Smart City System Design

Gurinderpal Singh

Thompson Rivers University

Internet Of Things Class

November 22, 2023

Smart City System Design

Gurinderpal Singh

Thompson Rivers University

Internet Of Things Class

November 22, 2023

ABSTRACT

The goal for this project is to design a smart city system that is sustainable and energy efficient. A city that also prioritizes the safety of its residents. The project focuses on tackling certain problems faced in normal cities and implements them. Now, there are numerous possibilities that could take place in a smart city but this project focuses on the fire detection, smart street and traffic light, smart parking lots, automated garbage disposal and weather forecasting of the city.

This project tries to create a vision for the future and tries to create an image of what the future of a city might look like. There are multiple sensors like DHT11, ultrasonic, PIR sensors used to implement the functionalities of the smart city.

TABLE OF CONTENTS

SUMMARY

The project tackles problems of late fire responses, parking problem in cities, efficient traffic and street lights and garbage disposal problems faced by many businesses. With the help of IOT sensors this project designed systems to solve these problems.

In this report all the hardware, software components of the smart city and their workings have been explained in detail. The integration if all these components and being run as one is also described below. The output and the system setup is shown in the report as well alongside the results and findings of the project.

The data gathered has been uploaded to the dashboard for future references.

2. INTRODUCTION

2.1 SUBJECT

A smart city is a city that uses latest technology, data analysis and artificial intelligence to provide a better life to its residents. The infrastructure is so efficient and data driven that everyday tasks are made less hectic and easy to carry out. Everything around us evolved in last decade and we no longer even remember the tasks carried out as a routine before the automation around us. Smart city is a leap in to the future and it tries to paint the picture of the innovations coming next.

2.2 PURPOSE

The main driving forces behind the idea of a smart city are deteriorating environment, change in climate and high consumption of fossil fuels. The other most crucial aim is to better the life quality of the demographic. A place where security, sustainability, efficiency and better quality of life is promised is desired by everyone and smart city provides a gateway to that.

2.3 SCOPE

The scope of a smart city is limitless as almost every industry could be incorporated in it. There are tons of problems to be solved and the formerly existing solutions to be improved on around us in our cities. So, the smart city would be an entire upgrade of technology. While, there is a wide scope and many things to carry out, this smart city project tackles the problems of pedestrian safety, parking problem, delayed fire responses and some other inconveniences like garbage disposal and problems from bad weather.

3. METHODS, ASSUMPTIONS AND PROCEDURES

The smart city is a sustainable and efficient in terms of energy and time. It provides safe environment for all the people living in it. Through analysis of the current cities and the way they function and run, the smart city provides a solution.

There are six different operations carried out in this smart city design. The problem behind designing each functionality, the hardware used, the software implementations used in all of these operations have been described below in detail.

3.1 FLAME DETECTION SYSTEM

The current fire detection system in most cities is below par. Most of the actions taken by the fire departments are late responses. In BC, wildfires are a huge problem and in drier areas near the cities it has caused a huge problem. The fire detection system in the smart city tackles this problem. It does not work in scenario of fires in deep wild but near the city boundaries and inside the city it can make a huge difference.

The assumption made during the installation of this system is that the city would be well budgeted to install the fire detecting sensors around the city. The other assumption is that the sensor function properly with any technical problem and their maintenance is looked after because any irregularity in that could have fatal consequences.

a. EQUIPMENT

The main equipment required to design this system is a flame sensor and the other components needed are male to female jumper wires, a buzzer, a lighter to test and a raspberry Pi to provide a connection and gather data from the sensor.

The flame sensor would send an electronic signal as soon as a flame is detected inside its range of 3 feet. It has 4 pins that are VCC to provide voltage of 5V, A0 for analog output, D0 for digital output, and GND to provide ground to the connection. The sensor used in this smart city is a three-pin version with D0 as the only signal pin.

b. CONNECTION

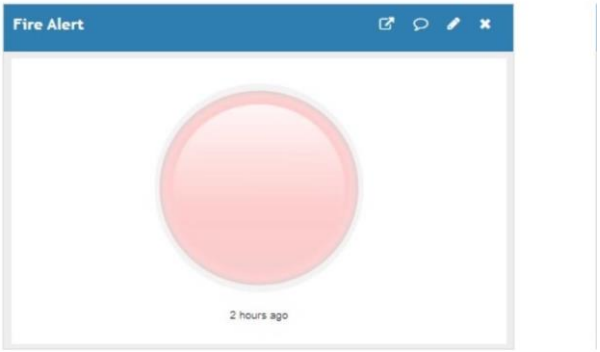
The VCC pin from the flame sensor is connected to the voltage pins while the A0 is connected to GPIO pin 21 and GND is given a ground connection using the male to female wires.

