Digital Turnstile: A Smarter Laser Tripwire

Alex Klimaj, *Cal Poly San Luis Obispo Department of Electrical Engineering*

Tyler Kowallis, *Department of Computer Engineering*

**Abstract—Many storefronts and businesses use simple motion detectors in doorways to alert employees when an individual has entered or left the premises. The intelligent tripwire system is able to differentiate between individuals entering or leaving the premises, generate analytics that document the peak business hours, and record the current occupancy of a facility and post it to a social media resource.**

***Index Terms—*Digital Circuits, Diode lasers, Microcontroller, Printed circuits, Python, Raspberry Pi**

I. Introduction

T

raditional doorway laser tripwires are used to alert employees when an individual enters or exits a building. The chime that the sensors trigger alerts employees that may be busy tending to the facility that an individual may be in need of service. For facilities such as San Luis Obispo, California’s SLO Op rock climbing facility, however, the laser tripwire and chime is as much of a nuisance as it is useful. On busy days when all employees are helping on the gym floor, a guest exiting the facility will trigger the tripwire alarm. An employee will temporarily stop the service she is giving to other guests to service what she believes to be an individual entering the facility. The misleading signal ultimately wastes the employees’ time and degrades the service they offer to guests on the gym floor.

An intelligent laser tripwire system, which we dub a digital turnstile, will solve this problem as well as provide additional value through analytics to the facility that the device is installed in. A single digital turnstile unit will be a series of two staggered lasers on one side of a doorway and photodetectors on the other side of the doorway. The photodetector signals will be processed by commercial off-the-shelf (COTS) integrated circuits and the resulting information will be sent to a microcontroller, much like a traditional tripwire system.

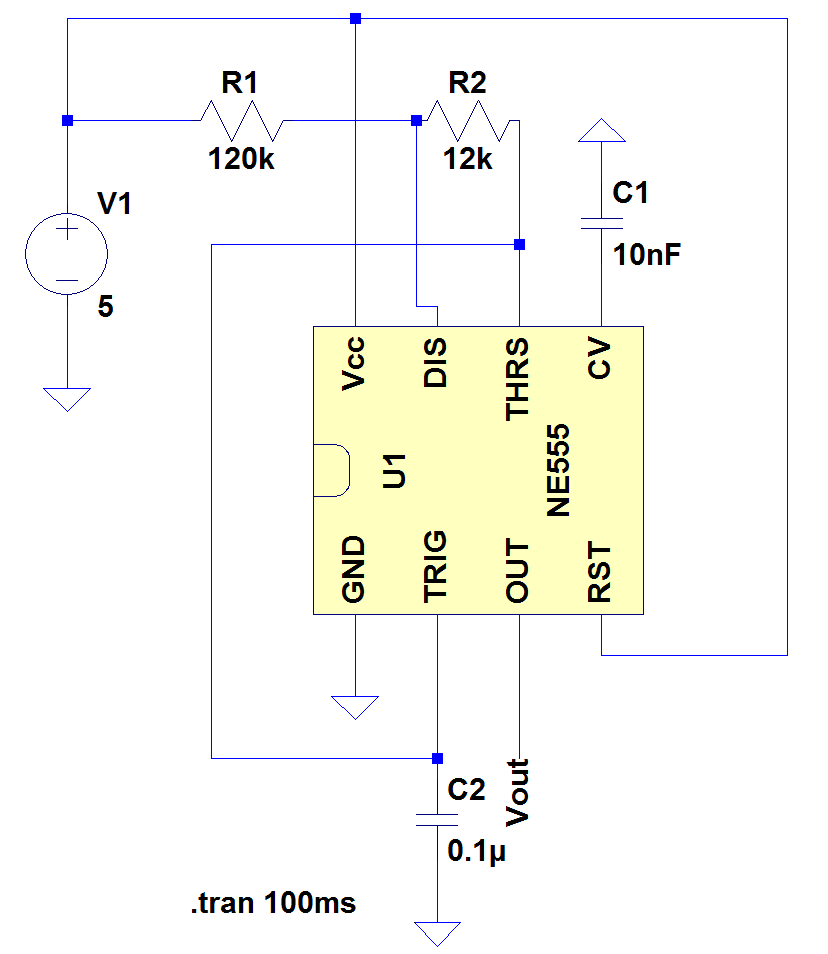
The microcontroller will analyze the turnstile signals and decide whether or not to sound a chime based on whether or not an entry was detected (as opposed to an exit). The microcontroller will also host a small web server that will display statistics such as current facility occupancy and peak business hours. These features distinguish the digital turnstile from a simple laser tripwire.

For facilities with a single doorway, a single digital turnstile will improve upon the traditional laser tripwire system. However, the digital turnstiles will be networkable to extend their functionality to facilities with multiple doorways. The master turnstile, the one that hosts the web server, is chosen explicitly through software.

