VTE College Central

Devices and Applications

Mini Project

Smart Car Parking system

Student's Name: Abirami Baskaran

Class: CE2004U

Date: 3rd February 2022

Group No: 2

ITE College Central

Content Page

A) Introduction

There are about 12,000 car parks in Singapore providing about 1.4 million parking lots. That being said, most of these car parks are located outdoors, where parked cars are exposed to problems such as heat and sunlight which can wear down the dashboard, steering wheel, and upholstery in a car. In addition, parking a car streetside poses a greater risk to the vehicle's body due to falling trees or children playing.

To create a safer environment for drivers to park their car, we have created a car park with an automatic roller shutter that operates according to weather conditions.

Another problem drivers usually face would be finding a parking lot. Though some carparks are equipped with a system that shows the number of vacant lots, it is still a challenge for drivers to spot out the empty lot.

Therefore, we have implemented a system which allows drivers to easily identify the location of free lots.

Carparks at certain locations may be expensive. With the stress of paying ERP and road tax being enough, drivers also have to worry about finding cheap parking places.

This car park does not only charge a fixed low amount but also provide an option of parking free for members.

The basic elements of this car park functions are automated and operates electronically. This reduces human intervention to a minimum and avoid constant input from an operator.

B) Objectives & Scenario Description

Our Objective:

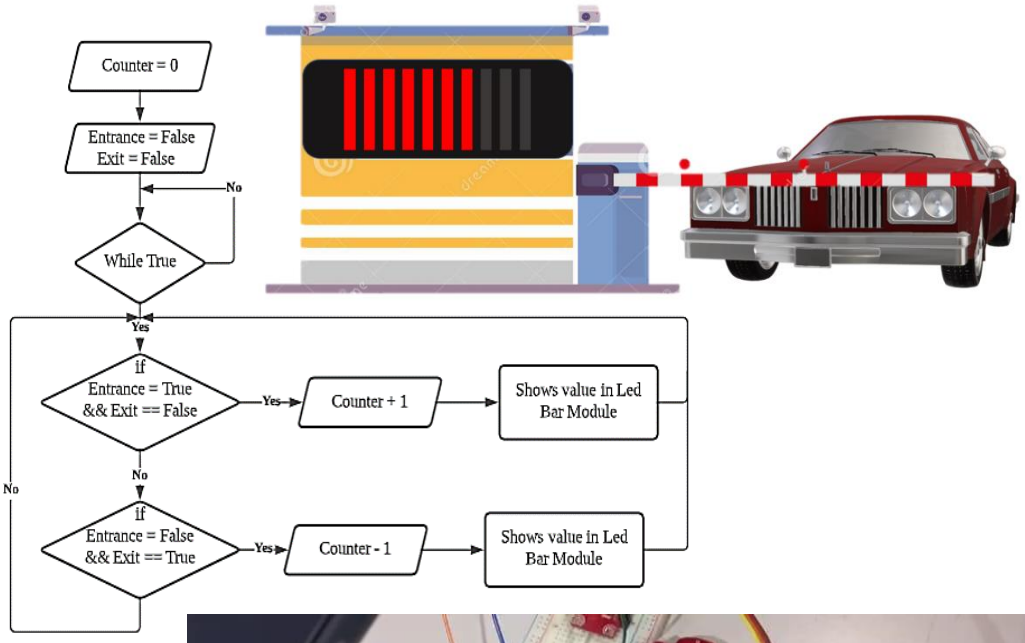
Our objective is to create a parking garage that is
more technologically advanced,
more convenient for drivers,
a safer environment for vehicles and
a better choice compared to other car parks in Singapore.

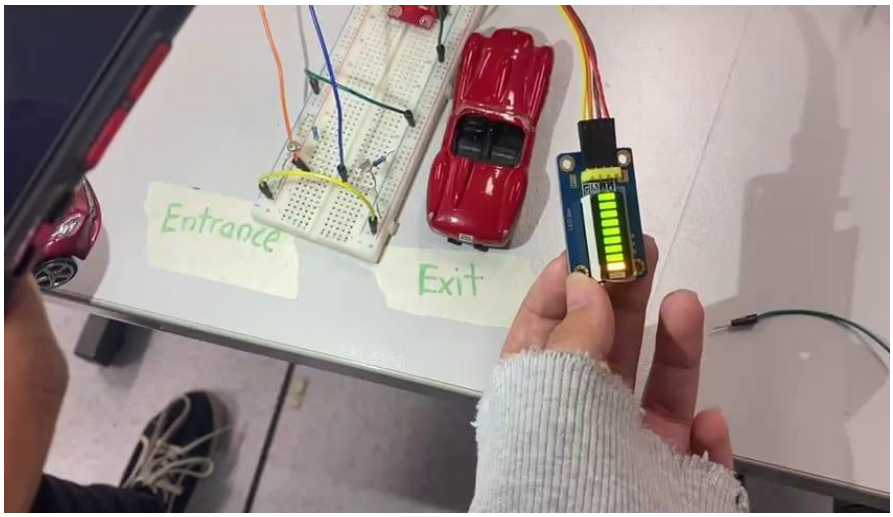
We hope that with the added facilities, the car park will be truly beneficial to drivers not just in terms of convenience but also financially.