A buzzer connected is connected in the circuit to alert the people in case of a fire alongside the resistor of 110 ohms to protect the sensor from higher voltages. The positive pin is connected to GPIO 20 and the negative o=pin is connected to GND.

c. WORKING

In case of a fire, the flames are detected by the sensor and an electronic signal is collected from the sensor.

Upon getting the signal, a buzzer goes off in order to alert the people nearby and simultaneously a signal is sent to the dashboard of the fire department of the city.



The image above shows the alert sign that shows up on the fire departments system of the city to respond to the fire.

The benefit of this system is that it saves time as no one has to call the emergency numbers and even saving a little time in times of fire can be a huge success.

d. WORKING OF THE CODE

The code for the flame detection is:

```
import RPi.GPIO as GPIO
```

```
import time
import urllib.request as urllib2
from gpiozero import Buzzer
class flamedetector:
def __init__(self):
//GPIO Pin defination
Pin = 21
```

```
buzzer = Buzzer(20)
//Connecting to ThingSpeak
myAPI = 'YULQKFZEJGS2J2LL'
#readKey = 'K5EABMKHIXQI3IHG'
baseURL = 'https://api.thingspeak.com/update?api_key=%s' %myAPI
t_end = time.time() + 35*1 //How long the sensor senses
GPIO.setmode(GPIO.BCM)
GPIO.setup(Pin, GPIO.IN, pull_up_down = GPIO.PUD_UP)
//Flame detection
def detect_flame():
if GPIO.input(Pin) == 0:
        return True
else:
        return False
try:
        while (time.time() < t_end):
if detect_flame():
        print("Flame detected!!!")
        buzzer.on()
        conn = urllib2.urlopen(baseURL + '&field6=%s'% (2)) //Connecting to
        ThingSpeak
        #print(conn.read())
        conn.close()
else:
        print("No flames detected!")
        conn = urllib2.urlopen(baseURL + '&field6=%s' %(0)) //Connecting to
ThingSpeak
        #print(conn.read())
        conn.close()
        time.sleep(5)
        buzzer.off()
except KeyboardInterrupt:
        GPIO.cleanup()
detect_flame()
```

- All the need libraries are imported initially.

- Then a class “flamedetector” is defined and its construction is given all the methods and variables needed to run the program. Everything is placed in the constructor in order to run the file as soon as an object of class is created.

- Pin value is set up and a buzzer object is created. Then a connection is made with the Thingspeak using a URL and key to the dashboard’s channel.

- The while loop in this code is given a runtime of 35 seconds in order to detect a flame for longer time and it also removes the possibility of getting a fake reading as multiple readings are taken over a time.

- If the flame is detected, the buzzer is turned on and the data is sent to the Thingspeak dashboard. The sensor takes reading every 5 seconds to ensure a correct detection.

3.2 SMART PARKING SYSTEM

Parking is one of the biggest problems faced in the cities nowadays. Poor parking can be time consuming and inconvenient. The smart city solves this problem through creating smart parking lots outdoors and indoors using sensors that would let the people know if and where is a parking spot available. This will not only save time but will also be more fuel efficient.

The assumption in this functionality is that the sensor is never blocked by someone with an object knowingly or unknowingly. The other assumption is that people don't stand in the parking spaces themselves for a long period of time.

a. EQUIPMENT

The hardware components needed to design a smart parking system are ultrasonic sensor, female to male wires, and a pi to gather data.

The ultrasonic sensor(HC-SR04) consists of 4pins: VCC pin to get voltage, Trigger to emit ultrasonic signals, Echo to collect the signal and GND to ground the connection.

b. CONNECTION

The VCC is connected to the voltage pin on the pi board while the trigger and echo are connected to the GPIO pins 23 and 18 respectively of the board using the wires. A resistor is connected in the circuit to protect the sensor from high voltage.

c. WORKING

This system is a simple application of an ultrasonic sensor where the sensor sends the sound wave using it trigger pin and the wave goes hits the obstruction and returns to the sensor where it is caught by the echo pin. Since, the speed of the sound wave is known and using

time taken by the wave to return the distance of obstruction is found out. This principle is used to design a smart parking system.

The obstruction here is a car and based on the distance of the car from the sensor, it is easier to decide whether a spot is empty or not.

The decision made from the data gathered from the sensor is gathered by the pie and then uploaded on the dashboard which in practice would be sent to an application or a website to let people know of the available parking lot's location.

d. SOFTWARE WORKING

The code to implement smart parking is:

```
from Adafruit_IO import *
import RPi.GPIO as GPIO
from gpiozero import *
from time import *
import time
import urllib.request as urllib2
import ssl

class parking:
def __init__(self):

GPIO.setmode(GPIO.BCM)
GPIO_TRIGGER= 23
GPIO_ECHO=18
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

myAPI = 'YULQKFZEJGS2J2LL'
#readKey = 'K5EABMKHIXQI3IHG'
baseURL = 'https://api.thingspeak.com/update?api_key=%s' %myAPI

#function for counting distance

def distance():
#sending sound from ultra sonic sensor
GPIO.output(GPIO_TRIGGER, True)
time.sleep(0.00001)
t_end = time.time() + 40*1
#stopping from sending sound from ultra sonic sensor
GPIO.output(GPIO_TRIGGER, False)
StartTime=time.time()
StopTime= time.time()
```

