes["pretest"]
51         self.ids['posttest'].list_data = notes["posttest"]
52         self.ids['bank'].list_data = notes["bank"]
53
54         # Add Buttons
55         self.default_buttons()
56
57     def _save_config(self):
58         '''Save the notes to the configuration file.'''
59         config.set('notes', {
60             "pretest": self.ids['pretest'].list_data,
61             "posttest": self.ids['posttest'].list_data,
62             "bank": self.ids['bank'].list_data
63         })
64
65     # Button Changes
66
67
```



```
68     def default_buttons(self):
69         buttons = self.ids['note_buttons']
70         buttons.clear_widgets()
71         buttons.add_widget(self.save_button)
72         buttons.add_widget(self.new_button)
73         buttons.add_widget(Widget())
74         buttons.add_widget(Widget())
75
76     def test_buttons(self):
77         buttons = self.ids['note_buttons']
78         buttons.clear_widgets()
79         buttons.add_widget(self.save_button)
80         buttons.add_widget(self.new_button)
81         buttons.add_widget(Widget())
82         buttons.add_widget(self.remove_button)
83
84     def bank_buttons(self):
85         buttons = self.ids['note_buttons']
86         buttons.clear_widgets()
87         buttons.add_widget(self.save_button)
88         buttons.add_widget(self.new_button)
89         buttons.add_widget(Widget())
90         buttons.add_widget(self.delete_button)
91
92     # Button Actions
93
94     def save(self, obj):
95         self._save_config()
96         super(NoteScreen, self).back()
97
98     def new(self, obj):
99         self._save_config()
100        self.move_to('new_note_screen')
101
102     def remove(self, obj):
103         '''Move selected notes to the Note Bank'''
104         notes = self.ids['pretest'].remove_selected() \
105             + self.ids['posttest'].remove_selected()
106         self.ids['bank'].add_items(notes)
107         self.default_buttons()
108
109     def delete(self, obj):
110         self.ids['bank'].remove_selected()
111         self.default_buttons()
112
113     # Content Buttons
114
115     def move_to_pretest(self):
116         notes = self.ids['pretest'].remove_selected() \
117             + self.ids['posttest'].remove_selected() \
118             + self.ids['bank'].remove_selected()
119         self.ids['pretest'].add_items(notes)
120
121     def move_to_posttest(self):
122         notes = self.ids['pretest'].remove_selected() \
123             + self.ids['posttest'].remove_selected() \
124             + self.ids['bank'].remove_selected()
125         self.ids['posttest'].add_items(notes)
126
127     def move_to_bank(self):
128         notes = self.ids['pretest'].remove_selected() \
129             + self.ids['posttest'].remove_selected() \
130             + self.ids['bank'].remove_selected()
131         self.ids['bank'].add_items(notes)
132
```

```
1 {
2   "title" : "Numeric",
3   "description" : "A numeric keypad",
4   "cols" : 3,
5   "rows" : 4,
6   "normal_1" : [
7     ["7", "7", "7", 1],
8     ["8", "8", "8", 1],
9     ["9", "9", "9", 1]],
10  "normal_2" : [
11    ["4", "4", "4", 1],
12    ["5", "5", "5", 1],
13    ["6", "6", "6", 1]],
14  "normal_3" : [
15    ["1", "1", "1", 1],
16    ["2", "2", "2", 1],
17    ["3", "3", "3", 1]],
18  "normal_4" : [
19    ["0", "0", "0", 1],
20    [".", ".", ".", 1],
21    ["\u232b", null, "backspace", 1]],
22  "shift_1" : [
23    ["7", "7", "7", 1],
24    ["8", "8", "8", 1],
25    ["9", "9", "9", 1]],
26  "shift_2" : [
27    ["4", "4", "4", 1],
28    ["5", "5", "5", 1],
29    ["6", "6", "6", 1]],
30  "shift_3" : [
31    ["1", "1", "1", 1],
32    ["2", "2", "2", 1],
33    ["3", "3", "3", 1]],
34  "shift_4" : [
35    ["0", "0", "0", 1],
36    [".", ".", ".", 1],
37    ["\u232b", null, "backspace", 1]]
38 }
39
```

```
1 <OperatorScreen>:
2     name: 'operator_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Save'
7                 on_release:
8                     if root.save(): root.back()
9             GranuSideButton:
10                text: 'Cancel'
11                on_release:
12                    root.back()
13            GranuNone:
14            GranuNone:
15        GranuContent:
16            GranuTitle:
17                text: 'Operator'
18            GridLayout:
19                rows: 1
20                cols: 1
21            StrInput:
22                id: operator
23                font_size: 30
24                multiline: False
25                size_hint_max_y: 30+15
26
```

```
1  """
2  An input text box that, when selected, allows the user to type in the current Operator
3  setting via a touch screen keyboard that will pop up. The value in the input text box
4  when you first visit this view is whatever value for the Operator setting is currently
5  stored in our settings file .
6  """
7
8  from kivy.lang import Builder
9
10 import configurator as config
11 from view.BaseScreen import BaseScreen
12 from view.input.StrInput import StrInput
13
14 Builder.load_file('view/screens/settings/OperatorScreen.kv')
15
16 class OperatorScreen(BaseScreen):
17     def on_pre_enter(self):
18         """Before the Screen loads, read the configuration file to get the current
19         operator and set the TextInput text."""
20         input = self.ids['operator']
21         input.text = str(config.get('operator', "Default User"))
22         input.validate()
23
24     def on_enter(self):
25         """Once the Screen loads, focus the TextInput"""
26         input = self.ids['operator']
27         input.focus = True
28         input.select_all()
29
30     def save(self):
31         """Save button was pressed: save the new operator in the configuration file."""
32         input = self.ids['operator']
33         valid = input.validate()
34         if valid:
35             config.set('operator', str(input.text))
36             return True
37         else:
38             input.focus = True
39             return False
40
```

```
1 <PlotScreen>:
2     name: 'plot_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Save'
7                 on_release:
8                     if root.save(): root.back()
9             GranuSideButton:
10                text: 'Cancel'
11                on_release:
12                    root.back()
13     GranuContent:
14         GranuTitle:
15             text: 'Plot Number'
16         StackLayout:
17             padding: 20
18             IntInput:
19                 id: plot_num
20                 font_size: 30
21                 size_hint_max_y: 30+15
22
```

```
1  """
2  An input text box that, when selected, allows the user to type in the current Plot
3  setting via a touch screen number pad that will pop up. The value in the input text box
4  when you first visit this view is whatever value for the Plot setting is currently stored
5  in our settings file.
6  """
7
8  from kivy.lang import Builder
9
10 import configurator as config
11 from view.BaseScreen import BaseScreen
12 from view.input.IntInput import IntInput
13
14 Builder.load_file('view/screens/settings/PlotScreen.kv')
15
16 class PlotScreen(BaseScreen):
17     def on_pre_enter(self):
18         """Before the Screen loads, read the configuration file to get the current
19         plot number."""
20         input = self.ids['plot_num']
21         input.text = str(config.get('plot_num', 1))
22         input.validate()
23
24     def on_enter(self):
25         """Once the Screen loads, focus the TextInput"""
26         input = self.ids['plot_num']
27         input.focus = True
28         input.select_all()
29
30     def save(self):
31         """Save button was pressed: save the new height in the configuration file.
32         Returns True if save was successful. False otherwise."""
33         input = self.ids['plot_num']
34         valid = input.validate()
35         if valid:
36             config.set('plot_num', int(input.text))
37             return True
38         else:
39             input.focus = True
40             return False
41
```

```
1 from .connections import *
2
3 class Pot:
4
5     def __init__(self):
6         self.pot = 0.0
7
8     def get_data(self):
9         try:
10             self.pot = POT_CHAN.value
11             return self.pot
12         except:
13             return self.pot
14
```

# Plant Stalk Measurement Device Software

To run the software, you must have python3 and Kivy installed.

Navigate to the src/ folder in terminal and run:

```
python3 main.py
```

## Compiling the Documentation as PDF

- Download Doxygen and open Doxyfile and run
  - Make sure to output LaTeX files "as an intermediate format for [hyperlinked] PDF"
- Download LaTeX <https://www.latex-project.org/get/>
- Add LaTeX's binaries and scripts to your PATH
- Run the make file in the LaTeX folder to generate a PDF