Scenario Description:

Car Update:

When a person drives towards the entrance, he/she can check the number of cars currently inside the location. An LED Bar Module is fixed on the side and will count up and down every time a car passes the entrance and exit respectively. This could avoid overcrowding as drivers will avoid entering the parking garage if they are aware of the crowd situation inside.

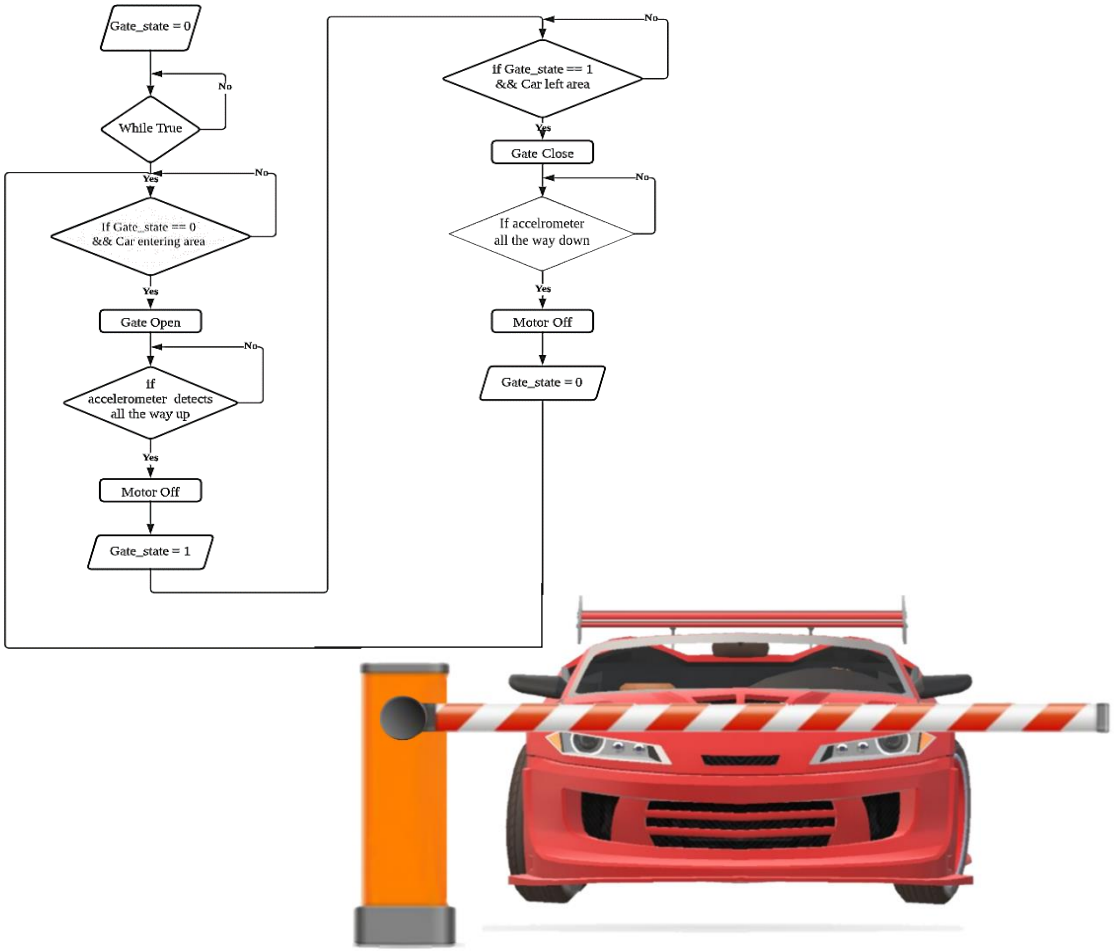


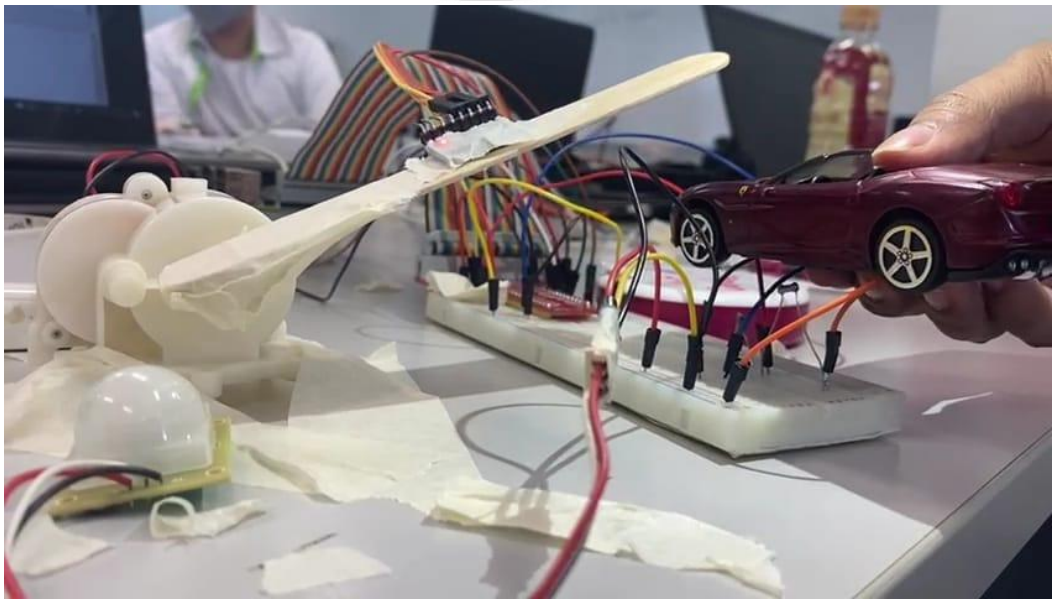


Gantry *(Entrance & Exit)*:

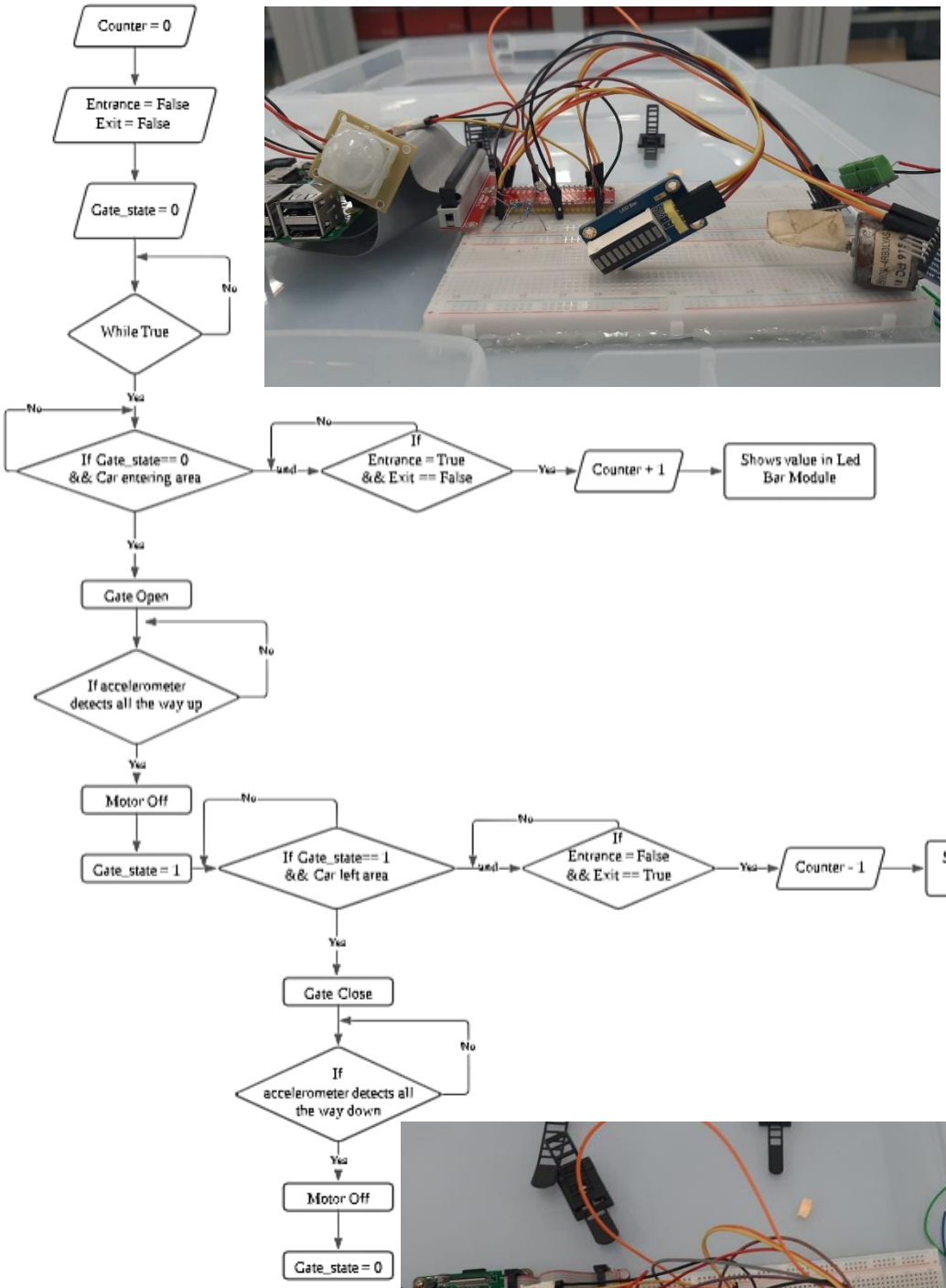
When a person drives towards the entrance, the barrier gate will operate and open after detecting motion and the shadow of the car.

