The eighth marketing requirement led to the following design specification: the master node shall display sensor data on a graphical user interface (GUI) in real time. This requirement was derived from usability, it is reasonable to assume that a user would want real time access to data through a GUI. The most sensible programming language for the GUI code is Python. Python is open-source code therefore it is free to use. It will run on all platforms and integrates easily.



Figure : Data rendering GUI.

The GUI takes advantage of Python’s extensive libraries. The data rendering GUI is starts on the pygame surface. Pygame is generally used to produce games and is a less conventional choice for a GUI but offers ease of use and a highly adaptive interface. By creating events when data is received the GUI data can be rendered on the pygame surface. Please refer to Figure 1 below for an example of the data rendering GUI. In Figure 1 represents the GUI previous to receiving any data. The surface is any picture image the user defines in the configuration file as the background. Sensed data display widgets are place on the surface per the user’s positions defined within the configuration file. Pyserial allows the GUI access to the serial port where incoming data is received. The GUI also utilizes threading; which allows the program to run multiple operations concurrently in the same processing space. This allows the GUI to receive data while rendering/updating previous data. Java script object notation (JSON) allows data to be interchangeable and easily formatted. The base station microcontroller outputs the sensed data in a specific format which allows the data rendering GUI to map the incoming data to its corresponding display widget, please refer to Figure 2 below for visual representation.

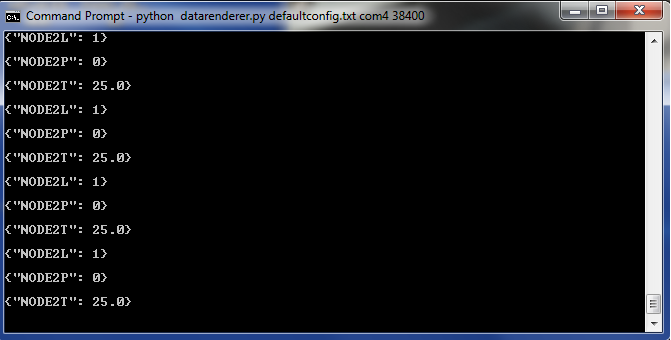


Figure 2: Incoming data.

The GUI consists of two main classes; sensed data display widget and data rendering. Currently the sensed data display widget is comprised of three sensed data types. The first sensed data display type is temperature; this widget will actively display the current sensor node temperature in degrees Celsius. Each sensor node in the network will have the capability of taking temperature data therefore each sensor node will have a temperature widget displayed on the GUI’s surface. Additionally, sensor nodes may have the ability to sense pressure or light. The light widgets will display a yellow signal indicator if in the presence of light or a grey signal indicator if not illuminated. The pressure display widget will behave similarly with green signal indicator for active pressure and white signal indicator for no pressure present. Refer to Figure 1, and Figure 3, for representation of the sensed data display widgets.

The data rendering class is where the bulk of the processing happens. The data rendering class consists of functions used to handle incoming data, render and update the GUI. The data handling function accesses the serial port and actively reads the serial data looking for an end of line character. Once the end of line character has been reached the GUI uses JSON to interpret the data. The decoded data is transformed into a pygame event which is used to control rendering of sensed data to the GUI. Refer to Figure 2, notice that only sensor node two is operating and that it has all three sensed data types associated. As the sensed data values change the GUI displays this data by updating the displayed value in increments until the new value is reached. To operate the data rendering GUI the user must create a configuration file which sets the number and type of widgets, their location on the surface, and the background image. These initial conditions are passed to the data rendering GUI as system variables. Figure 3 below represents the data rendering GUI after data was received from sensor node 2. Notice the temperature is showing 25° C, while the light sensor is currently illuminated and the pressure sensor is active.



Figure 3: Data rendering GUI after receiving data.