```
1 <SelectableRecycleBoxLayout>:
2     default_size: None, dp(56)
3     default_size_hint: 1, None
4     size_hint_y: None
5     height: self.minimum_height
6     orientation: 'vertical'
7     multiselect: True
8     touch_multiselect: True
9     keyboard_mode: 'managed'
10
```

```
1 from kivy.lang import Builder
2
3 from kivy.uix.recycleview import RecycleView
4 from kivy.uix.recycleview.views import RecycleDataViewBehavior
5 from kivy.uix.label import Label
6 from kivy.properties import BooleanProperty
7 from kivy.uix.recycleboxlayout import RecycleBoxLayout
8 from kivy.uix.behaviors import FocusBehavior
9 from kivy.uix.recycleview.layout import LayoutSelectionBehavior
10 from kivy.uix.behaviors import ButtonBehavior
11
12 from kivy.properties import ListProperty
13
14 Builder.load_file('view/SelectableList.kv')
15
16 class SelectableListBehavior(RecycleDataViewBehavior):
17     '''Add selection support to a Label.'''
18     index = None
19     selected = BooleanProperty(False)
20     selectable = BooleanProperty(True)
21
22     def refresh_view_attrs(self, rv, index, data):
23         '''Catch and handle data changes.'''
24         self.index = index
25         return super(SelectableListBehavior, self).refresh_view_attrs(rv, index, data)
26
27     def on_touch_down(self, touch):
28         '''Select this item on touch down.'''
29         if super(SelectableListBehavior, self).on_touch_down(touch):
30             return True
31         if self.collide_point(*touch.pos) and self.selectable:
32             ret = self.parent.select_with_touch(self.index, touch)
33             self.parent.parent._interact() # SelectableList was interacted with
34             return ret
35
36     def apply_selection(self, rv, index, is_selected):
37         '''Respond to the selection of items in the view.'''
38         self.selected = is_selected
39
40 class SelectableRecycleBoxLayout(FocusBehavior, LayoutSelectionBehavior,
41                                 RecycleBoxLayout):
42     ''' Adds selection and focus behaviour to the view. '''
43
44 class SelectableList(RecycleView):
45     '''A selectable list widget, that allows you to modify its data.
46
47     Known issues: Changing data in a RecycleView causes "sys.excepthook" errors at close:
48     https://github.com/kivy/kivy/issues/5986'''
49
50     __events__ = ('on_interaction', 'on_deselect_all', ) # Add an event that can be defined
51 in a kv file
52     list_data = ListProperty() # List of strings to be shown in the list
53
54     def __init__(self, **kwargs):
55         '''Update the SelectableList's RecycleView data whenever list_data changes.'''
56         super(SelectableList, self).__init__(**kwargs)
57         self.bind(list_data=self.update)
58
59     # Abstract!
60     def update(self, k, val):
61         '''Uses list_data to generate SelectableListWidgets.'''
62
63     def _interact(self):
64         '''Called by a list item when it is touched. Dispatches an on_interaction
65         event.'''
66         self.dispatch('on_interaction') # Call the Kivy event on_interaction
67         if len(self.layout_manager.selected_nodes) == 0:
68             self.dispatch('on_deselect_all')
```

```
68
69     def clear_selection(self):
70         '''Clears the selection in the SelectableList'''
71         lm = self.layout_manager
72         lm.clear_selection()
73
74     def get_selected(self):
75         '''Returns the items currently selected.'''
76         lm = self.layout_manager
77         sels = []
78         for i in lm.selected_nodes:
79             sels.append(self.list_data[i])
80         return sels
81
82     def remove_selected(self):
83         '''Removes the selection in the SelectableList. Returns the items removed.'''
84         lm = self.layout_manager
85         removed = []
86         for i in lm.selected_nodes:
87             removed.append(self.list_data[i])
88         lm.clear_selection()
89         for item in removed:
90             self.list_data.remove(item)
91         return removed
92
93     def add_items(self, list):
94         '''Add items to the list.'''
95         self.list_data = self.list_data + list
96
97     def on_interaction(self, *largs):
98         '''A Kivy event: can be defined in a kv file. Called when the SelectableList
99         gains focus.'''
100         pass
101
102     def on_deselect_all(self, *largs):
103         '''A Kivy event: can be defined in a kv file. Called when the user deselects the
104         last selected item in the SelectableList are deselected.'''
105         pass
106
```

```
1 import traceback
2 import sys
3 try:
4     from .SensorReal import Sensor
5 except:
6     print(traceback.print_exc()) # Tell us what happened
7     from .SensorFake import Sensor
8
```

```
1
2
3 class Sensor:
4     def __init__(self):
5         self.REAL_DATA = False
6         self.keys = ["Temperature", "Humidity", "Location", "X Load", "Y Load", "Pot Angle", "IMU
... Angle"]
7         self.sensor_data = {}
8         self.temp_fake = 0
9         self.hum_fake = 0
10        self.loc_fake = 0
11        self.x_fake = 0
12        self.y_fake = 0
13        self.pot_fake = 0
14        self.imu_fake = 0
15
16    def get_header_data(self):
17        self.temp_fake += 1
18        self.hum_fake += 2
19        self.loc_fake += 4
20        self.sensor_data["Temperature"] = self.temp_fake
21        self.sensor_data["Humidity"] = self.hum_fake
22        self.sensor_data["Location"] = self.loc_fake
23        return self.sensor_data
24
25    def get_sensor_data(self):
26        self.x_fake += 8
27        self.y_fake += 16
28        self.pot_fake += 32
29        self.imu_fake += 64
30        self.sensor_data["X Load"] = self.x_fake
31        self.sensor_data["Y Load"] = self.y_fake
32        self.sensor_data["Pot Angle"] = self.pot_fake
33        self.sensor_data["IMU Angle"] = self.imu_fake
34        return self.sensor_data
35
36    def get_all_data(self):
37        self.temp_fake += 1
38        self.hum_fake += 2
39        self.loc_fake += 4
40        self.x_fake += 8
41        self.y_fake += 16
42        self.pot_fake += 32
43        self.imu_fake += 64
44        self.sensor_data["Temperature"] = self.temp_fake
45        self.sensor_data["Humidity"] = self.hum_fake
46        self.sensor_data["Location"] = self.loc_fake
47        self.sensor_data["X Load"] = self.x_fake
48        self.sensor_data["Y Load"] = self.y_fake
49        self.sensor_data["Pot Angle"] = self.pot_fake
50        self.sensor_data["IMU Angle"] = self.imu_fake
51        return self.sensor_data
52
53    def get_sensor_keys(self):
54        return self.keys
55
56
```

```
1 from sensors.Temperature import Temperature
2 from sensors.Humidity import Humidity
3 from sensors.Location import Location
4 from sensors.X_Load import X_Load
5 from sensors.Y_Load import Y_Load
6 from sensors.Pot import Pot
7 from sensors.IMU import IMU
8 import datetime
9 import board
10 import busio
11 import adafruit_ads1x15.ads1015 as ADS
12 from adafruit_ads1x15.analog_in import AnalogIn
13
14 class Sensor:
15
16     def __init__(self):
17         self.REAL_DATA = True
18         self.keys = ["Temperature", "Humidity", "Location", "X Load", "Y Load", "Pot Angle", "IMU
... Angle"]
19         self.temp = Temperature()
20         self.hum = Humidity()
21         self.location = Location()
22         self.x_load = X_Load()
23         self.y_load = Y_Load()
24         self.pot_angle = Pot()
25         self.imu_angle = IMU()
26         self.sensor_data = {}
27         self.temp_fake = 0
28         self.hum_fake = 0
29         self.loc_fake = 0
30         self.x_fake = 0
31         self.y_fake = 0
32         self.pot_fake = 0
33         self.imu_fake = 0
34
35     def get_header_data(self):
36         self.sensor_data["Temperature"] = self.temp.get_data()
37         self.sensor_data["Humidity"] = self.hum.get_data()
38         self.sensor_data["Location"] = self.location.get_data()
39         return self.sensor_data
40
41     def get_sensor_data(self):
42         self.sensor_data["X Load"] = round(self.x_load.get_data(),4)
43         self.sensor_data["Y Load"] = round(self.y_load.get_data(),4)
44         self.sensor_data["Pot Angle"] = round(self.pot_angle.get_data(),3)
45         self.sensor_data["IMU Angle"] = round(self.imu_angle.get_data(),3)
46         return self.sensor_data
47
48     def get_all_data(self):
49         self.sensor_data["Temperature"] = self.temp.get_data()
50         self.sensor_data["Humidity"] = self.hum.get_data()
51         self.sensor_data["Location"] = self.location.get_data()
52         self.sensor_data["X Load"] = round(self.x_load.get_data(),4)
53         self.sensor_data["Y Load"] = round(self.y_load.get_data(),4)
54         self.sensor_data["Pot Angle"] = round(self.pot_angle.get_data(),3)
55         self.sensor_data["IMU Angle"] = round(self.imu_angle.get_data(),3)
56         return self.sensor_data
57
58     def get_sensor_keys(self):
59         return self.keys
60
61 if __name__ == "__main__":
62     sensor = Sensor()
63     print("\n *****Beginning Sensor Test***** \n")
64     print("Sensor Data: ")
65     data_array = sensor.get_sensor_data()
66     for key in sensor.get_sensor_keys():
67         print(key, data_array[key])
```