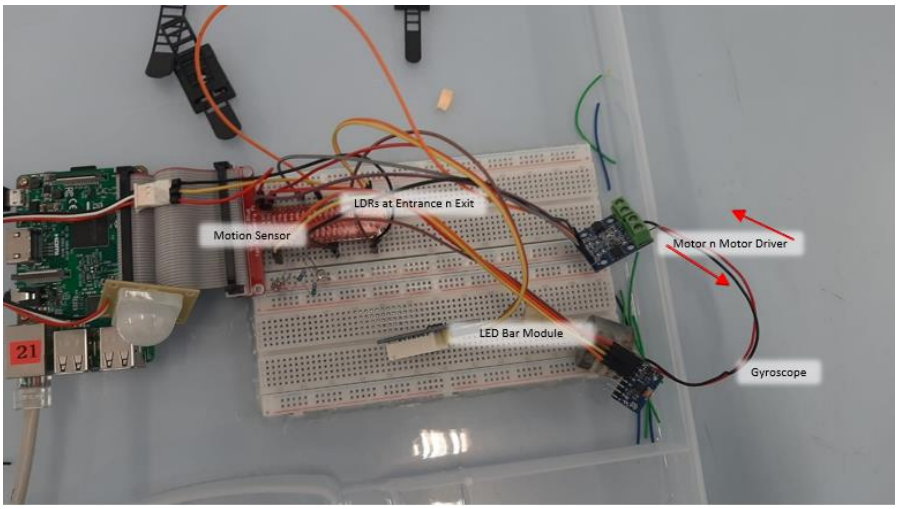
Once the car passes, and shadow is no longer detected, the gate closes. The accelerometer set at the barrier gate checks if the gate is up or down and will also direct the motor operating the gate.