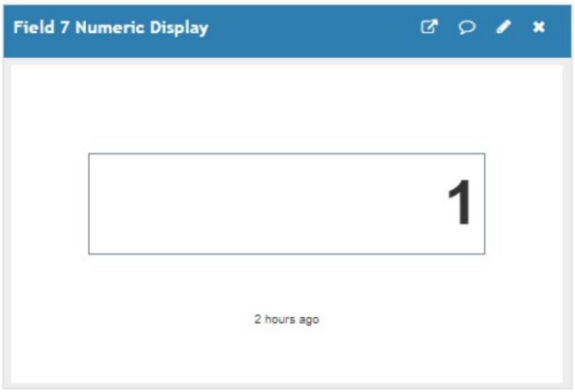
```
#getting start time

while GPIO.input(GPIO_ECHO)==0:
StartTime=time.time()
#getting end time
while GPIO.input(GPIO_ECHO)==1:
StopTime=time.time()
#calculating elapsed time
TimeElapsed=(StopTime-StartTime)
#claculating distance with elapsed time
distance= (TimeElapsed*34300)/2
print("Findind a parking spot")

try:
while (time.time() < t_end):
spot_available = False
dist = (distance)/100
#print(dist)
if (dist<1.5):
print("Spot unavailable")
print("Rechecking!!")
conn = urllib2.urlopen(baseURL + '&field7=%s'% (0))
#print(conn.read())
conn.close()
if(not spot_available):
time.sleep(3)
#except Exception as e:
#print(e)
else:
print("Spot available.")
conn = urllib2.urlopen(baseURL + '&field7=%s'% (1))
# print(conn.read())
conn.close()
time.sleep(7)
except KeyboardInterrupt:
        print("Measurement stop by user")
GPIO.cleanup()
distance()
#SP = parking()
```

- All the needed libraries like RPi.GPIO and ADAfruit_IO are imported to access their attributes and functions.
- A parking class is created and the constructor is given all the variables and method to run the program as soon as the object is created.
- The pin numbers are assigned to the pins and using GPIO.setup, the pins are used to get input and output.

- The key and URL is used to send the data to the dashboard.
- The function distance( ) is used to find the distance from the ultrasonic sensor. The while loop runs for 40 seconds. And conditions are set based on the echo that takes the input if spot is available or not.
- If distance is less than 1.5 metres, then the spot is made unavailable and then a recheck is done just in case it was being blocked by something for a while. And if distance is greater than 1.5 then the spot is available.
- Based on the result, the dashboard is updated with the available or not available parking spot.



3.3 Automatic garbage disposal

This system is designed for the commercial businesses where they won't have to worry about the disposal of the garbage. The system uses an ultrasonic sensor to check for the garbage if it is full or not. It focuses on the goal of providing high quality of life for residents in the smart city by reducing the number of things they'd have to worry about on daily purposes.

The assumption made here is that the system is connected to a network wired or wireless and is connected for entire day.

a. EQUIPMENT

A garbage can is needed to test the working of the system, female to male wires, ultrasonic sensor, and a pi are the only things needed to implement this design.

b. CONNECTION

The ultrasonic sensor is attached to the inside of the lid 0f the garbage can facing the garbage. The VCC, trigger, echo and GND pins are connected to the pi board where the data is sent by the sensor. The VCC is connected to a voltage pin, the trig is connected to GPIO pin 26 and the echo is connected to GPIO pin number 22.

c. WORKING

The ultrasonic sensor sends and catches the sound wave that bounces off of the garbage. Based on that the distance of the garbage from the lid of the can is calculated. When the garbage gets full, the sensor sends the data to the pi and email is sent to the sanitation department that the garbage is full and it gets picked up by them.

