BrightBot

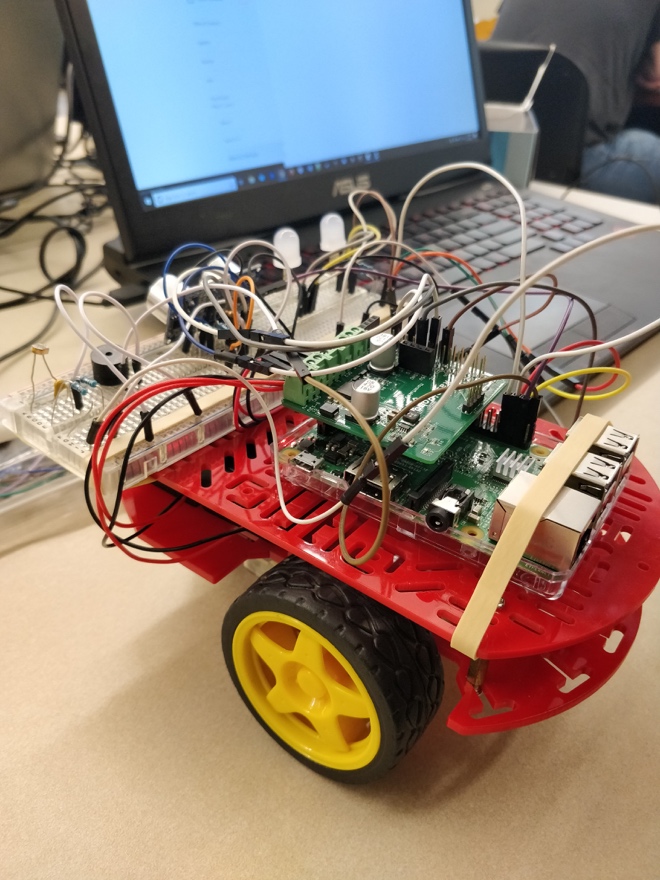
Group number 10

Zeshaan Haq, Jon Alabado, Vinet Patel

Abstract

Our project’s objective is to make a RaspiRobot rover that will essentially act as a moving flashlight. It will move on its own with attached RGB LEDs and change light depending on the brightness of the surrounding area. The rover will also be capable of detecting objects that get too close to it and will relay this information through a buzzer emitting a sound. The rover will then readjust its position to get away from the object and turn off the buzzer once this has been achieved. An additional feature for the rover would be a summary of the sensor information that would be transmitted over the internet via email. This email will contain information on the state that the rover is currently in such as which lights are turned on, if an object is currently in front of it, and the current brightness reading of the area. This summary will be sent over email every minute.

Finished Project:



Project Objectives:

|  |  |  |  |
| --- | --- | --- | --- |
| RGBs will change color based on the surrounding environment | The rover will be capable of detecting objects that get  too close to it and will relay this information through a buzzer emitting a sound. | The rover will  then readjust its position to get away from the object and turn off the buzzer once this has been  achieved. | Add a summary of the sensor information that  would be transmitted over the internet via email. This email will contain information on the state that the rover is currently in such as which lights are turned on, if an object is currently in front  of it, and the current brightness reading of the area. |

All of the mentioned objectives were met in the project.

Project Approach:

We implemented the project using in class materials such as the rover, rangefinder, photoresistor, and buzzer. The recipes that we used mainly came out of lab 2 which were recipes 10.1, 10.3, and 10.4. We also used 13.17 to get the rangefinder python code.

Firstly, we used a large breadboard that will fit our RGBs, buzzer, and rangefinder with all the connected components. After attaching our 4 RGB LEDs to the breadboard, the lightbulbs were weakened because the resistance would increase because we had a series circuit. So we solved the issue by essentially splitting the series in half. Because of this, we had two sets of three resistors connecting each pair of bulbs to ensure they aren’t getting way more resistance than they should. This resulted in higher brightness for our LEDS. But even after we did that, one or two of the lightbulbs was still not receiving enough power and brightness because of how the resistors were setup given the limited space we had on our breadboard. So one of the things we had to abandon was using two LEDs on each side of the breadboard and just using two on one side. We wanted to ensure the functionality was intact before worrying about the design and the number of LEDs would make the rover the most appealing.