Car Update & Gantry combined**:**

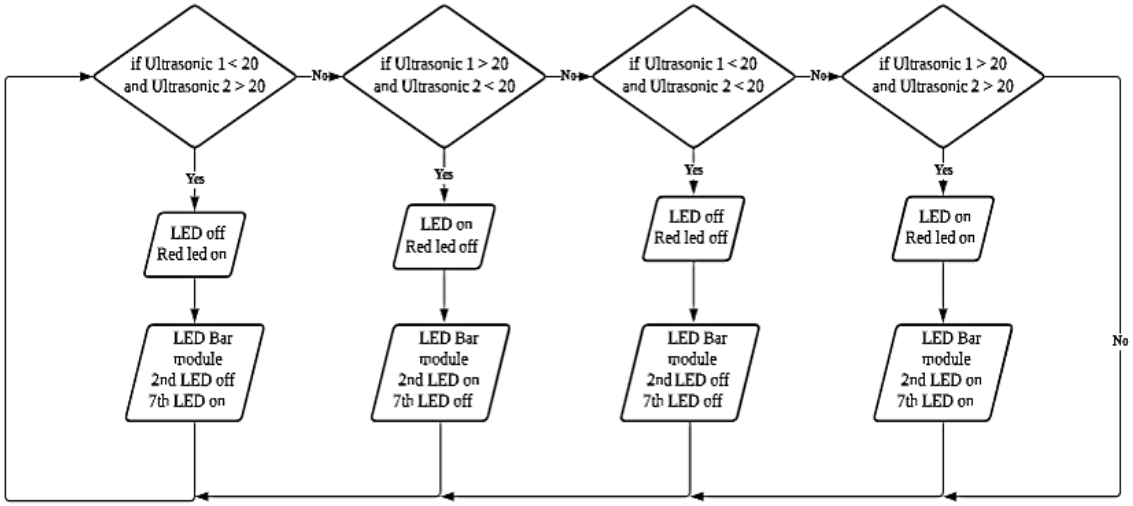


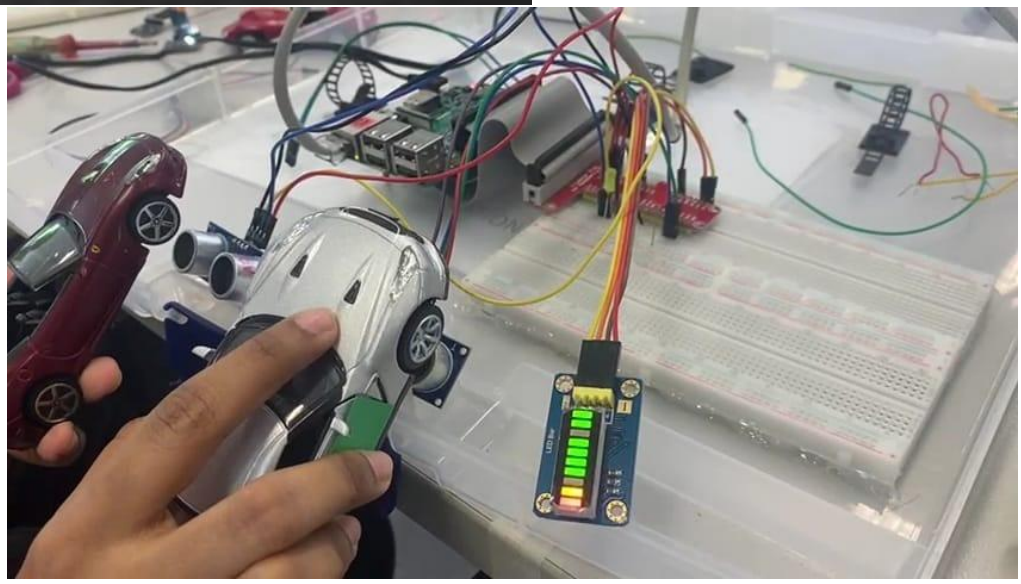


Convenient Parking:

As the driver drives in, he/she can view number of parking spots are available and the locations through another LED Bar Module set on top. When a bar is lighted up, it means the lot is vacant. Whereas if a certain bar is not lighted up, that lot is occupied.

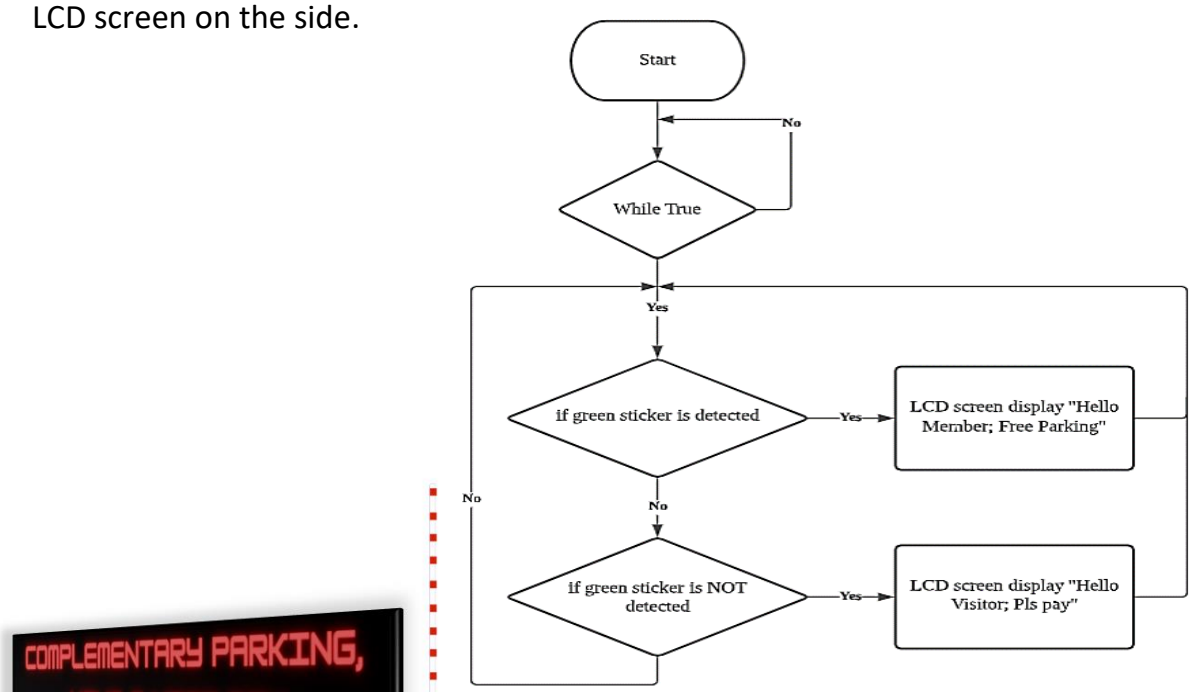
Besides that, parking lots have another LED on top which will light up if no car is present. This could make spotting out empty lots easier for drivers. The LEDs and the LED Bar Module are triggered by an Ultrasonic Sensor set at the top of each parking lot.