We take SLO Op as our design client, so the digital turnstile solution detailed in this paper is tailored for use in that facility. Our design will be tested in the SLO Op facility, which contains a main door and a members-only door, so we will employ a network of digital turnstiles using the facility’s wireless network as the host network. We will validate our design and its usefulness by collecting analytics over days of SLO Op’s regular business hours. We will also survey employees to determine the digital turnstile’s practicality and usefulness.

II. Design

The proof-of-concept design consisted of the laser module and photodetector and microcontroller (a Raspberry Pi) module aligned on a wooden board for portability during design and testing. The laser module in Figure 1 consists of a 555 timer and discrete resistors and capacitors configured to produce a 100 Hz square wave of 10% duty cycle. This signal, Vout, drives the TTL-controlled lasers, activating the laser on every logic-high excursion of the wave. The 5 V source is taken from the same line that powers the microcontroller.



**Figure 1—The laser timer circuit that drives the TTL-controlled lasers.**

We used a 0.5 mW laser diode that radiated a 650 nm red beam. The laser diodes are spaced approximately 3 cm apart.

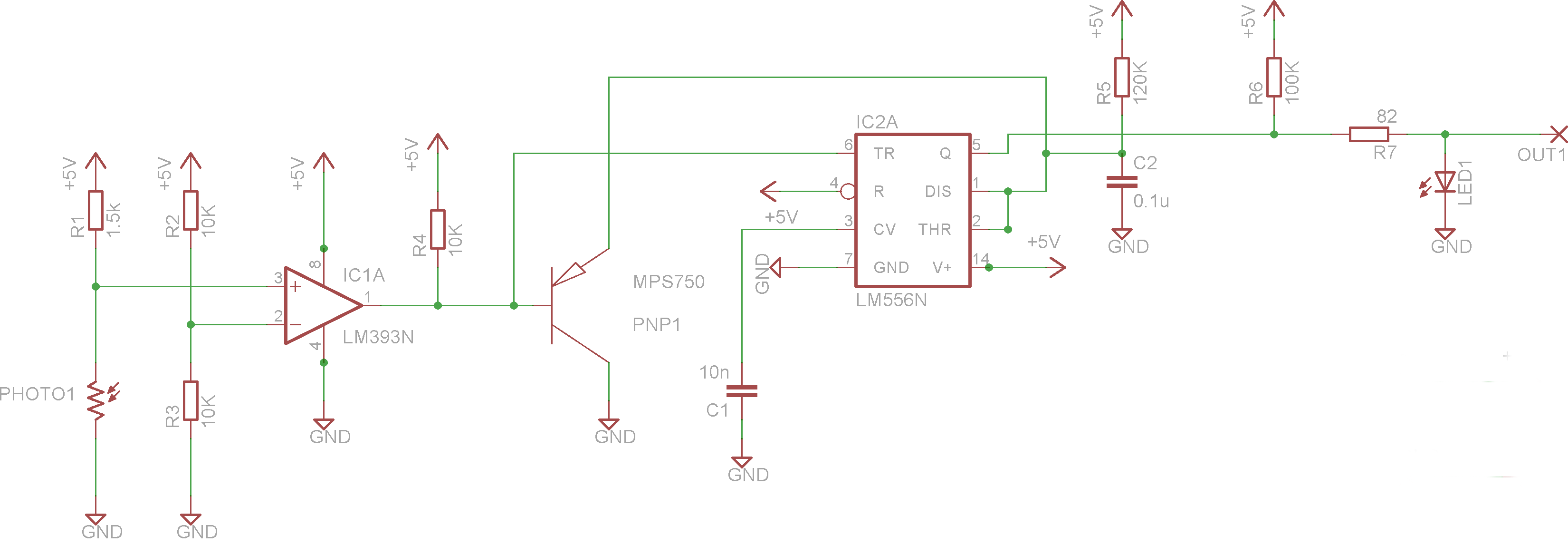
Across the board lies the laser detector (schematic in Figure 2) and microcontroller module. We used simple photodetectors to capture the laser pulses. The photodetectors act as a resistance in a voltage divider. The output voltage of the divider is fed into the LM393N comparator to generate a well-defined digital signal. A 555 timer acting as a missing-pulse detector creates a logic-high signal on its Q pin. This signal illuminates an LED to signal that the digital turnstile tripwire has been broken. The output signal, OUT1, is monitored by the Raspberry Pi, and a low-to-high transition on this line causes an interrupt on the microcontroller. The second photodetector in the circuit follows the same schematic in Figure 2. The 5 V rail is taken from the same line that powers the Raspberry Pi.

The microcontroller receives interrupts from each photodetector system on unique pins. So when an interrupt occurs, the microcontroller checks the pin that caused the interrupt to determine if the beam closest to the outside of the facility was broken first. If this beam was broken first and then the beam further inside the facility is broken, the microcontroller increments an internal counter of the current number of people in the facility and plays a chime through the facility’s audio system. If the microcontroller detects the beams broken in the opposite order, it does not play a chime but does decrement the occupancy counter.