Attachment # 1

datarenderer.py

# Joe Linton

# DataRenderer

import pygame

import sys

import serial

import threading

import StringIO

import json

import math

import time

class TemperatureWidget():

def \_\_init\_\_(self, loc=(0,0), targetloc=None, temp=0.0, name='node'):

self.loc = loc

self.targetloc = targetloc

self.data = temp

self.name = name

self.showntemp = self.data

self.lasttemp = self.data

self.font = pygame.font.SysFont("Courier", 24, bold=True)

self.image = self.font.render("%s:%0.1f C" % (self.name, float(self.showntemp)), True, (0, 0, 0), (255, 255, 255))

def on\_render(self, screen):

if self.showntemp != self.lasttemp:

self.image = self.font.render("%s:%0.1f C" % (self.name, float(self.showntemp)), True, (0, 0, 0), (255, 255, 255))

self.lasttemp = self.data

rect = self.image.get\_rect()

rect = rect.move(self.loc)

screen.blit(self.image, rect)

rect = rect.inflate(4, 4)

coords = [(rect.left, rect.top),

(rect.right, rect.top),

(rect.right, rect.bottom),

(rect.left, rect.bottom)]

pygame.draw.aalines(screen, (0, 0, 0), True, coords)

if self.targetloc:

distfunc = lambda x, y: math.sqrt((x[0]-y[0])\*\*2 + (x[1]-y[1])\*\*2)

dist = 0

for i in coords:

testdist = distfunc(i, self.targetloc)

if testdist < dist or dist ==0:

curcoord = i

dist = testdist

pygame.draw.aaline(screen, (0, 0, 0), curcoord, self.targetloc)

def on\_update(self, dt):

self.showntemp -= (self.showntemp - self.data) \* (dt/1000.0)

class PressureWidget():

def \_\_init\_\_(self, loc=(0,0), targetloc=None, pres=0.0, name='node'):

self.loc = loc

self.targetloc = targetloc

self.data = pres

self.name = name

self.shownpres = self.data

self.lastpres = self.data

self.font = pygame.font.SysFont("Courier", 36, bold=True)

if self.data > 0:

self.image = self.font.render("P", True, (0, 0, 0), (19, 166, 50))

else:

self.image = self.font.render("P", True, (0, 0, 0), (255, 255, 255))

def on\_render(self, screen):

if self.shownpres != self.lastpres:

if self.data > 0:

self.image = self.font.render("P", True, (0, 0, 0), (19, 166, 50))

else:

self.image = self.font.render("P", True, (0, 0, 0), (255, 255, 255))

self.lastpres = self.data

rect = self.image.get\_rect()

rect = rect.move(self.loc)

screen.blit(self.image, rect)

rect = rect.inflate(4, 4)

coords = [(rect.left, rect.top),

(rect.right, rect.top),

(rect.right, rect.bottom),

(rect.left, rect.bottom)]

pygame.draw.aalines(screen, (0, 0, 0), True, coords)

if self.targetloc:

distfunc = lambda x, y: math.sqrt((x[0]-y[0])\*\*2 + (x[1]-y[1])\*\*2)

dist = 0

for i in coords:

testdist = distfunc(i, self.targetloc)

if testdist < dist or dist ==0:

curcoord = i

dist = testdist

pygame.draw.aaline(screen, (0, 0, 0), curcoord, self.targetloc)

def on\_update(self, dt):

self.shownpres -= (self.shownpres - self.data) \* (dt/1000.0)

class LightWidget():

def \_\_init\_\_(self, loc=(0,0), targetloc=None, light=0.0, name='node'):

self.loc = loc

self.targetloc = targetloc

self.data = light

self.name = name

self.shownlight = self.data

self.lastlight = self.data

self.font = pygame.font.SysFont("Courier", 36, bold=True)

if self.data > 0:

self.image = self.font.render("L", True, (0, 0, 0), (255, 205, 42))

else:

self.image = self.font.render("L", True, (0, 0, 0), (180, 180, 180))

def on\_render(self, screen):

if self.shownlight != self.lastlight:

if self.data > 0:

self.image = self.font.render("L", True, (0, 0, 0), (255, 205, 42))

else:

self.image = self.font.render("L", True, (0, 0, 0), (180, 180, 180))

self.lastlight = self.data

rect = self.image.get\_rect()

rect = rect.move(self.loc)

screen.blit(self.image, rect)

rect = rect.inflate(4, 4)

coords = [(rect.left, rect.top),

(rect.right, rect.top),

(rect.right, rect.bottom),

(rect.left, rect.bottom)]

pygame.draw.aalines(screen, (0, 0, 0), True, coords)

if self.targetloc:

distfunc = lambda x, y: math.sqrt((x[0]-y[0])\*\*2 + (x[1]-y[1])\*\*2)

dist = 0

for i in coords:

testdist = distfunc(i, self.targetloc)

if testdist < dist or dist ==0:

curcoord = i

dist = testdist

pygame.draw.aaline(screen, (0, 0, 0), curcoord, self.targetloc)

def on\_update(self, dt):

self.shownlight -= (self.shownlight - self.data) \* (dt/1000.0)

class DataRenderer():

def \_\_init\_\_(self, bgimage):

pygame.init()

flags = pygame.HWSURFACE | pygame.DOUBLEBUF

self.screen = pygame.display.set\_mode((640,480), flags)

self.bgimage = pygame.image.load(bgimage).convert\_alpha()

rect = self.bgimage.get\_rect()

self.screen = pygame.display.set\_mode((rect.width, rect.height), flags)

self.bgimage = pygame.image.load(bgimage).convert\_alpha()

self.clock = pygame.time.Clock()

self.quit = False

self.renderables = {}

self.datafile = serial.Serial((sys.argv[2]), (sys.argv[3]))

self.datahandlerthread = threading.Thread(name="DataHandler\_WorkerThread", target=self.data\_handler)

self.datahandlerthread.start()

def data\_handler(self):

while not self.quit:

data = self.datafile.readline()

print data

try:

jsonval = json.loads(data)

event = pygame.event.Event(pygame.USEREVENT, {"Data": jsonval})

pygame.event.post(event)

except ValueError:

pass

def mainloop(self):

while not self.quit:

dt = self.clock.tick(60)

self.on\_update(dt)

self.on\_render(self.screen)

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.quit = True

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

self.quit = True

if event.type == pygame.USEREVENT:

for i in event.Data.keys():

if i in self.renderables.keys():

self.renderables[i].data = event.Data[i]

pygame.quit()

sys.exit()

def on\_render(self, screen):

rect = self.bgimage.get\_rect()

self.screen.blit(self.bgimage, rect)

for name in self.renderables:

self.renderables[name].on\_render(screen)

pygame.display.flip()

def on\_update(self, dt):

for name in self.renderables:

self.renderables[name].on\_update(dt)

def add\_widget(self, widget, mapname):

self.renderables[mapname] = widget

if len(sys.argv) < 2:

raise ValueError('Requires a configuration file (see defaultconfig.txt)')

try:

with open(sys.argv[1]) as file:

config = json.load(file)

dr = DataRenderer(config["BackgroundImage"])

for item in config["Widgets"]:

if item["Type"] == 'Temperature':

dr.add\_widget(TemperatureWidget(tuple(item["Params"]["Location"]), tuple(item["Params"]["TargetLocation"]), name=item["DataMapName"]), item["DataMapName"])

if item["Type"] == 'Pressure':

dr.add\_widget(PressureWidget(tuple(item["Params"]["Location"]), tuple(item["Params"]["TargetLocation"]), name=item["DataMapName"]), item["DataMapName"])

if item["Type"] == 'Light':

dr.add\_widget(LightWidget(tuple(item["Params"]["Location"]), tuple(item["Params"]["TargetLocation"]), name=item["DataMapName"]), item["DataMapName"])

dr.mainloop()

except KeyboardInterrupt as e:

pygame.quit()

dr.quit = True