d. WORKING OF SOFTWARE

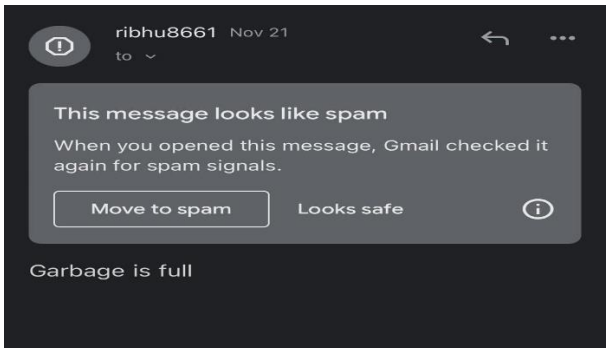
The working of the code is similar to that of the smart parking system as the same sensor is used and it works on the same principles.

```
import RPi.GPIO as GPIO
from gpiozero import *
from time import *
import time
import smtplib
import ssl

class garbage:
def __init__(self):
smtp_p = 465 // Set to 465, the port for SSL-encrypted SMTP.
smt_ser = "smtp.gmail.com" // Set to "smtp.gmail.com," the SMTP server for Gmail.
email_from = "ribhu8661@gmail.com" // Set to the sender's Gmail address.
email_to = "ribhubhartiya@gmail.com" // Set to the recipient's email address.
passkey = "bchrvgizaotqaqor" // Set to the sender's Gmail password
message = "Garbage is full"
simple_email_context = ssl.create_default_context()
GPIO.setmode(GPIO.BCM)
GPIO_TRIGGER= 26
GPIO_ECHO=22
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
#function for counting distance
def distance():
#sending sound from ultra sonic sensor
GPIO.output(GPIO_TRIGGER, True)
time.sleep(0.00001)
t_end = time.time() + 10*1
#stopping from sending sound from ultra sonic sensor
GPIO.output(GPIO_TRIGGER, False)
StartTime=time.time()
StopTime= time.time()
```

```
#getting start time
while GPIO.input(GPIO_ECHO)==0:
StartTime=time.time()
#getting end time
while GPIO.input(GPIO_ECHO)==1:
StopTime=time.time()
#calculating elapsed time
TimeElapsed=(StopTime-StartTime)
#claculating distance with elapsed time
distance= (TimeElapsed*34300)/2
while (time.time() < t_end):
dist = (distance)/100
#print(dist)
if (dist<50):
try:
#print("Connecting..")
server= smtplib.SMTP_SSL(smt_ser, smtp_p,context =simple_email_context)
server.login(email_from, passkey)
#print("Connection done")
server.sendmail(email_from,email_to,message)
print("Email sent for garbage collection.")
time.sleep(10)
except Exception as e:
print(e)
distance()
```

- RPi.GPIO, time, smtplib libraries are imported in the code.
- A class named garbage is made and the construction is initialized. And, the pin numbers are given.
- A from and to email is given with a unique passkey of the from email is set up to send an email when garbage gets full.
- Using the GPIO.setmode the pins are set to get and give signals. Based on when the echo gets the signal back, distance of the lid from garbage is calculated.
- When the distance gets less than 50cm (for testing purpose), the email is sent to sanitation department.
- Exception handling is done at the end.



3.4 Humidity and Temperature reporting and forecasting

The smart city's weather forecasting provides a real time data and also provides the predictions for a few minutes in the future. A DHT11 sensor is used to collect the data and send it to the city dashboard. The data gathered by this system helps the residents of the city to plan their days and make better decisions over the day and hence more convenience.

The weather could be a hard thing to predict and assuming it does not change drastically all of a sudden, the forecasting can be a great tool as it provides real time data.

a. EQUIPMENT

The equipment needed for this setup is a DHT11 sensor, wires, resistors and a pi. The data used in this smart city implementation is generated from the AI of the chat gpt API. The DH11 sensor used is a 3-pin sensor. The pins consist of VCC, data and GND pin.

b. CONNECTION

The dht11 VCC pin is connected to the voltage pin of the pi. The GND pin is connected to the ground and the data pin that communicates the values of temperature and humidity to the system is connected at GPIO 4. Again, there is a resistor to protect the sensor from excessive voltage.

c. WORKING

The sensor detects the reading of temperature and humidity and it is stored in the pi from where it is communicated to the dashboard. The AI generated data is used to learn the patterns in the weather and make a better forecasting decision.

d. WORKING OF THE CODE

The code of the DHT11 Sensor is:

```
import RPi.GPIO as GPIO
```

```
import sys
import urllib.request as urllib2
import Adafruit_DHT
import time
class humidity:
def __init__(self):
#api key
myAPI = 'YULQKFZEJGS2J2LL'
#readKey = 'K5EABMKHIXQI3IHG'
baseURL = 'https://api.thingspeak.com/update?api_key=%s' %myAPI
t_end = time.time() + 20*1
dht_sensor= Adafruit_DHT.DHT11
dht_pin=4
def dht11_data():
humidity, temperature =Adafruit_DHT.read_retry(dht_sensor, dht_pin)
return humidity, temperature
while (time.time() < t_end):
try:
humidity, temperature = dht11_data()
if isinstance(humidity, float) and isinstance(temperature, float):
humidity = '%.2f' % humidity
temperature = '%.2f' % temperature
conn = urllib2.urlopen(baseURL + '&field1=%s&field2=%s' % (temperature, humidity))
print(conn.read())
conn.close()
else:
print('Running...')
except Exception as E:
print(E)
dht11_data()
c=humidity()
```

- All the libraries needed like RPi.GPIO, sys, Adafruit_DHT and urllib.request are imported. The urllib library allows us to work with the URLs which is needed to fetch data for forecasting the weather.
- A class named humidity is created and its constructor is created.
- Key, pins, and the sensor are given their respective values.
- A dht11_data function is created and while loop is executed for 20sec in it to get the values of the temperature and humidity levels in the environment.
- Then the data is uploaded to the Thingspeak dashboard.