```
68 print("\n *****Ending Sensor Test***** \n")  
69
```

```
1 <SensorButton>:
2     font_size: 30
3     halign: 'center'
4
5 <SensorsScreen>:
6     name: 'sensors_screen'
7     GranuContainer:
8         GranuSideArea:
9             GranuSideButton:
10                text: 'Back'
11                on_release:
12                    root.back()
13     GranuContent:
14         GranuTitle:
15             text: 'Sensor Calibration'
16         GridLayout:
17             id: sensor_list
18             cols: 3
19             rows: 3
20             spacing: 10
21
```



```
1 from kivy.lang import Builder
2 from kivy.clock import Clock
3 from kivy.uix.button import Button
4 from kivy.properties import ObjectProperty
5
6 from view.BaseScreen import BaseScreen
7 from Sensor import Sensor
8
9 Builder.load_file('view/screens/settings/SensorsScreen.kv')
10
11 class SensorButton(Button):
12     def __init__(self, name, parent_screen, calib_screen, **kwargs):
13         '''Set the sensor name (which is also the key to retrieve calibration settings
14         from the config file. Reference the parent screen for move_to() function.
15         Reference the calibration screen to set it up to calibrate this setting.'''
16         super(Button, self).__init__(**kwargs)
17         self.name = name
18         # Screen References
19         self.parent_screen = parent_screen
20         self.calib_screen = calib_screen
21         # Kivy Properties
22         self.text = name
23
24     def on_release(self):
25         self.calib_screen.set_sensor(self.name)
26         self.parent_screen.move_to('calibrate_screen')
27
28 class SensorsScreen(BaseScreen):
29     def __init__(self, **kwargs):
30         '''Add a button for each sensor.'''
31         super(BaseScreen, self).__init__(**kwargs)
32         self.senseMan = Sensor()
33         def gui_init(dt):
34             '''Called once the Kivy file is parsed. Needed so we can access Kivy IDs.'''
35             calib_screen = self.manager.get_screen('calibrate_screen')
36             for s in self.senseMan.get_sensor_keys():
37                 # Perhaps Location and Time should be accessed in some other way?
38                 if s=='Location' or s=='Time': continue
39                 # Sensor name, parent screen (of button), calibration screen
40                 self.ids['sensor_list'].add_widget(SensorButton(s, self, calib_screen))
41         Clock.schedule_once(gui_init)
42
```

```
1 #:import Path pathlib.Path
2
3 <LoadDialog>:
4     title: 'Load Settings'
5     title_align: 'center'
6     size_hint: (0.9, 0.9)
7     BoxLayout:
8         orientation: "vertical"
9         FileChooserIconView:
10             id: filechooser
11             path: str(Path.home())
12         BoxLayout:
13             size_hint_y: None
14             height: 30
15             Button:
16                 text: "Cancel"
17                 on_release: root.cancel()
18             Button:
19                 text: "Load"
20                 on_release: root.load(filechooser.path, filechooser.selection[0])
21
22 <SaveDialog>:
23     title: 'Save Settings'
24     title_align: 'center'
25     size_hint: (0.9, 0.9)
26     BoxLayout:
27         orientation: "vertical"
28         BoxLayout:
29             orientation: "horizontal"
30             size_hint_y: None
31             height: filename.height
32             Label:
33                 text: 'Filename:'
34                 size_hint_x: None
35             StrInput:
36                 id: filename
37                 text: 'settings.json'
38                 size_hint_y: None
39                 height: 30
40                 multiline: False
41         FileChooserIconView:
42             id: filechooser
43             path: str(Path.home())
44             on_selection: filename.text = self.selection and self.selection[0] or ''
45         BoxLayout:
46             size_hint_y: None
47             height: 30
48             Button:
49                 text: "Cancel"
50                 on_release: root.cancel()
51             Button:
52                 text: "Save"
53                 on_release: root.save(filechooser.path, filename.text)
54
55 <SettingsButton@Button>:
56     font_size: 30
57     halign: 'center'
58
59 <SettingsScreen>:
60     name: 'settings_screen'
61     GranuContainer:
62         GranuSideArea:
63             GranuNone:
64             GranuNone:
65             GranuNone:
66             GranuSideButton:
67                 text: 'Back'
68                 on_release:
```

```
69         root.manager.transition.direction = 'right'
70         root.move_to('main_screen') # Move to main screen
71     GranuContent:
72         GranuTitle:
73             text: 'Settings'
74         GridLayout:
75             cols: 3
76             rows: 3
77             spacing: 10
78             row_default_height: self.height/4
79             SettingsButton:
80                 text: 'Height'
81                 on_release:
82                     root.move_to('height_screen')
83             SettingsButton:
84                 text: 'Plot'
85                 on_release:
86                     root.manager.transition.direction = 'left'
87                     root.move_to('plot_screen')
88             SettingsButton:
89                 text: 'Notes'
90                 on_release:
91                     root.manager.transition.direction = 'left'
92                     root.move_to('note_screen')
93             SettingsButton:
94                 text: 'Sensors'
95                 on_release:
96                     root.manager.transition.direction = 'left'
97                     root.move_to('sensors_screen')
98             SettingsButton:
99                 text: 'Operator'
100                 on_release:
101                     root.manager.transition.direction = 'left'
102                     root.move_to('operator_screen')
103     StackLayout:
104         orientation: 'rl-bt'
105         size_hint_y: None
106         spacing: 5
107         height: load_button.height - 6
108         Button:
109             id: load_button
110             text: 'Load File'
111             size_hint_y: None
112             height: self.font_size + 40
113             size_hint_x: 0.25
114             on_release: root.show_load()
115         Button:
116             text: 'Save File'
117             size_hint_y: None
118             height: self.font_size + 40
119             size_hint_x: 0.25
120             on_release: root.show_save()
121
```

```
1  """
2  From the settings screen you can navigate to these options: Height, Plot, Operator,
3  Folder, Notes
4  """
5
6  import os
7
8  from kivy.lang import Builder
9  from kivy.uix.floatlayout import FloatLayout
10 from kivy.properties import ObjectProperty
11 from kivy.properties import StringProperty
12 from kivy.uix.popup import Popup
13
14 import configurator as config
15 from view.BaseScreen import BaseScreen
16
17 Builder.load_file('view/screens/main/SettingsScreen.kv')
18
19 class LoadDialog(Popup):
20     '''A dialog to load a file. The load and cancel properties point to the
21     functions called when the load or cancel buttons are pressed.'''
22     load = ObjectProperty(None)
23     cancel = ObjectProperty(None)
24
25
26 class SaveDialog(Popup):
27     '''A dialog to save a file. The save and cancel properties point to the
28     functions called when the save or cancel buttons are pressed.'''
29     save = ObjectProperty(None)
30     cancel = ObjectProperty(None)
31
32 class SettingsScreen(BaseScreen):
33     def dismiss_popup(self):
34         self._popup.dismiss()
35
36     def show_load(self):
37         self._popup = LoadDialog(load=self.load, cancel=self.dismiss_popup)
38         self._popup.open()
39
40     def show_save(self):
41         self._popup = SaveDialog(save=self.save, cancel=self.dismiss_popup)
42         self._popup.open()
43
44     def load(self, path, filename):
45         config.load_from(os.path.join(path, filename))
46         self.dismiss_popup()
47
48     def save(self, path, filename):
49         config.save_as(os.path.join(path, filename))
50         self.dismiss_popup()
51
```