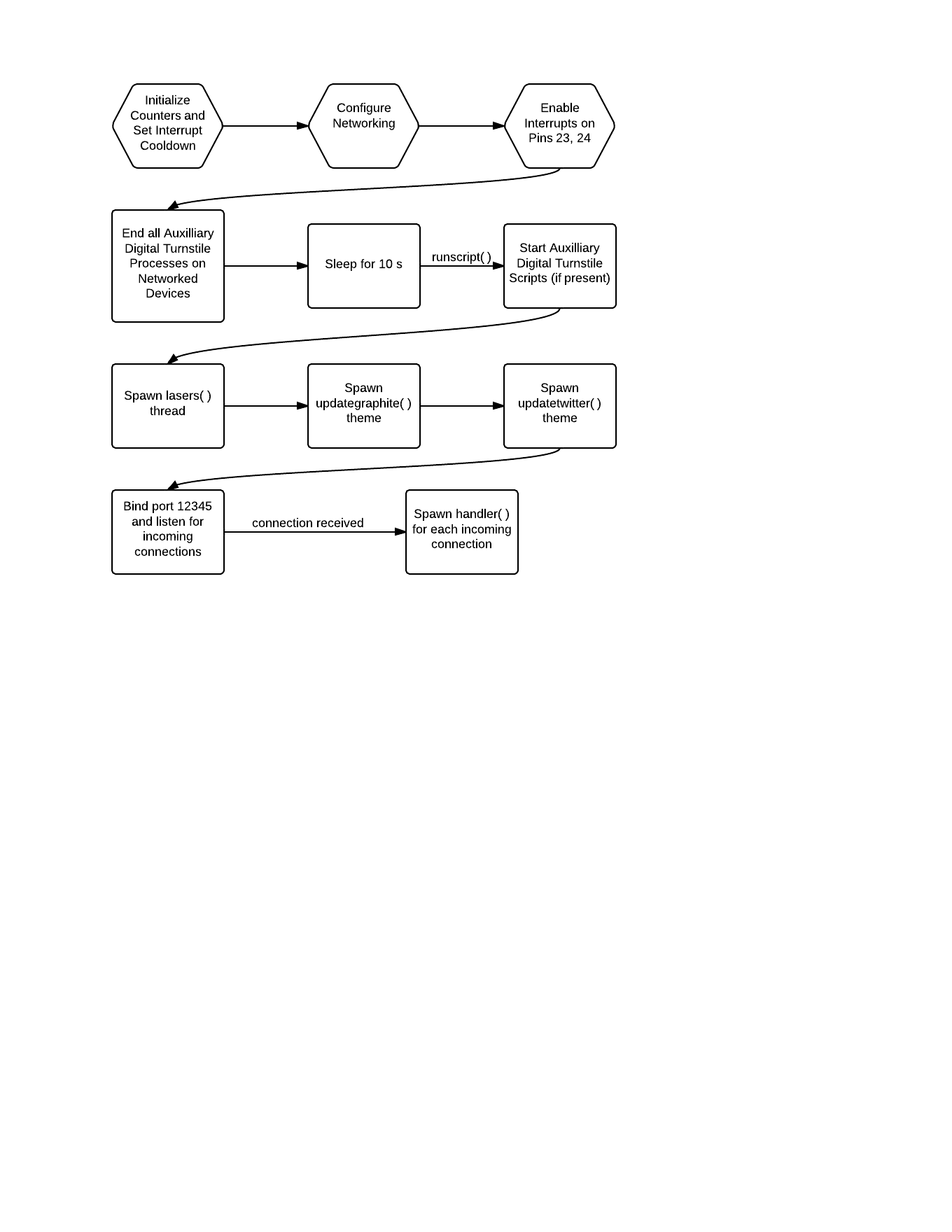
These counters are used in the analytics provided as part of our system. We are using Python as our design language, so we use the Tweepy Twitter-for-Python library and the Carbon real-time graphing library to share the analytics. The Raspberry Pi tweets the current facility occupancy every 5 minutes through the SLO Op Twitter account if the current occupancy has changed since the last measurement. Carbon displays statistics over time in a graphical format on a web interface, so information such as the day’s peak business hours, the facility’s current occupancy, and the rate at which guests arrive and depart can be shared online through SLO Op’s website.

The formal software flow diagram for the main process is shown in Figure 3 and a flow diagram of the thread it spawns that controls the lasers is shown in Figure 4.

