

```

1  ""
2  #####
3  Project: Sensor Subsystem
4
5  Authors: Team Spatium Lucis
6
7  Version: v2.0
8
9  Target Device: Raspberry Pi 3
10
11 Files: circadian.py, pir_sensor.py, rgb_sensor.py, send_circadian_values.py,
12        usr_sensor.py,
13        wait_for_cmd.py, start.py, stop.py, compensate.txt, config.txt, sensor_data.txt
14
15 Last edited: June 15, 2017
16 #####
17
18 (I) start.py:
19 This is the init script for the sensor subsystem. Usage:
20
21 $ python start.py
22
23 *****DO NOT USE sudo python!!!
24
25 (II) circadian.py:
26 This file serves as a custom Python module that houses functions that are common to all
27 the various sensor subsystem
28 files.
29
30 (a) Module imports:
31     (1) import subprocess
32     (2) import socket
33     (3) import MySQLdb
34     (4) import time
35     (5) import math
36     (6) import datetime
37
38 (b) Functions:
39     (1) def init_circadian_table():
40         This function creates base MASTER_CIRCADIANTABLE. Returns a list of lists
41         containing RGB brightnesses
42         percentages for each minute of the day for 7 AM wake up and 11 PM sleep.
43         Example index (not relative to the
44         actual table):
45         MASTER_CIRCADIANTABLE[420] == [ 30, 40, 50 ]. Therefore,
46         MASTER_CIRCADIANTABLE[420][0] == 30.
47
48     (2) def init_offset_table():
49         This function creates the base MASTER_OFFSET_TABLE. Returns a list of lists
50         containing rgb sensor offset
51         values for each minute of the day. These are needed because the rgb sensor
52         cannot properly pick up the values
53         of the lights so they are offset with these. This function calls
54         init_red_offset(), init_green_offset(),
55         and init_blue_offset() to generate the values for the MASTER_OFFSET_TABLE. This
56         table is for 7 AM to 11 PM
57         cycle. Example index (not relative to the actual table ):
58         MASTER_OFFSET_TABLE[420] == [ 120, 130, 140 ]. Therefore,
59         MASTER_OFFSET_TABLE[420][0] == 120.
60
61     (3) def init_red_offset(MASTER_OFFSET_TABLE):
62         This function generates the red offset values for the MASTER_OFFSET_TABLE.
63
64     (4) def init_green_offset(MASTER_OFFSET_TABLE):
65         This function generates the green offset values for the MASTER_OFFSET_TABLE.

```

```

58     (5) def init_blue_offset(MASTER_OFFSET_TABLE):
59         This function generates the blue offset values for the MASTER_OFFSET_TABLE.
60
61     (6) def init_master_lux_table():
62         This function creates the lux offset values for the MASTER_LUX_TABLE. Returns a
        list containing lux offset
63         values. These offset values are needed for the rgb sensor to calculate lux
        values because it doesn't naturally
64         do so. The values are for every minute of the day base on 7 AM/11 PM cycle.
        Example index (not relative to
65         the actual table):
66         MASTER_LUX_TABLE[420] == 50.
67
68     (7) def calc_user_tables(WAKE_UP_TIME, MASTER_CIRCADIAN_TABLE,
        MASTER_OFFSET_TABLE, MASTER_LUX_TABLE):
69         This function takes the user WAKE_UP_TIME, MASTER_CIRCADIAN_TABLE,
        MASTER_OFFSET_TABLE,
70         and the MASTER_LUX_TABLE then shifts these tables based on the WAKE_UP_TIME.
        Returns a tuple with the newly
71         shifted tables.
72
73     (8) def calc_Illuminance(lux, distance, angle):
74         This function takes a lux value, distance (in meters), and a viewing angle (in
        degrees). Returns the lumens
75         value. Uses toArea(), toSr(), and toRad() in calculation. Source:
76
77     (9) def get_pids():
78         Returns a list with all of the process ids (pids) of the following scripts:
79         pir_sensor.py, rgb_sensor.py, usr_sensor.py, wait_for_cmd.py, and
        send_circadian_values.py
80
81     (10) def get_system_time():
82         Returns the system time in mintues.
83
84     (11) def get_ip():
85         Returns the local IP address of the Raspberry Pi.
86
87     (12) def create_log(cursor, db, message, user_name):
88         This function takes a database cursor object, database object, a message
        string, and username string. Stores
89         the message into the database using execute_dB_query().