```
1 <SingleSelectableRecycleBoxLayout>:
2     default_size: None, dp(56)
3     default_size_hint: 1, None
4     size_hint_y: None
5     height: self.minimum_height
6     orientation: 'vertical'
7
8     keyboard_mode: 'managed'
9
```

```
1 from kivy.lang import Builder
2
3 from kivy.uix.recycleview import RecycleView
4 from kivy.uix.recycleview.views import RecycleDataViewBehavior
5 from kivy.uix.label import Label
6 from kivy.properties import BooleanProperty
7 from kivy.uix.recycleboxlayout import RecycleBoxLayout
8 from kivy.uix.behaviors import FocusBehavior
9 from kivy.uix.recycleview.layout import LayoutSelectionBehavior
10 from kivy.uix.behaviors import ButtonBehavior
11
12 from kivy.properties import ListProperty
13
14 Builder.load_file('view/SingleSelectableList.kv')
15
16 class SingleSelectableListBehavior(RecycleDataViewBehavior):
17     '''Add selection support to a Label.'''
18     index = None
19     selected = BooleanProperty(False)
20     singleSelectable = BooleanProperty(True)
21
22     def refresh_view_attrs(self, rv, index, data):
23         '''Catch and handle data changes.'''
24         self.index = index
25         return super(SingleSelectableListBehavior, self).refresh_view_attrs(rv, index, data)
26
27     def on_touch_down(self, touch):
28         '''Select this item on touch down.'''
29         if super(SingleSelectableListBehavior, self).on_touch_down(touch):
30             return True
31         if self.collide_point(*touch.pos) and self.selectable:
32             ret = self.parent.select_with_touch(self.index, touch)
33             self.parent.parent._interact() # SelectableList was interacted with
34             return ret
35
36     def apply_selection(self, rv, index, is_selected):
37         '''Respond to the selection of items in the view.'''
38         self.selected = is_selected
39
40 class SingleSelectableRecycleBoxLayout(FocusBehavior, LayoutSelectionBehavior,
41                                       RecycleBoxLayout):
42     ''' Adds selection and focus behaviour to the view. '''
43
44 class SingleSelectableList(RecycleView):
45     '''A selectable list widget, that allows you to modify its data.
46
47     Known issues: Changing data in a RecycleView causes "sys.excepthook" errors at close:
48     https://github.com/kivy/kivy/issues/5986'''
49
50     __events__ = ('on_interaction', 'on_deselect_all', ) # Add an event that can be defined
51 in a kv file
52     list_data = ListProperty() # List of strings to be shown in the list
53
54     def __init__(self, **kwargs):
55         '''Update the SelectableList's RecycleView data whenever list_data changes.'''
56         super(SingleSelectableList, self).__init__(**kwargs)
57         self.bind(list_data=self.update)
58
59     # Abstract!
60     def update(self, k, val):
61         '''Uses list_data to generate SelectableListWidgets.'''
62
63     def _interact(self):
64         '''Called by a list item when it is touched. Dispatches an on_interaction
65         event.'''
66         self.dispatch('on_interaction') # Call the Kivy event on_interaction
67         if len(self.layout_manager.selected_nodes) == 0:
68             self.dispatch('on_deselect_all')
```

```
68
69     def clear_selection(self):
70         '''Clears the selection in the SelectableList'''
71         lm = self.layout_manager
72         lm.clear_selection()
73
74     def get_selected(self):
75         '''Returns the items currently selected.'''
76         lm = self.layout_manager
77         sels = []
78         for i in lm.selected_nodes:
79             sels.append(self.list_data[i])
80         return sels
81
82     def remove_selected(self):
83         '''Removes the selection in the SelectableList. Returns the items removed.'''
84         lm = self.layout_manager
85         removed = []
86         for i in lm.selected_nodes:
87             removed.append(self.list_data[i])
88         lm.clear_selection()
89         for item in removed:
90             self.list_data.remove(item)
91         return removed
92
93     def add_items(self, list):
94         '''Add items to the list.'''
95         self.list_data = self.list_data + list
96
97     def on_interaction(self, *largs):
98         '''A Kivy event: can be defined in a kv file. Called when the SelectableList
99         gains focus.'''
100         pass
101
102     def on_deselect_all(self, *largs):
103         '''A Kivy event: can be defined in a kv file. Called when the user deselects the
104         last selected item in the SelectableList are deselected.'''
105         pass
106
```

```
1 <StaticRecycleBoxLayout>:
2     default_size: None, dp(30)
3     default_size_hint: 1, None
4     size_hint_y: None
5     height: self.minimum_height
6     orientation: 'vertical'
7     multiselect: False
8     touch_multiselect: False
9     keyboard_mode: 'managed'
10
11 <StaticLabel>:
12
13 <StaticList>:
14     viewclass: 'StaticLabel'
15     StaticRecycleBoxLayout:
16
```



```
1 from kivy.lang import Builder
2 from kivy.uix.recycleview import RecycleView
3 from kivy.uix.recycleview.views import RecycleDataViewBehavior
4 from kivy.uix.label import Label
5 from kivy.uix.recycleboxlayout import RecycleBoxLayout
6 from kivy.properties import ListProperty
7
8 Builder.load_file('view/StaticList.kv')
9
10 class StaticRecycleBoxLayout(RecycleBoxLayout):
11     pass
12
13 class StaticLabel(RecycleDataViewBehavior, Label):
14     '''Refresh Labels when list is changed.'''
15     index = None
16
17     def refresh_view_attrs(self, rv, index, data):
18         '''Catch and handle data changes.'''
19         self.index = index
20         return super(StaticLabel, self).refresh_view_attrs(rv, index, data)
21
22 class StaticList(RecycleView):
23     '''A static list widget.
24
25     Known issues: Changing data in a RecycleView causes "sys.excepthook" errors at close:
26     https://github.com/kivy/kivy/issues/5986'''
27
28     list_data = ListProperty() # List of strings to be shown in the list
29
30     def __init__(self, **kwargs):
31         '''Update the StaticList's RecycleView data whenever list_data changes.'''
32         super(StaticList, self).__init__(**kwargs)
33         self.bind(list_data=self._update)
34
35     def _update(self, k, val):
36         '''Uses list_data to generate StaticLabels.'''
37         self.data = [{'text': str(x)} for x in self.list_data]
38
```

```
1 from kivy.clock import Clock
2 from kivy.uix.textinput import TextInput
3
4 import view.keyboard_man as km
5
6 class StrInput(TextInput):
7     def validate(self):
8         '''Make sure the string is not empty or all whitespace.'''
9         notempty = len(self.text) > 0 # String is not empty or all whitespace
10        notspace = not self.text.isspace()
11        test = notempty and notspace
12        if test:
13            self.show_valid()
14        else:
15            self.show_invalid()
16        return test
17
18    def show_invalid(self):
19        '''Colors the textinput red.'''
20        self.background_color = (1, .7, .7, 1)
21
22    def show_valid(self):
23        '''Colors the textinput white.'''
24        self.background_color = (1, 1, 1, 1)
25
26    def on_text_validate(self):
27        '''Called when enter is pressed.'''
28        self.validate()
29        Clock.schedule_once(self.focus_and_select)
30
31    def on_focus(self, instance, value):
32        '''When the StrInput is focused, show a qwerty keyboard.'''
33        if value:
34            km.show_keyboard(self, 'text')
35
36    def focus_and_select(self, *args):
37        '''Focus the TextInput and select all of its text.'''
38        self.focus = True
39        self.select_all()
40
```