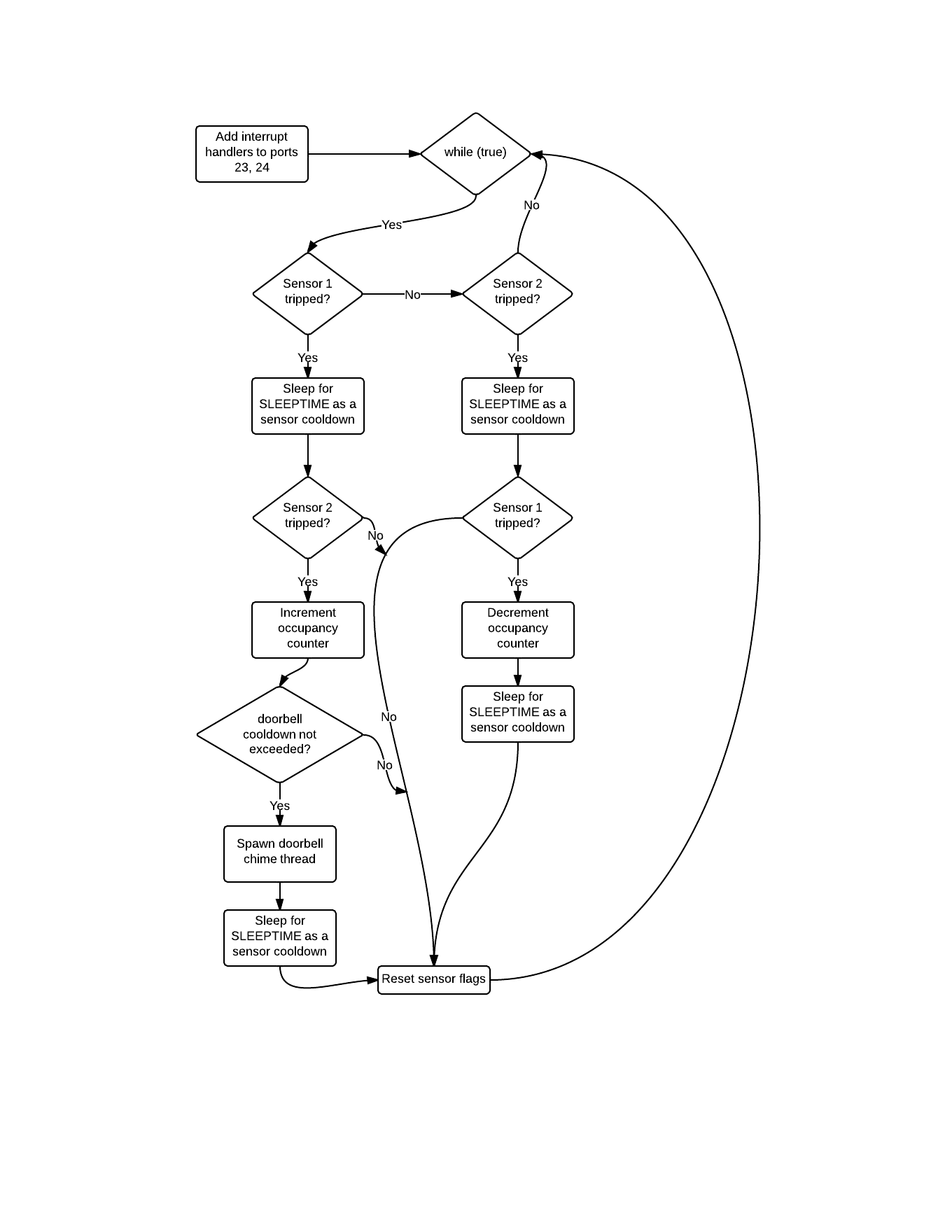
After we performed tests to verify that the design functioned properly, we drafted a printed circuit board in Eagle that would replace the protoboard that the photodetector circuit was designed on. To protect and enclose the modules, we 3D-printed a case to hold the laser module (Figure 5) and the detector and microcontroller module (Figure 6).