90
91     (13) def execute_dB_query(cursor, db, sql, sql_args):
92         This function takes a database cursor object, database object, an sql string,
        and a tuple of lists that
93         contain the sql query arguments and executes the query.
94
95     (14) def get_circadian_cmd(USER_CIRCADIAN_TABLE, PREV_PRIMARY_COLORS,
        PREV_SECONDARY_COLORS, IS_PRIMARY_DEG,
96         IS_SEC_ON, IS_SEC_DEG):
97         This function takes the USER_CIRCADIAN_TABLE, PREV_PRIMARY_COLORS,
        PREV_SECONDARY_COLORS, IS_PRIMARY_DEG,
98         IS_SEC_ON, and IS_SEC_DEG lists as input. Returns a tuple containing the
        circadian string, list for new
99         previous primary colors, and list for new previous secondary colors.
100
101 (III) pir_sensor.py:
102 This file is for the usage of the PIR motion sensor.
103
104     (a) Module imports:
105         (1) import time
106         (2) import os
107         (3) import signal
108         (4) import subprocess
109         (5) import circadian
110         (6) import MySQLdb
111         (7) import socket
112         (8) import datetime
113         (9) import RPi.GPIO as GPIO

```

```

114
115 (b) Signal handling:
116     These are the signal handler setups. When a kill -<number> is issued to the
117     LINUX system, if it is one of the
118     following numbers then it will be handled differently in the Python script.
119     These are basically software
120     interrupts.
121     signal.signal(3, handle_change_cmd)
122     signal.signal(4, catch_other_signals)
123     signal.signal(5, catch_other_signals)
124     signal.signal(6, handle_send_compensation)
125     signal.signal(7, handle_send_circadian)
126     signal.signal(8, handle_wait_for_cmd_dB_connect)
127     signal.signal(10, catch_other_signals)
128     signal.signal(11, handle_rgb_dB_connect)
129     signal.signal(12, handle_usr_dB_connect)
130     signal.signal(15, handle_send_circadian_dB_connect)
131
132     You will notice that throughout the scripts that some signal handler functions
133     don't do anything and that may
134     seem redundant. This was done on purpose for (1) to catch the signal and (2) to
135     keep the signal handling
136     consistent among the scripts.
137
138 (c) Functions:
139     (1) def catch_other_signals(signum, stack):
140         Does nothing but catch signals. Used with signal handling.
141
142     (2) def handle_change_cmd(signum, stack):
143         Catches kill -3. Simply performs time.sleep(3).
144
145     (3) def handle_send_compensation(signum, stack):
146         Catches kill -6. Simply performs time.sleep(3).
147
148     (4) def handle_send_circadian(signum, stack):
149         Catches kill -7. Simply performs time.sleep(3).
150
151     (5) def handle_wait_for_cmd_dB_connect(signum, stack):
152         Catches kill -8. Alerts the pir_sensor.py script that the wait_for_cmd.py
153         script has connected to the database.
154
155     (6) def handle_rgb_dB_connect(signum, stack):
156         Catches kill -11. Alerts the pir_sensor.py script that the rgb_sensor.py script
157         has connected to the database.
158
159     (7) def handle_usr_dB_connect(signum, stack):
160         Catches kill -12. Alerts the pir_sensor.py script that the usr_sensor.py script
161         has connected to the database.
162
163     (8) def handle_send_circadian_dB_connect(signum, stack):
164         Catches kill -15. Alerts the pir_sensor.py script that the
165         send_circadian_values.py script has connected to
166         the database.
167
168     (9) def handle_motion_detection(PIR_PIN):
169         This function is the hardware interrupt handler for the PIR sensor.
170
171 (IV) rgb_sensor.py:
172     This file is for the usage of the RGB sensor.
173
174 (a) Module imports;
175     (1) import time
176     (2) import os
177     (3) import signal
178     (4) import subprocess
179     (5) import circadian
180     (6) import MySQLdb
181     (7) import socket
182     (8) import datetime

```

```

175         (9) import smbus
176
177     (b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)
178         signal.signal(3, handle_change_cmd)
179         signal.signal(4, handle_sleep_mode)
180         signal.signal(5, handle_wake_up)
181         signal.signal(6, catch_other_signals)
182         signal.signal(7, handle_send_circadian)
183         signal.signal(8, handle_wait_for_cmd_dB_connect)
184         signal.signal(10, handle_pir_dB_connect)
185         signal.signal(11, catch_other_signals)
186         signal.signal(12, handle_usr_dB_connect)
187         signal.signal(15, handle_send_circadian_dB_connect)
188
189     (c) Functions:
190         (1) def catch_other_signals(signum, stack):
191             Does nothing but catch signals. Used with signal handling.
192
193         (2) def handle_change_cmd(signum, stack):
194             Catches kill -3. Handles when the user changes a parameter on the website.
195
196         (3) def handle_sleep_mode(signum, stack):
197             Catches kill -4. Updates database with sensor reading of 0 when system goes
198             into sleep mode.