Another issue we had was getting the rover to actually get the wheels moving and do what it’s supposed to do. We had to do a lot of fiddling with the code because for some reason when we ran for example, “rr. rorward”, the rover was changing directions and incorrectly moving its wheels.

One of the core functions we had to implement is called detectLightStength which takes the resistance as a parameter. This function uses the photoresistor to change the RGB colors depending on the resistance. We first had to test the average number of resistances we were getting in the classroom and we used 5000 and 2500 as baseline numbers to determine if it is dark or bright. We had to do a lot of gluing of code from previous labs namely the pot step.py and also gui-sliderRGB.

We used the rover interface functions to get to it to reverse and go forward after detecting an object and buzzing, and we changed the email setup code to send an email every minute with information about the current lighting of the environment.

Other than our breadboard set up and rover getting to work, we didn’t have any major challenges. We accomplished what we wanted to do, but of course if we had more time we could optimize the code and produce faster reaction times for the buzzer and LED to activate.

Conclusion:

To summarize, I think the final project was a fun experience. We were able to apply what we learned from our labs and piece them together to create BrightBot. Even though we accomplished what we wanted, I felt like there were a lot of options in terms of usefulness that we could have pursued. One setback was that most of the projects my group looked at required you to purchase expensive materials that the lab does not supply. Other than that time was another problem with this project. Other groups had the same issue where less time resulted in more bugs and less optimization.

My experience in CSC 299 was overall positive. I had no idea what a raspberry pi was before taking this class, so I’m glad I was exposed to the hardware side of things instead of just software like from my java and python courses at DePaul. I liked how every class was hands on and interactive, and everyone was willing to help each other.

Python Code:

#import statements

import RPi.GPIO as GPIO

import time, math

import smtplib

from rrb3 import \*

#gmail setup

GMAIL\_USER = 'jonmalabado@gmail.com'

GMAIL\_PASS = 'SHSrse11'

SMTP\_SERVER = 'smtp.gmail.com'

SMTP\_PORT = 587

#global variables

objNear = False

lighting = ""

#motor setup

GPIO.setmode(GPIO.BCM)

GPIO.setup(17, GPIO.OUT)

motorPin = GPIO.PWM(17, 500)

motorPin.start(100)

# RGB pin setup

GPIO.setmode(GPIO.BCM)

GPIO.setup(13, GPIO.OUT)

GPIO.setup(16, GPIO.OUT)

GPIO.setup(20, GPIO.OUT)

#color initalization

pwmRed = GPIO.PWM(13, 500)

pwmRed.start(0)

pwmGreen = GPIO.PWM(16, 500)

pwmGreen.start(20)

pwmBlue = GPIO.PWM(20, 500)

pwmBlue.start(0)

# buzzer pin setup

buzzer\_pin = 21

GPIO.setmode(GPIO.BCM)

GPIO.setup(21, GPIO.OUT)

#rangefinder pin setup

trigger\_pin = 18

echo\_pin = 23

GPIO.setmode(GPIO.BCM)

GPIO.setup(trigger\_pin, GPIO.OUT)

GPIO.setup(echo\_pin, GPIO.IN)

# capacitance and ohm setup

C = 0.33 # uF

R1 = 1000 # Ohms

# Pin a charges the capacitor through a fixed 1k resistor and the thermistor in series

# pin b discharges the capacitor through a fixed 1k resistor

a\_pin = 19

b\_pin = 26

#buzzer functions and variables

pitchGlobal = 750.0

def buzz(pitch, duration):

period = 1.0 / pitch

delay = period / 2

cycles = int(duration \* pitch)

for i in range(cycles):

GPIO.output(buzzer\_pin, True)

time.sleep(delay)