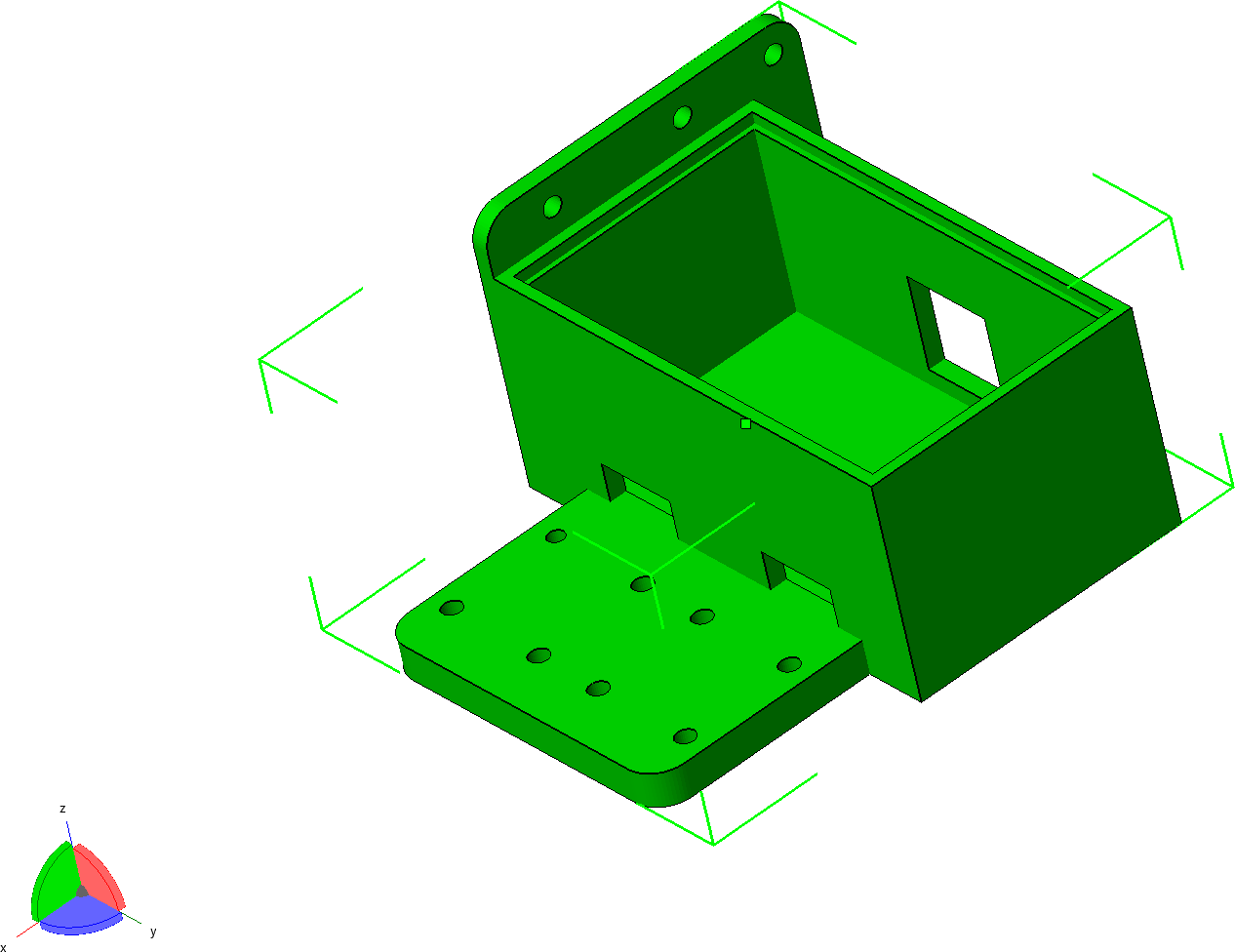
**Figure 2—The laser detector circuit consists of a photodetector as part of a voltage divider, a comparator to clean up the logic signal produced by the photodetector, and a 555 timer acting as a missing-pulse detector.**

****

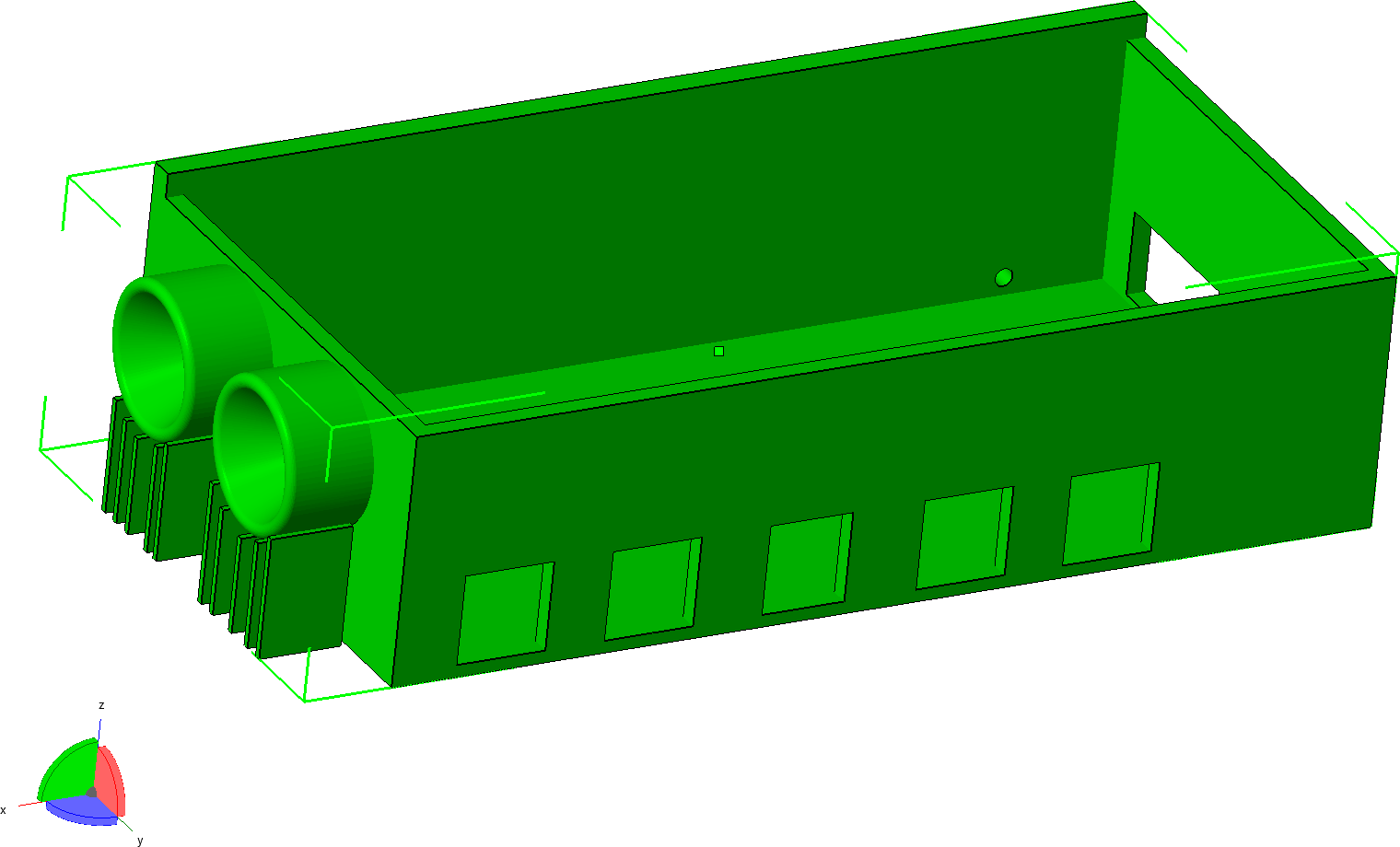
**Figure 3—The software flow diagram of the main process (for the turnstile that hosts the analytics server). This process spawns threads to handle interrupts and analytics.**