199
200         (4) def handle_wake_up(signum, stack):
201             Catches kill -5. Used in waking from sleep mode.
202
203         (5) def handle_send_circadian(signum, stack):
204             Catches kill -7. Simply does time.sleep(3).
205
206         (6) def handle_wait_for_cmd_dB_connect(signum, stack):
207             Catches kill -8. Tells rgb_sensor.py that wait_for_cmd.py connected to the
208             database.
209
210         (7) def handle_pir_dB_connect(signum, stack):
211             Catches kill -10. Tells rgb_sensor.py that pir_sensor.py connected to the
212             database.
213
214         (8) def handle_usr_dB_connect(signum, stack):
215             Catches kill -12. Tells rgb_sensor.py that usr_sensor.py connected to the
216             database.
217
218         (9) def handle_send_circadian_dB_connect(signum, stack):
219             Catches kill -15. Tells rgb_sensor.py that sends_circadian_values.py connected
220             to the database.
221
222 (V) send_circadian_values.py:
223 This file is responsible for sending circadian values to the lighting subsystem. This
224 sends compensation values as well.
225
226     (a) Module imports:
227         (1) import time
228         (2) import os
229         (3) import signal
230         (4) import subprocess
231         (5) import circadian
232         (6) import MySQLdb
233         (7) import socket
234         (8) import datetime
235
236     (b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)
237         signal.signal(3, handle_change_cmd)
238         signal.signal(4, handle_sleep_mode)
239         signal.signal(5, handle_wake_up)
240         signal.signal(6, handle_send_compensation)
241         signal.signal(7, catch_other_signals)
242         signal.signal(8, handle_wait_for_cmd_dB_connect)
243         signal.signal(10, handle_pir_dB_connect)

```

```

238     signal.signal(11, handle_rgb_dB_connect)
239     signal.signal(12, handle_usr_dB_connect)
240     signal.signal(15, catch_other_signals)
241
242 (c) Functions:
243     (1) def catch_other_signals(signum, stack):
244         Catches signals. Does nothing else.
245
246     (2) def handle_change_cmd(signum, stack):
247         Catches kill -3. Used for when a user changes something from the website.
248
249     (3) def handle_sleep_mode(signum, stack):
250         Catches kill -4. Sends values to the lighting subsystem to put it to sleep.
251
252     (4) def handle_wake_up(signum, stack):
253         Catches kill -5. Makes the script send a value to wake the lights up.
254
255     (5) def handle_send_compensation(signum, stack):
256         Catches kill -6. Sends compensation values to the lighting subsystem.
257
258     (6) def handle_wait_for_cmd_dB_connect(signum, stack):
259         Catches kill -8. Alerts the send_circadian_values.py that wait_for_cmd.py
        connected to the DB.
260
261     (7) def handle_pir_dB_connect(signum, stack):
262         Catches kill -10. Alerts the send_circadian_values.py that pir_sensor.py
        connected to the DB.
263
264     (8) def handle_rgb_dB_connect(signum, stack):
265         Catches kill -11. Alerts the send_circadian_values.py that rgb_sensor.py
        connected to the DB.
266
267     (9) def handle_usr_dB_connect(signum, stack):
268         Catches kill -12. Alerts the send_circadian_values.py that usr_sensor.py
        connected to the DB.