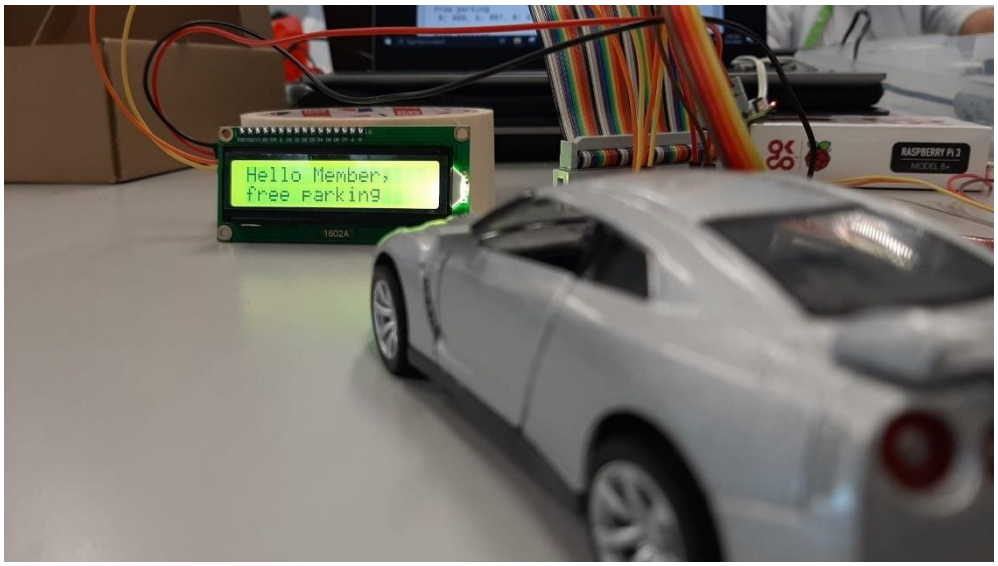




Parking Permit Sticker:

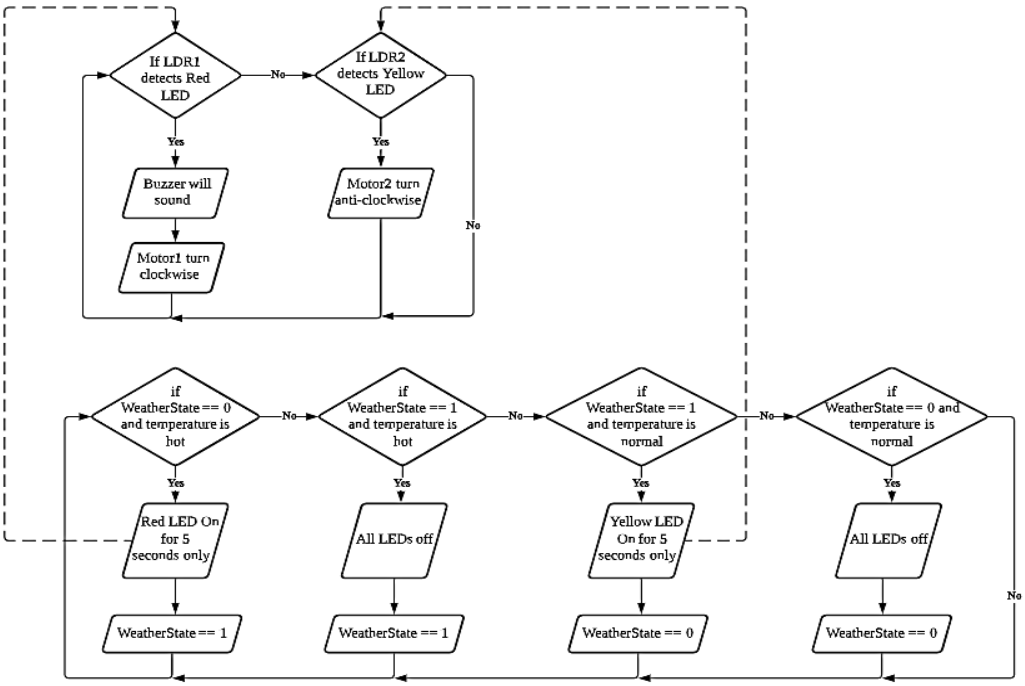
Near the exit is where payment for parking is involved. Once a person wishes to leave the carpark and drive towards the exit gantry, he/she will be told to pay the amount they owe. An RGB sensor is set near the gate which detects for the permit sticker only given to members. Members with the sticker will have complementary parking while visitors would have to pay $2.30. The payment will be displayed on an LCD screen on the side.