****

**Figure 4—The software flow diagram for the thread that handles interrupts triggered from tripping lasers. Laser 1 is the laser closest to the outside of the facility. Laser 2 is further inside the facility. If laser 1 is tripped and then laser 2 is tripped and the sufficient doorbell cooldown is achieved, the doorbell chime is sounded.**

****

**Figure 5—The 3D-printed case for the laser module. This case serves to keep the lasers aligned and also to protect the module.**

****

**Figure 6—The receiver and microcontroller module. This modules serves to keep the receivers spaced and aligned correctly with the laser module. The case also protects all of the components inside.**

III. Results

We installed the laser and receiver modules in the main and employees-only doors in the SLO Op rock climbing gym. Figure 7 shows the modules installed at the main door. We tracked the proper operation of the digital turnstile by watching the statistics posted to Twitter over time. At the time of this writing, the Twitter handle for our reports is @SloOpDoorBot. We also tracked the building occupancy using the graph produced by the Carbon Python module.

Both Twitter and the Carbon analytics are based off of the same metric, so the measures were always consistent between the two services. The Carbon graphs allow one to see the occupancy over time whereas the Twitter service reports the same information in a linear micro-blog post fashion that is less intuitive for viewing an entire day’s occupancy trend but more effective for finding the current facility occupancy. The Twitter service is also mobile-device-friendly, allowing customers to receive updates about the facility occupancy real-time to their mobile devices.

Employees at the SLO Op climbing gym find the digital turnstile very reliable and effective. Because the doorbell only sounds when a visitor is entering, the employees spend less time servicing a departing visitor employees believed was entering and more time servicing incoming customers and climbers on the gym floor. The cooldown (see the code in Appendix B) for the lasers works effectively as well, keeping the doorbell from sounding several times in quick succession if a guest enters and trips the wires with her legs and perhaps a jacket around her waist. SLO Op has also integrated the analytics and statistics reports into their website, linking to the Twitter handle that displays the current occupancy.

IV. Conclusion

The digital turnstile is an effective, intelligent replacement for the traditional laser tripwire used in doorways. Its advantages over traditional systems are the ability to differentiate between incoming personnel and outgoing personnel and the ability to post statistics of occupancy online. This allows the system to integrate with social media services offered by the facility that uses the digital turnstile. The system’s intelligence leads to better customer service by giving the employees signals about individuals entering the store without a distracting graphical message. The employees can hear the tone wherever they are in the store and do not need to monitor camera feeds or move around the gym floor to determine if a guest entered or left. Finally, the networkability of the system allows for multiple digital turnstiles to be configured together to provide coverage of a facility with multiple doorways. The digital turnstiles are networked over the customer’s private network, so the device is easily adaptable and scalable.



**Figure 7—The digital turnstile installed at the main door of the SLO Op facility. On the right is the laser module, and on the left is the receiver and microcontroller module.**