```
1 from connections import *
2
3 def temp_test():
4     print("\n *****Beginning TEMP Test***** \n")
5
6     for i in range(3):
7         print("Temperature: ", am.temperature)
8         print("Humidity: ", am.relative_humidity)
9         time.sleep(2)
10
11     print("\n *****Ending TEMP Test***** \n")
12
13 if __name__ == "__main__":
14     temp_test()
15
```

```
1 from .connections import *
2
3 class Temperature:
4
5     def __init__(self):
6         self.temp = 0.0
7
8     def get_data(self):
9         try:
10             self.temp = am.temperature
11             return self.temp
12         except:
13             return self.temp
14
```

```
1 from tests.gpio_test import *
2 from tests.adc_test import *
3 from tests.gps_test import *
4 from tests.motor_test import *
5 from tests.temp_test import *
6 from tests.accl_test import *
7 import RPi.GPIO as GPIO
8
9 if __name__ == "__main__":
10     temp_test()
11     accl_test()
12     gpio_test()
13     adc_test()
14     gps_test()
15     motor_test()
16     GPIO.cleanup()
17
```

```
1 #:import MeshLinePlot kivy.garden.graph.MeshLinePlot
2
3 <TestingResultsInfoBox@Label>:
4     font_size: 25
5     color: 0,0,0,1
6     halign: 'center'
7     valign: 'center'
8     canvas.before:
9         Color:
10             rgba: .74,.74,.74,1
11         Rectangle:
12             pos: self.pos
13             size: self.size
14
15 <TestingResultsScreen>:
16     name: 'testing_results_screen'
17     GranuContainer:
18         GranuSideArea:
19             GranuSideButton:
20                 text: 'Update\nNotes'
21                 on_release:
22                     root.move_to('note_screen') # Move to note_screen height screen
23             GranuSideButton:
24                 text: 'Break\nHeight'
25                 on_release:
26                     root.move_to('break_height_screen') # Move to break height screen
27             GranuSideButton:
28                 text: 'Reject'
29                 on_release:
30                     root.move_to('main_screen') # Move to main screen
31             GranuSideButton:
32                 text: 'Save'
33                 on_release:
34                     root.save_test()
35                     root.move_to('main_screen') # Move to main screen
36     GranuContent:
37         GranuTitle:
38             text: 'Testing Results'
39         Graph:
40             id: graph_test
41             plot: MeshLinePlot
42             background_color: .5, .5, .5, 1
43             xlabel: 'Time(s)'
44             ylabel: 'X_Load(adc)'
45             x_ticks_major: root.x_major
46             y_ticks_major: root.y_major
47             y_grid_label: True
48             x_grid_label: True
49             padding: 5
50             x_grid: True
51             y_grid: True
52             xmin: 0
53             ymin: 0
54             xmax: root.x_max
55             ymax: root.y_max
56
```

```
1  """
2  Shows all data: Temperature, Humidity, Location, Time, and all Sensor data
3  """
4
5  from kivy.lang import Builder
6  from kivy.properties import NumericProperty
7  from kivy.properties import StringProperty
8  from kivy.properties import ListProperty
9  from kivy.clock import Clock
10 from Sensor import Sensor
11 import datetime
12 import time
13 import math
14 from TestSingleton import TestSingleton
15
16 from view.BaseScreen import BaseScreen
17 from view.StaticList import StaticList
18 from view.elements import *
19 import configurator as config
20 import csv
21
22 from kivy.garden.graph import Graph, MeshLinePlot
23
24 Builder.load_file('view/screens/main/testing/TestingResultsScreen.kv')
25
26 ONE_SEC = 1
27
28
29 class TestingResultsScreen(BaseScreen):
30     x_max = NumericProperty(1)
31     y_max = NumericProperty(1)
32     x_major = NumericProperty(1)
33     y_major = NumericProperty(1)
34     datasets = []
35
36
37     def find_max_x_load(self):
38         max = 0
39         for dataset in self.datasets:
40             if(dataset.x_load > max):
41                 max = dataset.x_load
42         return max
43
44     def on_enter(self):
45         self.graph = self.ids['graph_test']
46         self.plot = MeshLinePlot(color=[1, 1, 1, 1])
47         ts = TestSingleton()
48         self.datasets = ts.get_datasets()
49         last_index = len(self.datasets) - 1
50
51         self.x_max = math.ceil(self.datasets[last_index].timestamp / 5) * 5
52         #self.y_max = math.ceil(self.find_max_x_load() / 10000) * 10000
53         self.y_max = 2000
54         self.x_major = int(self.x_max/5)
55         self.y_major = int(self.y_max/5)
56
57
58         self.plot.points = [(self.datasets[i].timestamp, self.datasets[i].pot_angle) for i
... in range(0, len(self.datasets))]
59         #for i in range(0, len(self.datasets)):
60         #    print("Time:", self.datasets[i].timestamp, " -- X_Load:",
... self.datasets[i].x_load)
61
62         self.graph.add_plot(self.plot)
63
64
65     def save_test(self):
66         ts = TestSingleton()
```

```

67     self.datasets = ts.get_datasets()
68     ts.set_break_height(str(config.get('break_height', 0)))
69
70     #Prepare the notes
71     notes = config.get('notes', {
72         "pretest": [],
73         "posttest": [],
74         "bank": []
75     })
76     pre_notes = notes["pretest"]
77     post_notes = notes["posttest"]
78     while(len(pre_notes) < 5):
79         pre_notes.append('')
80     while(len(post_notes) < 5):
81         post_notes.append('')
82     dt = datetime.datetime.now()
83     filename = 'Tests/' + dt.strftime('%Y_%m_%d_%H_%M_%S') + '.csv'
84
85     with open(filename, 'w+', newline='') as csvFile:
86         writer = csv.writer(csvFile)
87         writer.writerow(['-----META DATA-----'])
88         writer.writerow(['SOFTWARE VERSION', '2.0.0'])
89         writer.writerow(['DEVICE OPERATOR', str(config.get('operator', 0))])
90         writer.writerow(['-----TEST ATTRIBUTES-----'])
91         writer.writerow(['FIELD', 'VALUE', 'UNIT'])
92         writer.writerow(['YEAR', dt.strftime("%Y")])
93         writer.writerow(['MONTH', dt.strftime("%m")])
94         writer.writerow(['DAY', dt.strftime("%d")])
95         writer.writerow(['TIME', dt.strftime("%H:%M:%S"), 'Local Time Zone'])
96         writer.writerow(['PLOT', str(config.get('plot_num', 0)), '#'])
97         writer.writerow(['HEIGHT', str(config.get('height', 0)), 'cm'])
98         writer.writerow(['TEMPERATURE', '40', 'C'])
99         writer.writerow(['HUMIDITY', '40', '%'])
100        writer.writerow(['LATITUDE', '40', 'angular degrees'])
101        writer.writerow(['LONGITUDE', '40', 'angular degrees'])
102        writer.writerow(['-----OPTIONAL DATA-----'])
103        writer.writerow(['PRE_TEST_NOTE_1', pre_notes[0]])
104        writer.writerow(['PRE_TEST_NOTE_2', pre_notes[1]])
105        writer.writerow(['PRE_TEST_NOTE_3', pre_notes[2]])
106        writer.writerow(['PRE_TEST_NOTE_4', pre_notes[3]])
107        writer.writerow(['PRE_TEST_NOTE_5', pre_notes[4]])
108        writer.writerow(['POST_TEST_NOTE_1', post_notes[0]])
109        writer.writerow(['POST_TEST_NOTE_2', post_notes[1]])
110        writer.writerow(['POST_TEST_NOTE_3', post_notes[2]])
111        writer.writerow(['POST_TEST_NOTE_4', post_notes[3]])
112        writer.writerow(['POST_TEST_NOTE_5', post_notes[4]])
113        writer.writerow(['BREAK_HEIGHT', str(config.get('break_height', 0)), 'cm'])
114        writer.writerow(['LCA_WEIGHT', '0', 'g'])
115        writer.writerow(['-----SENSOR CALIBRATION DATA (stored_value*A + B =
... raw_data)-----'])
116        writer.writerow(['SENSOR', 'A', 'B', 'UNIT', 'ID'])
117        writer.writerow(['LOAD_X', '0', '0', 'N', 'loadx1'])
118        writer.writerow(['LOAD_Y', '0', '0', 'Newton', 'loady1'])
119        writer.writerow(['IMU', '0', '0', 'Deg', 'imu1'])
120        writer.writerow(['POT', '0', '0', 'Deg', 'pot1'])
121        writer.writerow(['TEMP', '0', '0', 'C', 'templ'])
122        writer.writerow(['HUM', '0', '0', '%', 'hum1'])
123        writer.writerow(['-----TEST DATA-----'])
124        writer.writerow(['TIME (s)', 'ANGLE_POT', 'ANGLE_IMU', 'LOAD_X', 'LOAD_Y'])
125        datasets = ts.get_datasets()
126        for ds in datasets:
127            writer.writerow([ds.timestamp, ds.pot_angle, ds.imu_angle, ds.x_load,
... ds.y_load])
128
129
130        csvFile.close()
131
132    def on_leave(self):

```