The code for the data from AI is:

```
import requests
from dotenv import load_dotenv
import os
class dataAI:
```

```
def __init__(self):
load_dotenv()
myThingSpeakAPI = 'YULQKFZEJGS2J2LL'
baseURL = 'https://api.thingspeak.com/update?api_key=%s' %myThingSpeakAPI
OPENAI_API_KEY = "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"
# def fetch():
#
# THINGSPEAK_CHANNEL_API_URL =
'https://api.thingspeak.com/channels/2350013/.json?api_key=K5EABMKHIXQI3IHG&results=2'
#
# try:
# response = requests.get(THINGSPEAK_CHANNEL_API_URL)
# response.raise_for_status()
def fetch_weather_data():
THINGSPEAK_CHANNEL_API_URL =
"https://api.thingspeak.com/channels/2350013/feeds.json?api_key=K5EABMKHIXQI3IHG&results
=2"
```

```
response = requests.get(THINGSPEAK_CHANNEL_API_URL)
print(response.status_code)
if response.status_code == 200:
feeds = response.json().get('feeds',[])
if feeds:
return feeds
else:
print('no data in channel')
else:
print("Failed to retrieve data form thingspeak")
return None
def analyze_data_with_openai(data):
headers ={
'Content-Type' : 'application/json',
'Authorization' : f'Bearer {OPENAI_API_KEY}'
}
prompt = f'I have weather of room data: {data}. give me numeri value for temperature and humidity
according to the values of field1 and field2'
data = {
'prompt' : prompt,
'max_tokens' : 150
}
# print(data)
response = requests.post('https://api.openai.com/v1/engines/text-davinci-003/completions',
headers=headers, json=data)
if response.status_code == 200:
text = response.json().get('choices', [{}])[0].get('text', '').strip()
return text
else:
print('Failed to analyze response from openai')
return None
def extract_numbers(data_str):
numbers = []
current_num = ''
for char in data_str:
if char.isdigit() or char=='.':
current_num+=char
elif current_num != '':
numbers.append(float(current_num))
current_num=''
```

```
if current_num != '':
numbers.append(float(current_num))
return numbers
def convert_to_integers(number_list):
return [int(num) for num in number_list]
def rundata():
if not OPENAI_API_KEY:
print('yes')
weather_data = fetch_weather_data()
# print(weather_data)
if weather_data:
analysis = analyze_data_with_openai(weather_data)
if analysis:
print("Data analysis: ", analysis)
else:
print("Failed to analyze weather data.")
else:
print("No weather data to analyze.")
temp = extract_numbers(analysis)
temp2 = convert_to_integers(temp)
#print(temp2[0], temp2[1])
update_thingspeak(temp2[0], temp2[1])
def update_thingspeak(field1_value,field2_value):
thingspeak_url = baseURL
data = {
'api_key':myThingSpeakAPI,
'field3': field1_value,
'field4': field2_value
}
response = requests.post(thingspeak_url, data=data)
if response.status_code == 200:
print('Updated forecast is available')
else:
print('Failed to send data', respose.status_code)
rundata()
```

- All the needed libraries are imported.
- A class named dataAI is created and the constructor is defined.
- OPENAI API key and API key from thingsboard is set up to make a connection between them.
- The data is fetched from the thingspeak channel and feeds data is taken as well.
- The AI then analyses the data and data values are extracted from it.
- Values are formatted and then updated and the data is then updated on the dashboard.
- At the end, the function is called inside the constructor to start the process as soon as an object of the class is made.

3.5 SMART TRAFFIC LIGHTS

The smart city has a smart traffic lights system where these lights are only installed in the school zones and the time is set up based on the school timings. The goal behind these lights is that the pedestrian would not have to press a button and the walking signal will come up automatically as soon as they are near. It has been designed keeping the safety of the pedestrians in mind.