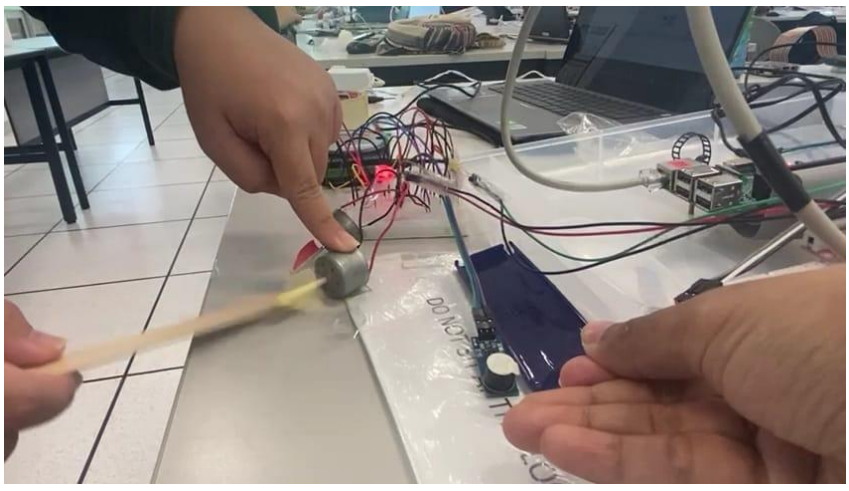




Shelter & Weather Conditions:

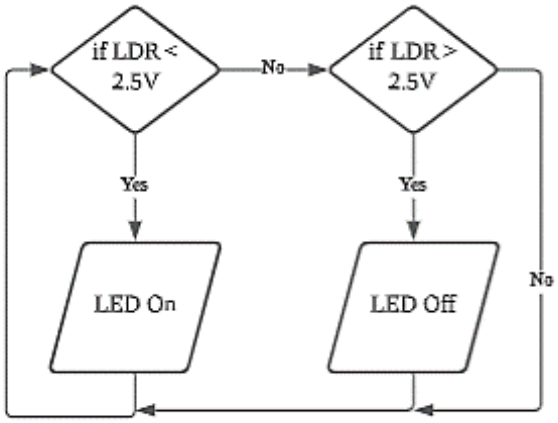
The carpark originally has a transparent glass roof to avoid getting drenched. On days when the heat rises, the electric shutter will automatically turn on and cover the roof. A buzzer will also go on when the shutter closes. When temperature sensor detects normal weather after a hot day, the shutter automatically opens up.

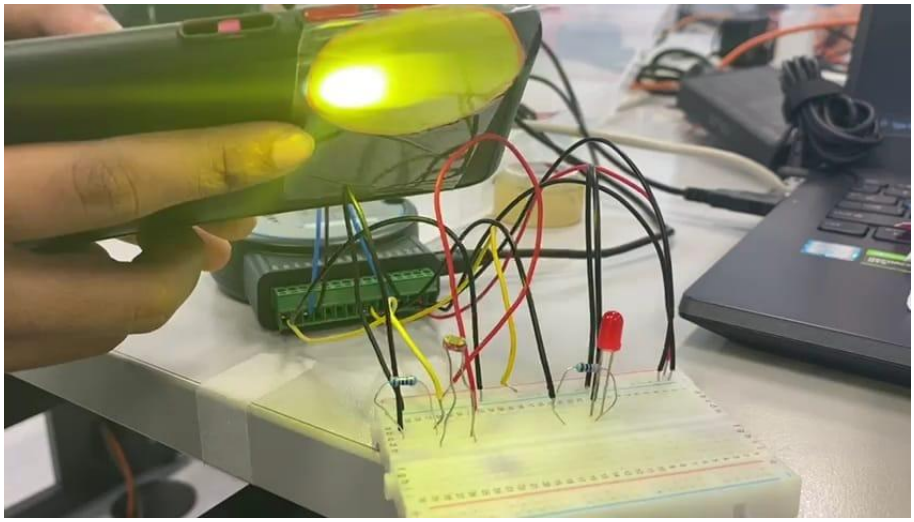




Luminaires:

When the interior brightness is low due to sunset, cloudy weather or when the shutter covers the roof, the luminaires will turn on for a proper view for anyone walking or driving inside.