```
133
134
135     self.graph.remove_plot(self.plot)
136     self.graph._clear_buffer()
137
138
139
140
141
```

```
1 <TestingInfoBox@Label>:
2     font_size: 25
3     color: 0,0,0,1
4     halign: 'center'
5     valign: 'center'
6     canvas.before:
7         Color:
8             rgba: .74,.74,.74,1
9         Rectangle:
10             pos: self.pos
11             size: self.size
12
13 <TestingScreen>:
14     name: 'testing_screen'
15     GranuContainer:
16         GranuSideArea:
17             GranuSideButton:
18                 text: 'Update\nNotes'
19                 on_release:
20                     root.move_to('note_screen') # Move to notes screen
21             GranuSideButton:
22                 text: "Start"
23                 on_release:
24                     root.move_to('test_in_progress_screen') # Move to test in progress screen
25             GranuSideButton:
26                 text: "Tests"
27                 on_release:
28                     root.move_to('tests_screen') # Move to tests screen
29             GranuSideButton:
30                 text: 'Back'
31                 on_release:
32                     root.move_to('main_screen') # Move to main screen
33         GranuContent:
34             GranuTitle:
35                 text: 'Testing'
36             GridLayout:
37                 cols: 2
38                 spacing: 10
39                 TestingInfoBox:
40                     id: height
41                     text: "Height: " + root.height_num + " cm"
42                 TestingInfoBox:
43                     id: operator
44                     text: "Operator: " + root.operator
45                 TestingInfoBox:
46                     id: plot
47                     text: "Plot: " + root.plot
48                 TestingInfoBox:
49                     id: time
50                     text: "Time: " + root.time
51             BoxLayout:
52                 orientation: 'horizontal'
53                 size_hint_y: None
54                 height: self.minimum_height
55                 ListTitle:
56                     text: 'Pre-test Notes'
57                 ListTitle:
58                     text: 'Post-test Notes'
59             BoxLayout:
60                 orientation: 'horizontal'
61                 StaticList:
62                     id: pretest
63                 StaticList:
64                     id: posttest
65
66
```

```
1  """
2  Testing Menu
3  """
4
5  from kivy.lang import Builder
6  from kivy.properties import NumericProperty
7  from kivy.properties import StringProperty
8  from kivy.properties import ListProperty
9  from kivy.clock import Clock
10 from Sensor import Sensor
11
12 from view.BaseScreen import BaseScreen
13 from view.StaticList import StaticList
14 import configurator as config
15 from view.elements import *
16 import datetime
17
18 Builder.load_file('view/screens/main/TestingScreen.kv')
19
20 ONE_SEC = 1
21
22 class TestingScreen(BaseScreen):
23     height_num = StringProperty("N/A")
24     plot = StringProperty("N/A")
25     operator = StringProperty("N/A")
26     time = StringProperty("N/A")
27     datasets = []
28
29     def on_pre_enter(self):
30         """Before the Screen loads, read the configuration file to get the current
31         list of notes. Show the default buttons."""
32         self.event = Clock.schedule_interval(self.update_time, ONE_SEC)
33         self.height_num = str(config.get('height', 0))
34         self.plot = str(config.get('plot_num', 0))
35         self.operator = config.get('operator', 'N/A')
36         self.time = datetime.datetime.now().strftime("%I:%M:%S %p")
37         # Get notes from config file
38         notes = config.get('notes', {
39             "pretest": [],
40             "posttest": [],
41             "bank": []
42         })
43         # Set the data
44         self.ids['pretest'].list_data = notes["pretest"]
45         self.ids['posttest'].list_data = notes["posttest"]
46
47     def update_time(self, obj):
48         self.time = datetime.datetime.now().strftime("%I:%M:%S %p")
49
50     def on_leave(self):
51         self.event.cancel()
52
53
```

```
1 #:import MeshLinePlot kivy.garden.graph.MeshLinePlot
2
3 <TestInProgressInfoBox@Label>:
4     font_size: 50
5     color: 0,0,0,1
6     halign: 'center'
7     valign: 'center'
8     color: 1,1,1,1
9     canvas.before:
10         Color:
11             rgba: .5,.4,.4,1
12         Rectangle:
13             pos: self.pos
14             size: self.size
15
16 <TestInProgressScreen>:
17     name: 'test_in_progress_screen'
18     GranuContainer:
19         GranuSideArea:
20             GranuNone:
21             GranuNone:
22             GranuNone:
23             GranuSideButton:
24                 text: 'Stop'
25                 on_release:
26                     root.move_to('testing_results_screen') # Move to testing results screen
27     GranuContent:
28         GranuTitle:
29             text: 'Test in Progress'
30         TestInProgressInfoBox:
31             id: test_time_box
32             text: "Test in progress for\n" + str(root.test_time) + " seconds"
33         Graph:
34             id: graph_test
35             plot: MeshLinePlot
36             background_color: .5, .5, .5, 1
37             xlabel: 'Time(s)'
38             ylabel: 'X_Load(adc)'
39             x_ticks_major: root.x_major
40             y_ticks_major: root.y_major
41             y_grid_label: True
42             x_grid_label: True
43             padding: 5
44             x_grid: True
45             y_grid: True
46             xmin: 0
47             ymin: 0
48             xmax: root.x_max
49             ymax: root.y_max
```

```
1  """
2  Test in Progress
3  """
4
5  import datetime
6
7  from kivy.lang import Builder
8  from kivy.properties import NumericProperty
9  from kivy.properties import StringProperty
10 from kivy.properties import ListProperty
11 from kivy.clock import Clock
12 from Dataset import Dataset
13 from Sensor import Sensor
14 from TestSingleton import TestSingleton
15 from Sensor import Sensor
16
17 from view.BaseScreen import BaseScreen
18 import configurator as config
19 from kivy.config import Config as KivyConfig
20 from view.elements import *
21 import datetime
22 import time
23 import math
24
25 from kivy.garden.graph import Graph, MeshLinePlot
26
27 Builder.load_file('view/screens/main/testing/TestInProgressScreen.kv')
28
29 INTERVAL = .003
30 SECOND_CAP = 1/INTERVAL
31
32 class TestInProgressScreen(BaseScreen):
33     test_time = NumericProperty(0)
34     x_max = NumericProperty()
35     y_max = NumericProperty()
36     x_major = NumericProperty()
37     y_major = NumericProperty()
38     temperature = 0
39     humidity = 0
40     location = 0
41     x_load = 0
42     y_load = 0
43     pot_angle = 0
44     imu_angle = 0
45     data_rate = 0
46     second_counter = 0
47     double_counter = 0
48     start_time = 0
49     start_timestamp = datetime.datetime.now().strftime("%I:%M:%S %p")
50
51     def on_pre_enter(self):
52         self.test_time = 0
53         self.temperature = 0
54         self.humidity = 0
55         self.location = 0
56         self.x_load = 0
57         self.y_load = 0
58         self.pot_angle = 0
59         self.imu_angle = 0
60         self.data_rate = 0
61         self.second_counter = 0
62         self.double_counter = 0
63         self.start_time = datetime.datetime.now()
64         self.datasets = []
65         self.x_max = 5
66         self.y_max = 2000
67         self.x_major = int(self.x_max/5)
68         self.y_major = int(self.y_max/5)
```