269
270 (VI) usr_sensor.py:
271 This file is responsible for the usage of the ultra sonic range sensor.
272
273 (a) Module imports:
274     (1) import time
275     (2) import os
276     (3) import signal
277     (4) import subprocess
278     (5) import circadian
279     (6) import MySQLdb
280     (7) import socket
281     (8) import datetime
282     (9) import math
283     (10) import RPi.GPIO as GPIO
284
285 (b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)
286     signal.signal(3, catch_other_signals)
287     signal.signal(4, catch_other_signals)
288     signal.signal(5, catch_other_signals)
289     signal.signal(6, catch_other_signals)
290     signal.signal(7, catch_other_signals)
291     signal.signal(8, handle_wait_for_cmd_dB_connect)
292     signal.signal(10, handle_pir_dB_connect)
293     signal.signal(11, handle_rgb_dB_connect)
294     signal.signal(12, catch_other_signals)
295     signal.signal(15, handle_send_circadian_dB_connect)
296
297 (c) Functions:
298     (1) def catch_other_signals(signum, stack):
299         Catches signals. Does nothing else.
300
301     (2) def handle_wait_for_cmd_dB_connect(signum, stack):
302         Catches kill -8. Tells usr_sensor.py that wait_for_cmd.py connected to the DB.

```

```

303
304     (3) def handle_pir_dB_connect(signum, stack):
305         Catches kill -10. Tells usr_sensor.py that pir_sensor.py connected to the DB.
306
307     (4) def handle_rgb_dB_connect(signum, stack):
308         Catches kill -11. Tells usr_sensor.py that rgb_sensor.py connected to the DB.
309
310     (5) def handle_send_circadian_dB_connect(signum, stack):
311         Catches kill -15. Tells usr_sensor.py that send_circadian_values.py connected
            to the DB.
312
313 (VII) wait_for_cmd.py:
314 This file is responsible for receiving commands from the website.
315
316     (a) Module imports:
317         (1) import time
318         (2) import os
319         (3) import signal
320         (4) import subprocess
321         (5) import circadian
322         (6) import MySQLdb
323         (7) import socket
324         (8) import datetime
325
326     (b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)
327         signal.signal(3, catch_other_signals)
328         signal.signal(4, handle_sleep_mode)
329         signal.signal(5, handle_wake_up)
330         signal.signal(6, catch_other_signals)
331         signal.signal(7, catch_other_signals)
332         signal.signal(8, catch_other_signals)
333         signal.signal(10, handle_pir_dB_connect)
334         signal.signal(11, handle_rgb_dB_connect)
335         signal.signal(12, handle_usr_dB_connect)
336         signal.signal(15, handle_send_circadian_dB_connect)
337
338     (c) Functions:
339         (1) def catch_other_signals(signum, stack):
340             Catches signals and does nothing else.
341
342         (2) def handle_pir_dB_connect(signum, stack):
343             Catches kill -10. Tells wait_for_cmd.py that pir_sensor.py connected to the DB.
344
345         (3) def handle_rgb_dB_connect(signum, stack):
346             Catches kill -11. Tells wait_for_cmd.py that rgb_sensor.py connected to the DB.
347
348         (4) def handle_usr_dB_connect(signum, stack):
349             Catches kill -12. Tells wait_for_cmd.py that usr_sensor.py connected to the DB.
350
351         (5) def handle_send_circadian_dB_connect(signum, stack):
352             Catches kill -15. Tells wait_for_cmd.py that send_circadian_values.py connected
            to the DB.
353
354         (6) def handle_sleep_mode(signum, stack):
355             Catches kill -4. Tells the script that the system entered sleep mode.
356
357         (7) def handle_wake_up(signum, stack):
358             Catches kill -5. Tells the script that the system exited sleep mode.
359
360         (8) def boot_up():
361             Initial database check to pair with the lighting subsystem and retrieve
            previous sensor subsystem settings if
362             any.
363
364 (VIII) stop.py
365 This script is used for killing the sensor subsystem. Usage:
366
367 $ python stop.py
368

```

```

369  ****DO NOT USE sudo python!!!
370
371  (IX) pause.py
372  This script is used for suspending the sensor and lighting subsystems. Usage:
373
374  $ python pause.py
375
376  ****DO NOT USE sudo python!!!
377
378  (X) compensate.txt:
379  Holds the compensation data.
380
381  (XI) sensor_data.txt:
382  Holds the sensor readings. Uses in compensation.
383
384  (XII) config.txt:
385  Holds the values that were sent by the user. Needed because DB was misbehaving.
386
387  #####
388  Project: Lighting Subsystem
389
390  Authors: Team Spatium Lucis
391
392  Version: v1.0
393
394  Target Device: Raspberry Pi 3
395
396  Files: lighting_sub.py, pause.py
397
398  Last edited: August 10, 2017
399
400  Note: This code is not as polished as the sensor subsystem one. I didn't really get the
401  time to separate
402  the code into different scripts and stuff like the sensor code because of my summer
403  class/work schedule.