The assumption made during this is that the school timings will not be varying a lot and the conditions like off school days due to certain uncertain events are not taken in to account.

a. EQUIPMENT

The PIR sensor, the wires, the LED's to implement traffic lights are needed for this setup. The PIR sensor consists of 3 pins which are VCC for voltage , GND for ground and OUT to Motion only when its from a source with a body heat.

b. CONNECTION

The VCC of the PIR is connected tot the voltage pin of the pi. The GND is connected to ground and the OUT is connected at the GPIO pin 14.
LED anodes are connected at GPIO 25,7,8 and the negatives are connected to the ground in order to make a connection to implement traffic lights.

c. WORKING

As soon as the sensor detects the motion near it, it turns the traffic signal red making the traffic stop and the pedestrians could easily cross the road. The PIR sensor only detects the motion from the sources with some body heat like animals, humans etc.

d. WORKING OF THE CODE

The code to implement the smart traffic lights is:

```
import RPi.GPIO as gpio
import time
from gpiozero import MotionSensor as mws
from datetime import datetime as t
import pytz
from gpiozero import TrafficLights
class trafficLights:
def __init__(self):
easter= pytz.timezone('Canada/Pacific') //Fetch TimeZone
currtime=t.now(easter) //Current Time
after = (currtime.hour>=14) //Used to activate the sensor in school hours
t_end = time.time() + 10*1 //How long the sensor senses
gpio.setmode(gpio.BCM)
light = TrafficLights(25,8,7)
//set GPIO Pins
pirpin=14
gpio.setwarnings(False)
gpio.setup(25,gpio.OUT)
```

```
gpio.setup(8,gpio.OUT)
gpio.setup(7,gpio.OUT)
gpio.setup(pirpin, gpio.IN)
//Traffic Lights function
def lights(pirpin):
print("Pedestrian detected")
print("Lights on")
light.green.off()
light.amber.on()
time.sleep(1)
light.amber.off()
light.red.on()
time.sleep(3)
light.red.off()
light.green.on()
```

```
print("Motion sensored alarmed")
print("ready")
try:
if (after):
light.green.on()
gpio.add_event_detect(pirpin,gpio.RISING, callback=lights)
light.green.on()
while (time.time() < t_end):
time.sleep(5)
except KeyboardInterrupt:
print("Quit")
gpio.cleanup()
lights(pirpin)
        #tl = trafficLights()
```

- All the libraries needed are imported first.
- A class named trafficLights is created and the constructor is defined.
- Since, the system is based to run during school timings, the time is fetched and if the time is school time the traffic lights run.
- The pin of the sensor is setup and using setmode the sensors are made ready to get the signal.
- A function lights () is run whenever a pedestrian is detected and the other LEd's are made to sleep while the red LED is turned on so that the pedestrians could cross the rode.
- The while loop is run for only 10 sec for testing which means after 10 sec if no motion is detected the lights can turn red and traffic starts again.
- At the end the method lights is called to run the code.

3.6 SMART STREET LIGHTS

The smart street light system will be turned on during the night and these lights use a PIR sensor so that the light will only turn on when there is someone walking down the street. This makes them more efficient to use and provide the smart city residents with more safety as there will be no dark corners in the city.

The assumption made in this system is that there are no bad lighting conditions during the day due to bad weather. In which case, the lights would still be off as the time condition will not be satisfied.

The street light will be connected alternatively in such a way that the second lamp is placed right where the range of first ends and so on.

a. EQUIPMENT

The Pir sensor to detect motion and an LED to implement a street light are the only two things that are needed to create this system. The other thing needed is wires and a pi to power the sensor and the LED.

b. CONNECTION

The VCC pin of the PIR sensor is connected to the voltage pin of the pi, GND is given ground pin of the pi whereas the OUT pin is connected to the GPIO 15.

The LED anode is connected to the GPIO 2 and the cathode is connected to the ground.

c. WORKING

Whenever the sensor detects some motion the LED is turned on until the notion is no longer detected and hence keep the pedestrians safe.

c. WORKING OF THE CODE

```
import RPi.GPIO as gpio
import time
from datetime import datetime as t
import pytz
class GPTmotion:
def __init__(self):
easter= pytz.timezone('Canada/Pacific') // Fetch TimeZone
currtime=t.now(easter) // Current Time
after = (currtime.hour>=17) //Used to activate the sensor after hours
t_end = time.time() + 10*1; //How long the sensor senses
//set GPIO pins
gpio.setmode(gpio.BCM)
```

```
led= 2
pirpin=15
gpio.setwarnings(False)
gpio.setup(led,gpio.OUT)
gpio.setup(pirpin, gpio.IN)
//Function for LED functions
def lights(pirpin):
print("Motion detected")
print("Lights on")
gpio.output(led,gpio.HIGH)
time.sleep(3)
print("Light off")
gpio.output(led,gpio.LOW)
print("Motion sensored alarmed")
print("ready")
try:
if (after):
gpio.add_event_detect(pirpin,gpio.RISING, callback=lights)
while(time.time()< t_end):
#while 1:
time.sleep(1)
except KeyboardInterrupt:
print("Quit")
gpio.cleanup()
lights(pirpin)
```

- All the libraries needed are imported
- A class named lights in created and a constructor is initialized.
- The time zone and time are taken so that the lights can be turned in at night.
- The sensor is set to take the input and read data using GPIO.setmode().
- All the pin numbers and the Led are initialized.
- A function lights is created which turns the LED on as soon as some motion is detected.
- For testing purpose the loop is only run for 10 seconds and testing for motion takes place again.
- The function is called at the end for the code to run and sensor to gather data.