```
69     self.datasets = []
70     self.test_sensor = Sensor()
71     self.plot = MeshLinePlot(color=[1, 1, 1, 1])
72
73     self.event = Clock.schedule_interval(self.update_dataset, INTERVAL)
74     #ClockBaseInterruptBehavior.interupt_next_only = True
75 def find_max_x_load(self):
76     max = 0
77     for dataset in self.datasets:
78         if(dataset.x_load > max):
79             max = dataset.x_load
80     return max
81
82 def update_dataset(self, obj):
83     self.second_counter += 1
84     time_delta = datetime.datetime.now() - self.start_time
85     total_time_passed = time_delta.seconds + (time_delta.microseconds * .000001)
86     self.test_time = time_delta.seconds
87     if self.second_counter >= SECOND_CAP/2:
88         self.double_counter += 1
89         self.second_counter = 0
90         self.graph = self.ids['graph_test']
91         self.graph.remove_plot(self.plot)
92         self.graph._clear_buffer()
93         self.plot = MeshLinePlot(color=[1, 1, 1, 1])
94         last_index = len(self.datasets) - 1
95
96         self.x_max = math.ceil(self.datasets[last_index].timestamp / 5) * 5
97         #if(self.find_max_x_load() == 0):
98             # self.y_max = 10000
99         #else:
100             # self.y_max = math.ceil(self.find_max_x_load() / 10000) * 10000
101         self.x_major = int(self.x_max/5)
102         #self.y_major = int(self.y_max/5)
103
104
105
106
107
108     self.plot.points = [(self.datasets[i].timestamp, self.datasets[i].pot_angle) for
... i in range(0, len(self.datasets))]
109
110     self.graph.add_plot(self.plot)
111
112
113
114     sensor_values = self.test_sensor.get_sensor_data()
115     self.x_load = sensor_values["X Load"]
116     self.y_load = sensor_values["Y Load"]
117     self.pot_angle = sensor_values["Pot Angle"]
118     self.imu_angle = sensor_values["IMU Angle"]
119
120
121     new_dataset = Dataset(total_time_passed, self.x_load, self.y_load, self.pot_angle,
... self.imu_angle, self.data_rate)
122     self.datasets.append(new_dataset)
123     # This next chunk is what we actually have to change to read from the sensors
124     # self.temperature += 1
125     # self.humidity += 2
126     # self.location += 3
127     # self.x_load += 1
128     # self.y_load += 5
129     # self.pot_angle += 6
130     # self.imu_angle += 7
131
132 def on_pre_leave(self):
133     self.event.cancel()
134     ts = TestSingleton()
```

```
135         ts.clear_all()
136         ts.set_height(str(config.get('height', "")))
137         ts.set_plot(str(config.get('plot_num', "")))
138         config.set('break_height', "N/A")
139         #ts.set_pre_notes(str(config.get('height', "")))
140         #ts.set_post_notes(str(config.get('height', "")))
141         ts.set_operator(str(config.get('operator', "")))
142         ts.set_timestamp(self.start_timestamp)
143         ts.set_datasets(self.datasets)
144         self.datasets = []
145         self.graph.remove_plot(self.plot)
146         self.graph._clear_buffer()
147
148         #for dataset in self.datasets:
149
150     ... #print("Timestamp:",dataset.timestamp,"Temperature:",dataset.temperature,"Humidity:",dataset
151     ... .humidity,"Location:",dataset.location,"X
152     ... Load:",dataset.x_load,"Y_Load:",dataset.y_load,"Pot Angle:",dataset.pot_angle,"IMU
153     ... Angle",dataset.imu_angle,"CPU Time:",dataset.cpu_time)
154
155         #print("Timestamp:",dataset.timestamp,"Temperature:",dataset.temperature,"Data
156     ... Rate:",dataset.data_rate)
```

```
1 class TestSingleton:
2     class __TestSingleton:
3         def __init__(self):
4             self.clear_all()
5         def clear_all(self):
6             self.height = ""
7             self.plot = ""
8             self.pre_notes = ""
9             self.post_notes = ""
10            self.operator = ""
11            self.timestamp = ""
12            self.datasets = []
13            self.break_height = ""
14    instance = None
15    def __init__(self):
16        if not TestSingleton.instance:
17            TestSingleton.instance = TestSingleton.__TestSingleton()
18
19    def clear_all(self):
20        self.instance.clear_all()
21
22    def print_test(self):
23        print("Timestamp:", self.instance.timestamp)
24        print("Height:", self.instance.height)
25        print("Plot:", self.instance.plot)
26        print("Operator:", self.instance.operator)
27        print("Break Height:", self.instance.break_height)
28        print("Pre Notes:", self.instance.pre_notes)
29        print("Post Notes:", self.instance.post_notes)
30        print("First Dataset Temp:", self.instance.datasets[0].temperature)
31        print("Second Dataset Temp:", self.instance.datasets[1].temperature)
32
33    def get_height(self):
34        return self.instance.height
35    def set_height(self, height):
36        self.instance.height = height
37
38    def get_plot(self):
39        return self.instance.plot
40    def set_plot(self, plot):
41        self.instance.plot = plot
42
43    def get_pre_notes(self):
44        return self.instance.pre_notes
45    def set_pre_notes(self, pre_notes):
46        self.instance.pre_notes = pre_notes
47
48    def get_post_notes(self):
49        return self.instance.post_notes
50    def set_post_notes(self, post_notes):
51        self.instance.post_notes = post_notes
52
53    def get_operator(self):
54        return self.instance.operator
55    def set_operator(self, operator):
56        self.instance.operator = operator
57
58    def get_timestamp(self):
59        return self.instance.timestamp
60    def set_timestamp(self, timestamp):
61        self.instance.timestamp = timestamp
62
63    def get_datasets(self):
64        return self.instance.datasets
65    def set_datasets(self, datasets):
66        self.instance.datasets = datasets
67
68    def get_break_height(self):
```



```
69         return self.instance.break_height
70     def set_break_height(self, break_height):
71         self.instance.break_height = break_height
72
73     #class Dataset:
74     #    def __init__(self, name, test):
75     #        self.name = name
76     #        self.number = test
77
78     #a = TestSingleton()
79     #a.set_height("tall")
80     #d1 = Dataset("ben", "1")
81     #d2 = Dataset("sarah", "2")
82     #datasets1 = []
83     #datasets1.append(d1)
84     #datasets1.append(d2)
85     #a.set_datasets(datasets1)
86     #print("a is", a.get_height())
87     #print(a.get_datasets()[0].name)
88     #b = TestSingleton()
89     #b.set_height("short")
90     #d3 = Dataset("alexander", "3")
91     #d4 = Dataset("quigley", "4")
92     #datasets2 = []
93     #datasets2.append(d3)
94     #datasets2.append(d4)
95     #b.set_datasets(datasets2)
96     #b.clear_all()
97     #print("b is", b.get_height())
98     #print(b.get_datasets()[0].name)
99     #print("a is", a.get_height())
100    #print(a.get_datasets()[0].name)
101
102
103
104
```

```
1 <Test>
2     canvas.before:
3         Color:
4             rgba: (.0, 0.9, .1, .3) if self.selected else (0, 0, 0, 1)
5         Rectangle:
6             pos: self.pos
7             size: self.size
8
9 <TestList>
10     viewclass: 'Note'
11     SingleSelectableRecycleBoxLayout:
12
13 <TestsScreen>:
14     name: 'tests_screen'
15     GranuContainer:
16         GranuSideArea:
17             id: tests_buttons
18         GranuContent:
19             GranuTitle:
20                 text: 'Tests'
21             BoxLayout:
22                 orientation: 'horizontal'
23             NoteList:
24                 id: tests_list
25                 on_interaction:
26                     root.test_buttons()
27                 on_deselect_all:
28                     root.default_buttons()
29
```