404  It's okay though because the bottle necks in performance came from the sensor codes
405  various threading
406  locks, delays, etc. and the lighting code doesn't really have these. Most of the thread
407  are more or less
408  independent, and the current code, although long and at points redundant, still works
409  like a charm :)
410
411  #####
412
413  (I) lighting_sub.py:
414  This is the init and main script for the lighting subsystem. Usage:
415
416  $ sudo python lighting_sub.py
417
418  NOTE: You may get some error saying that some address is already in use. If this
419  happens then do the
420  following:
421      (1) Perform a CTRL + Z
422      (2) Type ps -al and hit enter
423      (3) Locate the 'pid' for 'sudo'
424      (4) Type sudo kill -9 <pid for 'sudo'> and hit enter
425      (5) Try to run the lighting_sub.py script again
426
427      (a) Module Imports:
428          (1) import time
429          (2) import socket
430          (3) import threading
431          (4) import RPi.GPIO as GPIO
432          (5) import MySQLdb
433          (6) import wiringpi
434          (7) import sys
435          (8) import os

```

```

430         (9) import signal
431
432     (b) Section of code after imports and before the functions:
433     There is a chunk of code after the imports and before the functions that
434     basically setting up
435     the PWM pin information for the lights. The wiringpi python module was used
436     because the RPi.GPIO
437     module would produce this ridiculous flickering that was comparable to a camera
438     flash when the
439     lights were dim (like early morning/late evening values). Trust me, continue to
440     use this module.
441     Because the Rpi3 only has really 1 hardware PWM pin, we used soft PWMs for the
442     lights. These
443     actually look pretty good but will never beat a hardware PWM.
444     Example soft PWM python code:
445     wiringpi.wiringPiSetupGpio()
446     GPIO.setmode(GPIO.BCM) # from the Rpi.GPIO module. Needed to set pin mode
447     and for relays.
448     wiringpi.pinMode(17, 17)
449     wiringpi.softPwmCreate(17, 0, 100)
450
451     This will create a softPWM channel on pin 17(BCM) with frequency 100Hz.
452
453     BCM Pins:
454     (1) Pin 17: Primary Red
455     (2) Pin 27: Primary Green
456     (3) Pin 22: Primary Blue
457     (4) Pin 6: Secondary Red #the code has a comment saying pin 29 for some
458     reason. Ignore it.
459     (5) Pin 13: Secondary Green
460     (6) Pin 26: Secondary Blue
461
462     The rest of this section of code simply establishes some mutex locks for the
463     threads and a couple
464     of global variables.
465
466     (c) Functions:
467     (1) def get_ip():
468     Gets the local IP address of the lighting subsystem and returns it as a string
469
470     (2) def boot_up():
471     Checks the database for an existing entry of the local IP address. If it exists
472     then move on to
473     wait for the sensor subsystem to connect. If it does not exist then it will be
474     added to the
475     database and wait to be paired.
476
477     (3) def begin_threading():
478     Creates the pir_thread (waits for sleep/wake values), delete_thread (when the
479     system is deleted),
480     the comp_thread (waits for compensation values), and the light_cmd thread
481     (waits for circadian
482     commands). Also creates some threading events to help with synchronization.
483
484     (4) def delete_cmd():
485     Receives a command from the sensor subsystem that the system is begin deleted.
486     Turns the lights
487     off then ends the script.
488
489     (5) def comp_cmd():
490     Receives the compensation command from the sensor subsystem (see sensor
491     subsystem for command format)
492     and brightens/dims the lights accordingly. There are 64 combinations depending
493     on which lights are
494     brightening or dimming (6 lights -> 2^6 = 64). I basically made a truth table
495     for the lights.
496
497     (6) Handler Functions:
498     These functions are used for the threads that are spawned in the comp_cmd() and

```



```
483     light_cmd() threads.
484     They are used to change the primary or secondary lights to a certain brightness
485     using the PWM.
486
487     (7) def PIR_cmd():
488         The function for the pir thread. Receives a command from the sensor subsystem
489         to turn the lights
490         off for sleep mode, or wake them up.
491
492     (8) def light_cmd():
493         This function is basically identical to the comp_cmd() function except it uses
494         a different
495         command string format. See sensor subsystem for the proper format.
496
497 (II) pause.py:
498 This script is used to suspend the lighting subsystem for whatever reason. Usage:
499
500 $ sudo python pause.py
501 """
```