3.7 INTEGRATION

Since all the function are separate and run separately, there has to be an integration where they could run together as one system.

To integrate, multiprocessing library of python has been used. Each of the individual sensor files have been imported in to this one file.

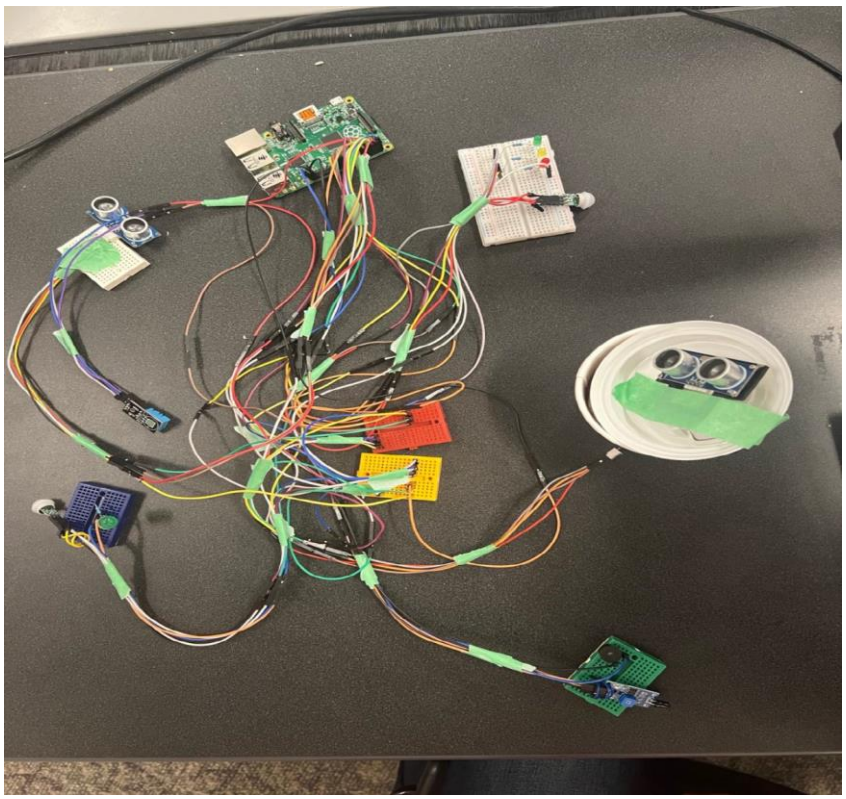
Each sensor has been used to create a process. And it is done through passing the sensor object as a Process. Since the code for every sensor is in constructor, the files run as soon the Operating system runs the Process.

The code file is:

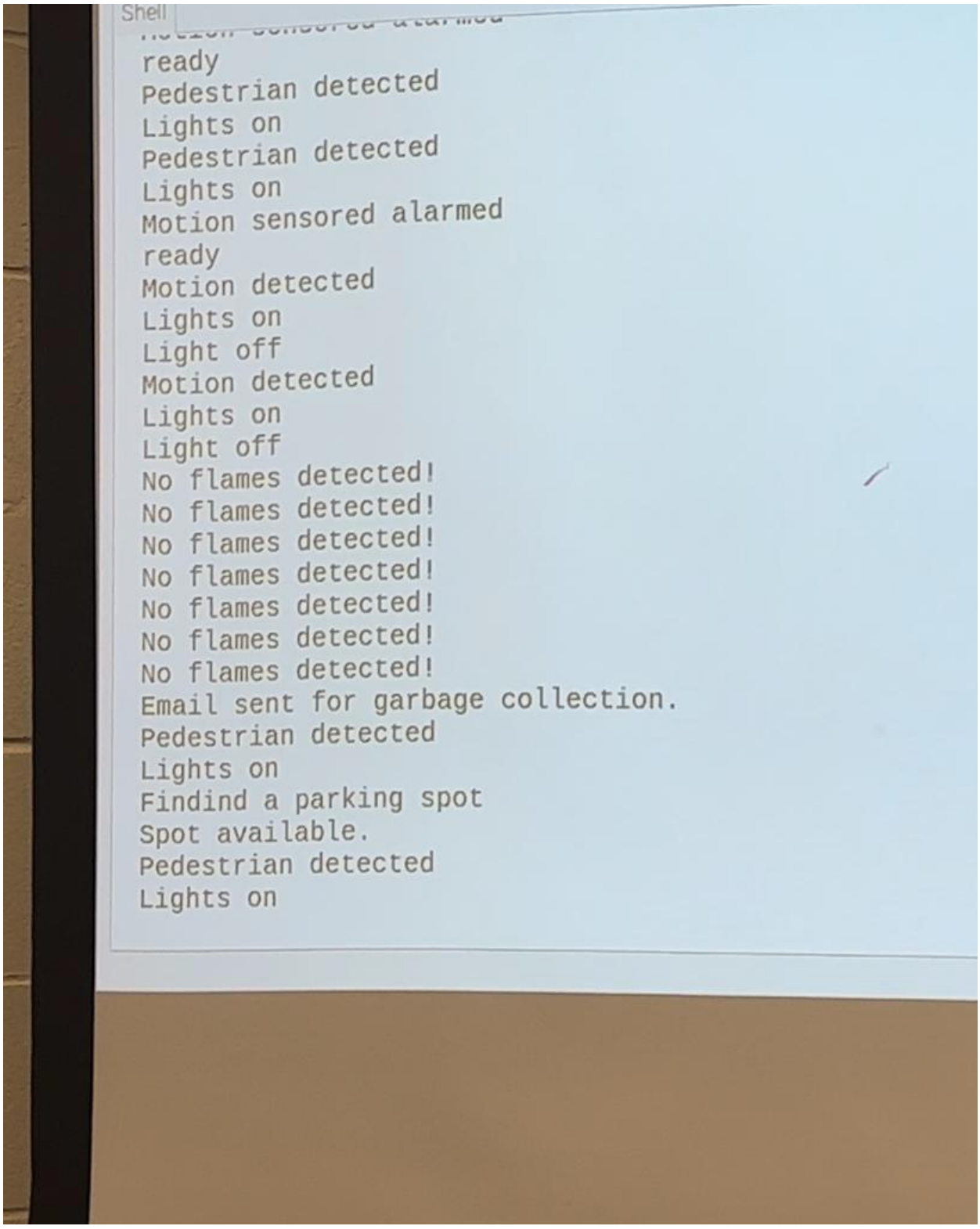
```
from humidity import *
from multiprocessing import Process
from ultra_garbage import *
from firedetector import *
from GPTmotion import *
from Parking_test import *
from trafficLights import *
from readingDataFromOpenAi import *
#import tkinter as tk
#import subprocess
#ht = humidity()
tl = trafficLights()
gm = GPTmotion()
fs = flamedetector()
ug = garbage()
pk = parking()
#data = dataAI()
if __name__ == "__main__":
#def on_button_click():
while(1):
# process1 = Process(target = humidity)
process2 = Process(target=trafficLights)
```

```
process3 = Process(target = GPTmotion)
process4 = Process(target = flamedetector)
process5 = Process(target = garbage)
process6 = Process(target = parking)
# process7 = Process(target = dataAI)
#Defining the Button for running all these processes
# """root = tk.Tk()
# root.title("Tkinter button click")
#
# button = tk.Button(root, text = "run External Script", command=on_button_click)
#
# button.pack()
# root.mainloop()"""
```

3.8 SYSTEM SETUP



3.9 OUTPUT


Shell
ready
Pedestrian detected
Lights on
Pedestrian detected
Lights on
Motion sensored alarmed
ready
Motion detected
Lights on
Light off
Motion detected
Lights on
Light off
No flames detected!
No flames detected!
No flames detected!
No flames detected!
No flames detected!
No flames detected!
No flames detected!
Email sent for garbage collection.
Pedestrian detected
Lights on
Findind a parking spot
Spot available.
Pedestrian detected
Lights on

4. RESULTS AND DISCUSSIONS

The results found through testing of this project tell that through use of sensors and IOT many problems around us can be solved. Moreover, the data that is collected during this project's is posted on the dashboard and can be used to build on this system.
There are numerous things and features that could be added to this project.

5. CONCLUSION

The conclusion from this project is that there are several solutions available for automating and improving on the things which currently lack in our cities. It might take some investment in the beginning phases of a smart city but there are a lot of savings and it promotes sustainability which are more beneficial in the long run. Smart Cities are the future and this project show the glimpse of the future.

6. REFERENCES

- Waheb A. Jabbar, Chong Wen Wei, Nur Atiqah Ainna M. Azmi, Nur Aiman Haironnazli, 2021, An IoT Raspberry Pi-based parking management system for smart campus, https://www.sciencedirect.com/science/article/abs/pii/S2542660521000317

- Outrosdiasvirao. (n.d.). Smart Traffic Signal Light. Hackster.io. https://www.hackster.io/outrosdiasvirao/smart-traffic-signal-light-450936

7. ABRREVIATIONS AND SYMBOLS

- pi: Raspberry Pi
- sec: Seconds
- hrs: Hours
- min: Minutes
- temp: Temperature
- hum: Humidity
- IoT: Internet of Things
- API: Application Programming Interface

- PIR: Passive Infrared
- LED: Light Emitting Diode
- IoT: Internet of Things
- API: Application Programming Interface
- URL: Uniform Resource Locator

- SSL: Secure Sockets Layer
- DHT: Digital Humidity and Temperature
- GPIO: General Purpose Input/Output