```
1  """
2  Test in Progress
3  """
4
5
6  from kivy.lang import Builder
7
8  from kivy.properties import ListProperty
9  import configurator as config
10
11
12
13  from view.BaseScreen import BaseScreen
14  from view.SingleSelectableList import SingleSelectableList, SingleSelectableListBehavior,
... SingleSelectableRecycleBoxLayout
15  from view.elements import *
16
17  from os import listdir
18  from os.path import isfile, join
19
20  from kivy.garden.graph import Graph, MeshLinePlot
21
22  Builder.load_file('view/screens/main/testing/TestsScreen.kv')
23
24  class Test(SingleSelectableListBehavior, Label):
25      pass
26
27  class TestList(SingleSelectableList):
28      def update(self, k, val):
29          self.data = [{'text': str(x)} for x in self.list_data]
30
31
32  class TestsScreen(BaseScreen):
33
34      def __init__(self, **kwargs):
35          super(BaseScreen, self).__init__(**kwargs)
36          self.back_button = GranuSideButton(text = 'Back')
37          self.back_button.bind(on_release = self.go_back)
38          self.remove_button = GranuSideButton(text = 'Remove All')
39          self.remove_button.bind(on_release = self.remove_tests)
40          self.export_button = GranuSideButton(text = 'Export All')
41          self.export_button.bind(on_release = self.export_tests)
42          self.test_details_button = GranuSideButton(text = 'Test\nDetails')
43          self.test_details_button.bind(on_release = self.test_details)
44
45      def on_pre_enter(self):
46          self.test_filenames = [f for f in listdir("Tests") if isfile(join("Tests", f))]
47
48
49
50          self.default_buttons()
51
52          self.ids['tests_list'].list_data = self.test_filenames
53
54
55      def go_back(self, obj):
56          super(TestsScreen, self).back()
57
58      def remove_tests(self, obj):
59          print("We should remove all tests!")
60
61      def export_tests(self, obj):
62          print("We should export all tests!")
63
64      def test_details(self, obj):
65          print("We should show test details!")
66
67      # Button Changes
```

```
68
69     def default_buttons(self):
70         buttons = self.ids['tests_buttons']
71         buttons.clear_widgets()
72         buttons.add_widget(self.back_button)
73         buttons.add_widget(self.remove_button)
74         buttons.add_widget(self.export_button)
75         buttons.add_widget(Widget())
76
77     def test_buttons(self):
78         buttons = self.ids['tests_buttons']
79         buttons.clear_widgets()
80         buttons.add_widget(self.back_button)
81         buttons.add_widget(self.remove_button)
82         buttons.add_widget(self.export_button)
83         buttons.add_widget(self.test_details_button)
84
85
86     def on_leave(self):
87         pass
88
```

```
1 {
2   "title": "Qwerty",
3   "description": "A classical US Keyboard",
4   "cols": 15,
5   "rows": 5,
6   "normal_1": [
7     [ "~", "~", "~", 1], [ "1", "1", "1", 1], [ "2", "2", "2", 1],
8     [ "3", "3", "3", 1], [ "4", "4", "4", 1], [ "5", "5", "5", 1],
9     [ "6", "6", "6", 1], [ "7", "7", "7", 1], [ "8", "8", "8", 1],
10    [ "9", "9", "9", 1], [ "0", "0", "0", 1], [ "-", "-", "-", 1],
11    [ "=", "=", "=", 1], [ "\u232b", null, "backspace", 2]
12  ],
13  "normal_2": [
14    [ " ", null, "", 1.5], [ "q", "q", "q", 1], [ "w", "w", "w", 1],
15    [ "e", "e", "e", 1], [ "r", "r", "r", 1], [ "t", "t", "t", 1],
16    [ "y", "y", "y", 1], [ "u", "u", "u", 1], [ "i", "i", "i", 1],
17    [ "o", "o", "o", 1], [ "p", "p", "p", 1], [ "[", "[", "[", 1],
18    [ "]", "]", "]", 1], [ "\\", "\\", "\\", 1.5]
19  ],
20  "normal_3": [
21    [ " ", null, "", 1.8], [ "a", "a", "a", 1], [ "s", "s", "s", 1],
22    [ "d", "d", "d", 1], [ "f", "f", "f", 1], [ "g", "g", "g", 1],
23    [ "h", "h", "h", 1], [ "j", "j", "j", 1], [ "k", "k", "k", 1],
24    [ "l", "l", "l", 1], [ ";", ";", ";", 1], [ "'", "'", "'", 1],
25    [ " ", null, "", 2.2]
26  ],
27  "normal_4": [
28    [ "\u21ea", null, "capslock", 2.5], [ "z", "z", "z", 1], [ "x", "x", "x", 1],
29    [ "c", "c", "c", 1], [ "v", "v", "v", 1], [ "b", "b", "b", 1],
30    [ "n", "n", "n", 1], [ "m", "m", "m", 1], [ ",", ",", ",", 1],
31    [ ".", ".", ".", 1], [ "/", "/", "/", 1], [ "\u21ea", null, "capslock", 2.5]
32  ],
33  "normal_5": [
34    [ "space", " ", "spacebar", 13], [ "\u2a2f", null, "escape", 2]
35  ],
36  "shift_1": [
37    [ "~", "~", "~", 1], [ "!", "!", "!", 1], [ "@", "@", "@", 1],
38    [ "#", "#", "#", 1], [ "$", "$", "$", 1], [ "%", "%", "%", 1],
39    [ "^", "^", "^", 1], [ "&", "&", "&", 1], [ "*", "*", "*", 1],
40    [ "(", "(", "(", 1], [ ")", ")", ")", 1], [ "_", "_", "_", 1],
41    [ "+", "+", "+", 1], [ "\u232b", null, "backspace", 2]
42  ],
43  "shift_2": [
44    [ " ", null, "", 1.5], [ "Q", "Q", "Q", 1], [ "W", "W", "W", 1],
45    [ "E", "E", "E", 1], [ "R", "R", "R", 1], [ "T", "T", "T", 1],
46    [ "Y", "Y", "Y", 1], [ "U", "U", "U", 1], [ "I", "I", "I", 1],
47    [ "O", "O", "O", 1], [ "P", "P", "P", 1], [ "{", "{", "{", 1],
48    [ "}", "}", "}", 1], [ "|", "|", "|", 1.5]
49  ],
50  "shift_3": [
51    [ " ", null, "", 1.8], [ "A", "A", "A", 1], [ "S", "S", "S", 1],
52    [ "D", "D", "D", 1], [ "F", "F", "F", 1], [ "G", "G", "G", 1],
53    [ "H", "H", "H", 1], [ "J", "J", "J", 1], [ "K", "K", "K", 1],
54    [ "L", "L", "L", 1], [ ":", ":", ":", 1], [ "\", "\", "\", 1],
55    [ " ", null, "", 2.2]
56  ],
57  "shift_4": [
58    [ "\u21ea", null, "capslock", 2.5], [ "Z", "Z", "Z", 1], [ "X", "X", "X", 1],
59    [ "C", "C", "C", 1], [ "V", "V", "V", 1], [ "B", "B", "B", 1],
60    [ "N", "N", "N", 1], [ "M", "M", "M", 1], [ "<", "<", "<", 1],
61    [ ">", ">", ">", 1], [ "?", "?", "?", 1.5], [ "\u21ea", null, "capslock", 2.5]
62  ],
63  "shift_5": [
64    [ "space", " ", "spacebar", 13], [ "\u2a2f", null, "escape", 2]
65  ]
66 }
67
```

```
1 <UpdateScreen>:
2     name: 'update_screen'
3     GranuContainer:
4         GranuSideArea:
5             GranuSideButton:
6                 text: 'Back'
7                 on_release:
8                     root.back()
9     GranuContent:
10         GranuTitle:
11             text: 'Update'
12
```

```
1 from kivy.lang import Builder
2
3 from view.BaseScreen import BaseScreen
4
5 Builder.load_file('view/screens/settings/UpdateScreen.kv')
6
7 class UpdateScreen(BaseScreen):
8     pass
9
```

```
1 from .connections import *
2
3 class X_Load:
4
5     def __init__(self):
6         self.load = 0.0
7
8     def get_data(self):
9         try:
10             self.load = X_LOAD_CHAN.voltage*1000 #scale to milliVolts
11             return self.load
12         except:
13             return self.load
14
```



```
1 from .connections import *
2
3 class Y_Load:
4
5     def __init__(self):
6         self.load = 0.0
7
8     def get_data(self):
9         try:
10             self.load = Y_LOAD_CHAN.voltage*1000 #scale to milliVolts
11             return self.load
12         except:
13             return self.load
14
```