**Appendix A—Bill of Materials**

|  |  |  |  |
| --- | --- | --- | --- |
| **Part** | **Amount** | **Supplier** | **Total Price** |
| Long Range WiFi USB with Antenna for Raspberry Pi | 1 | Amazon | $21.79 |
| *Edimax EW-7811Un 150 Mbps Wireless 11n Nano Size USB Adapter* | 1 | Amazon | $10.89 |
| COM-09273 - 555 Timer  ($0.95 ea.) | 2 | Sparkfun Electronics | $1.90 |
| SEN-09088 - Mini Photocell  ($1.50 ea.) | 2 | Sparkfun Electronics | $3.00 |
| COM-08654 - Laser Module - Red with TTL Control  ($11.95 ea.) | 2 | Sparkfun Electronics | $23.90 |
| DEV-11837 - Raspberry Pi - Model A | 1 | Sparkfun Electronics | $29.95 |
| DEV-11546 - Raspberry Pi - Model B | 1 | Sparkfun Electronics | $39.95 |
| PRT-00643 - RJ45 8-Pin Connector  ($1.50 ea.) | 2 | Sparkfun Electronics | $3.00 |
| PRT-12702 - Solder-able Breadboard - Mini  ($2.95 ea.) | 2 | Sparkfun Electronics | $5.90 |
| TOL-11456 - Wall Charger - 5V USB (1A)  ($3.95 ea.) | 2 | Sparkfun Electronics | $7.90 |
| CAB-10215 - USB microB Cable - 6 Foot  ($4.95 ea.) | 2 | Sparkfun Electronics | $9.90 |
| OSH Park PCB printing | 2 | OSH Park | $12.40 |
| CAT5E Jack | 2 | Radioshack | $11.68 |
| 556 Dual Timer | 2 | Radioshack | $4.48 |
| Dual mini comparator | 1 | Radioshack | $4.94 |
| 1/8" mno y-cable | 1 | Radioshack | $4.94 |
| 16' speaker cable | 1 | Radioshack | $8.90 |
| 10' extension cord | 1 | Radioshack | $8.99 |
| ENT 357 | 1 | Radioshack | $5.39 |
| LM393NFS-ND ($0.41 each) | 2 | Digi-Key | $1.23 |
| LED | 2 | Digi-Key | -- |
| Resistor 100K Ohm | 2 | Digi-Key | -- |
| Resistor 120K Ohm | 2 | Digi-Key | -- |
| Resistor 82 Ohm | 2 | Digi-Key | -- |
| Resistor 820 Ohm | 2 | Digi-Key | -- |
| Resistor 10K Ohm | 6 | Digi-Key | -- |
| MPS750 PNP BJT | 2 | Digi-Key | -- |
| Capacitor 0.1uF | 2 | Digi-Key | -- |
| Capacitor 10nF | 2 | Digi-Key | -- |

**Table A1—Bill of Materials. The cost of most of the Digi-Key components is unknown because although Digi-Key is the supplier, a team member already owned the part and did not need to order it.**

**Appendix B—Main Door Source Code**

import RPi.GPIO as GPIO

import time

from datetime import datetime

from datetime import timedelta

from socket import \*

import thread

from threading import Thread

import paramiko

import os

import tweepy

import sys

start = datetime.now()

MEMBERIP = '' #Enter the IP address of the members door pi

DOORBELLPORT = 12346 #Port to send the doorbell command through

PORT = 12345 #Send in out info over this port

people = 0 #Number of people in the gym

sensor1 = 0 #flag for sensor 1

sensor2 = 0 #flag for sensor 2

SLEEPTIME = 0.2 #time to wait for the 2nd sensor to trip after the first one in seconds

BOUNCETIME = 300 #laser debounce time in ms

WAITTIME = 0.5 #Time to wait between people

BUFF = 1024

HOST = '' #Listen to all hosts

CARBON\_SERVER = '' #enter Carbon server IP in the quotes

CARBON\_PORT = #enter port number here

#enter the corresponding information from your Twitter application:

CONSUMER\_KEY = ''#keep the quotes, replace this with your consumer key

CONSUMER\_SECRET = ''#keep the quotes, replace this with your consumer secret key

ACCESS\_KEY = ''#keep the quotes, replace this with your access token

ACCESS\_SECRET = ''#keep the quotes, replace this with your access token secret

auth = tweepy.OAuthHandler(CONSUMER\_KEY, CONSUMER\_SECRET)

auth.set\_access\_token(ACCESS\_KEY, ACCESS\_SECRET)

api = tweepy.API(auth)

############################################################################

GPIO.setmode(GPIO.BCM)

GPIO.setup(23, GPIO.IN, pull\_up\_down = GPIO.PUD\_UP)

GPIO.setup(24, GPIO.IN, pull\_up\_down = GPIO.PUD\_UP)

############################################################################

def updatetwitter():

global people

while 1:

now = time.strftime("%c")

line = now + "\nNumber of people in Slo Op: " + str(people)

api.update\_status(line)

time.sleep(600) #update twitter every 10 minutes

############################################################################

def endscript():

try:

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(

paramiko.AutoAddPolicy())

ssh.connect(MEMBERIP, username="pi", password="sloopbomb")

ssh\_stdin, ssh\_stdout, ssh\_stderr = ssh.exec\_command('sudo pkill -9 -f membersdoor.py &')

except Exception,e:

print "ERROR: Couldn't send command to end memberdoor.py"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt send command to end memberdoor.py: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

return

############################################################################

def response(key):

return 'Server response: ' + key

def handler(clientsock,addr):

global people

while 1:

data = clientsock.recv(BUFF)

if not data: break

print repr(addr) + ' recv:' + repr(data)

clientsock.send(response(data))

print repr(addr) + ' sent:' + repr(response(data))

if "close" == data.rstrip(): break # type 'close' on client console to close connection from the server side

if data == "in":

people += 1

print "Someone came in the members door"

print "People in the gym: " + str(people)

if data == "out":

if people > 0:

people -= 1

print "Someone went out the members door"

print "People in the gym: " + str(people)

clientsock.close()

print addr, "- closed connection" #log on console

return

############################################################################

def sendcommand(string):