GPIO.output(buzzer\_pin, False)

time.sleep(delay)

# rangefinder functions

def send\_trigger\_pulse():

GPIO.output(trigger\_pin, True)

time.sleep(0.0001)

GPIO.output(trigger\_pin, False)

def wait\_for\_echo(value, timeout):

count = timeout

while GPIO.input(echo\_pin) != value and count > 0:

count = count - 1

def get\_distance():

send\_trigger\_pulse()

wait\_for\_echo(True, 10000)

start = time.time()

wait\_for\_echo(False, 10000)

finish = time.time()

pulse\_len = finish - start

distance\_cm = pulse\_len / 0.000058

return (distance\_cm)

# light sensor functions

# empty the capacitor ready to start filling it up

def discharge():

GPIO.setup(a\_pin, GPIO.IN)

GPIO.setup(b\_pin, GPIO.OUT)

GPIO.output(b\_pin, False)

time.sleep(0.1)

# return the time taken (uS) for the voltage on the capacitor to count as a digital input HIGH

# than means around 1.65V

def charge\_time():

GPIO.setup(b\_pin, GPIO.IN)

GPIO.setup(a\_pin, GPIO.OUT)

GPIO.output(a\_pin, True)

t1 = time.time()

while not GPIO.input(b\_pin):

pass

t2 = time.time()

return (t2 - t1) \* 1000000

# Take an analog reading as the time taken to charge after first discharging the capacitor

def analog\_read():

discharge()

t = charge\_time()

discharge()

return t

# Convert the time taken to charge the capacitor into a value of resistance

# To reduce errors, do it 100 times and take the average.

def read\_resistance():

n = 10

total = 0;

for i in range(1, n):

total = total + analog\_read()

t = total / float(n)

T = t \* 0.632 \* 3.3

r = (T / C) - R1

return r

#detects strength of nearby light and changes RGB colors

def detectLightStrength(resistance):

global lighting

if (resistance > 5000):

resistance = 5000

lighting = "Dark"

if (resistance < 0):

resistance = 0

lighting = "Bright"

if (resistance > 2500):

lighting = "Dark"

else:

lighting = "Bright"

colorCode = (resistance / 5000) \* 100

#print(resistance)

updateBlue(colorCode)

updateRed(100 - colorCode)

#changes blue value

def updateBlue(duty):

pwmBlue.ChangeDutyCycle(duty)

#changes red value

def updateRed(duty):

pwmRed.ChangeDutyCycle(duty)

#email function

def send\_email(recipient, subject, text):

smtpserver = smtplib.SMTP(SMTP\_SERVER, SMTP\_PORT)

smtpserver.ehlo()

smtpserver.starttls()

smtpserver.ehlo

smtpserver.login(GMAIL\_USER, GMAIL\_PASS)

header = 'To:' + recipient + '\n' + 'From: ' + GMAIL\_USER

header = header + '\n' + 'Subject:' + subject + '\n'

msg = header + '\n' + text + ' \n\n'

smtpserver.sendmail(GMAIL\_USER, recipient, msg)

smtpserver.close()

def emailTimer():

print("sending email")

outStr = 'Current Lighting: ' + lighting + '\nObject Near: ' + str(objNear)

send\_email('jon\_alabado2@comcast.net', 'Brightbot Report', outStr)

try:

global objNear

rr = RRB3(9, 6)

rr.forward()

start = time.time()

elapsed = 0

while True:

if (elapsed < 60):

elapsed = ((time.time() - start) % 65)

detectLightStrength(read\_resistance())

roboDistance = get\_distance()

print("RoboDistance: " + str(roboDistance))

print(lighting)

if (roboDistance < 11):

buzz(pitchGlobal, 1.0)

objNear = True

rr.stop()

rr.reverse(1, 1)

rr.forward()

else:

objNear = False

else:

emailTimer()

elapsed = 0

#time.sleep(1)

finally:

print("clean up")

GPIO.cleanup()