# SOCK\_STREAM == a TCP socket

sock = socket(AF\_INET, SOCK\_STREAM)

#sock.setblocking(0) # optional non-blocking

print "sending data => [%s]" % (string)

try:

sock.connect((MEMBERIP, DOORBELLPORT))

sock.send(string)

reply = sock.recv(16384) # limit reply to 16K

print "reply => \n [%s]" % (reply)

return reply

except Exception,e:

print "ERROR: Couldn't send Ring to membersdoor.py"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt send Ring to membersdoor.py: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

sock.close()

return "No Reply Received"

############################################################################

def runscript():

try:

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(

paramiko.AutoAddPolicy())

ssh.connect(MEMBERIP, username="pi", password="sloopbomb")

ssh\_stdin, ssh\_stdout, ssh\_stderr = ssh.exec\_command('sudo python /home/pi/membersdoor.py &')

except Exception,e:

print "ERROR: Couldn't send command to start membersdoor.py"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt send command to start membersdoor.py: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

return

############################################################################

def sensor1function(channel):

global sensor1

sensor1 = 1

def sensor2function(channel):

global sensor2

sensor2 = 1

############################################################################

def updategraphite():

while 1:

try:

## date and time representation

temp = 'people %d %d\n' % (people, int(time.time()))

print 'sending message:\n%s' % temp

carbonsock = socket()

carbonsock.connect((CARBON\_SERVER, CARBON\_PORT))

carbonsock.sendall(temp)

carbonsock.close()

except Exception,e:

print "ERROR: Couldn't send command to graphite"

#print "Trying to start graphite"

#os.system("sudo python /opt/graphite/bin/carbon-cache.py start &")

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt update graphite: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

time.sleep(300)

############################################################################

def lasers():

global people

global sensor1

global sensor2

global start

GPIO.add\_event\_detect(23, GPIO.FALLING, callback=sensor1function, bouncetime=BOUNCETIME)

GPIO.add\_event\_detect(24, GPIO.FALLING, callback=sensor2function, bouncetime=BOUNCETIME)

print("Digital Turnstile started")

while True:

#Sensor handling in/out detection

if sensor1 == 1:

time.sleep(SLEEPTIME)

if sensor2 == 1:

people += 1

print "Someone came in the main door"

print "People in the gym: " + str(people)

sensor1 = 0

sensor2 = 0

stop = datetime.now()

elapsed = stop - start

if elapsed > timedelta(seconds=30):

start = datetime.now()

try:

thread.start\_new\_thread(sendcommand, ('ringbell',))

except Exception,e:

print "ERROR: Couldn't send command to ring doorbell"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt send command to ring doorbell: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

time.sleep(WAITTIME)

sensor1 = 0

sensor2 = 0

elif sensor2 == 1:

time.sleep(SLEEPTIME)

if sensor1 == 1:

if people > 0:

people -= 1

print "Someone went out the main door"

print "People in the gym: " + str(people)

sensor1 = 0

sensor2 = 0

time.sleep(WAITTIME)

sensor1 = 0

sensor2 = 0

sensor1 = 0

sensor2 = 0

GPIO.cleanup()

############################################################################

if \_\_name\_\_=='\_\_main\_\_':

try:

endscript()

except Exception,e:

print "ERROR: Couldn't end membersdoor.py"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt end membersdoor.py: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

time.sleep(10)

try:

runscript()

except Exception,e:

print "ERROR: Couldn't run membersdoor.py"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt run membersdoor.py: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

try:

thread.start\_new\_thread(lasers, ())

except Exception,e:

print "ERROR: Couldn't Start Laser Thread"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt Start Laser Thread: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

try:

thread.start\_new\_thread(updategraphite, ())

except Exception,e:

print "ERROR: Couldn't Start updategraphite Thread"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt Start updategraphite Thread: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

try:

thread.start\_new\_thread(updatetwitter, ())

except Exception,e:

print "ERROR: Couldn't Start Twitter Thread"

print str(e)

with open('Door.log','a') as f:

now = time.strftime("%c")

temp = 'Couldnt Start Twitter Thread: %s %s\n' % (now, e)

f.write(temp)

f.close() # you can omit in most cases as the destructor will call if

ADDR = (HOST, PORT)

serversock = socket(AF\_INET, SOCK\_STREAM)

serversock.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1)

serversock.bind(ADDR)

serversock.listen(5)

while 1:

print 'waiting for connection... listening on port', PORT

clientsock, addr = serversock.accept()

print '...connected from:', addr

thread.start\_new\_thread(handler, (clientsock, addr))

############################################################################