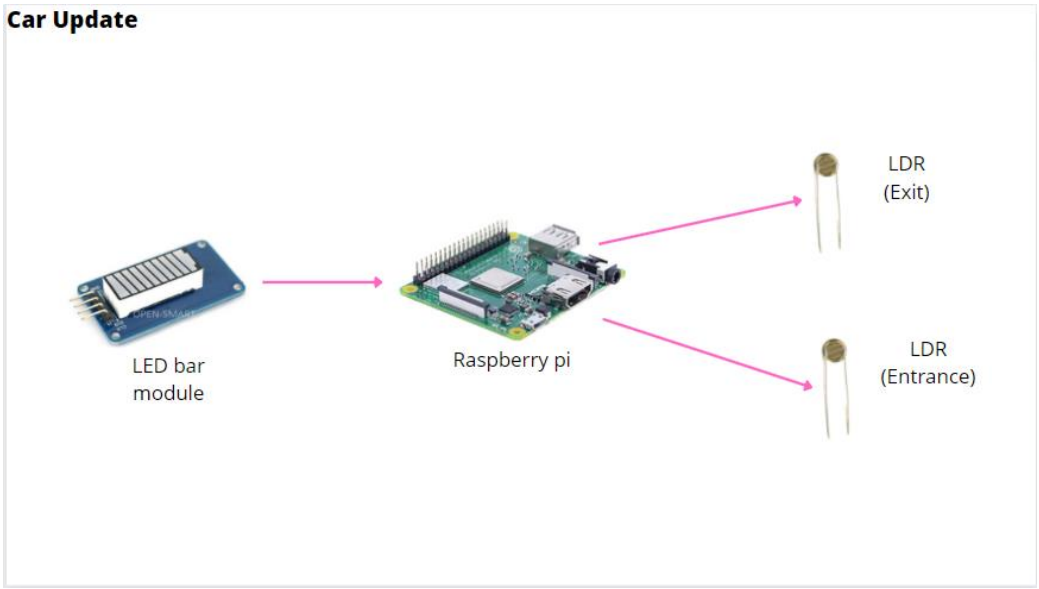




C)Project Description *(With Connection Diagram)*

Car Update:

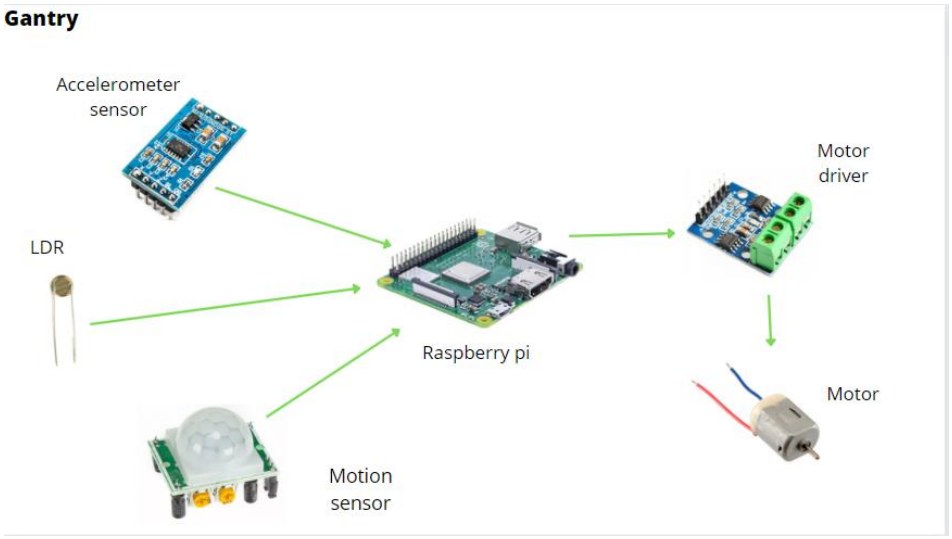
2 LDRs are set one near the entrance and another near the exit. Whenever the LDR at Entrance gets covered by a car *(Light value LOW)*, value will be added by 1 digit. Whenever the LDR at Exit gets covered by a car *(Light value LOW),* counter value will be deducted by 1 digit. The counter value is immediately updated on the LED Bar Module.



Gantry:

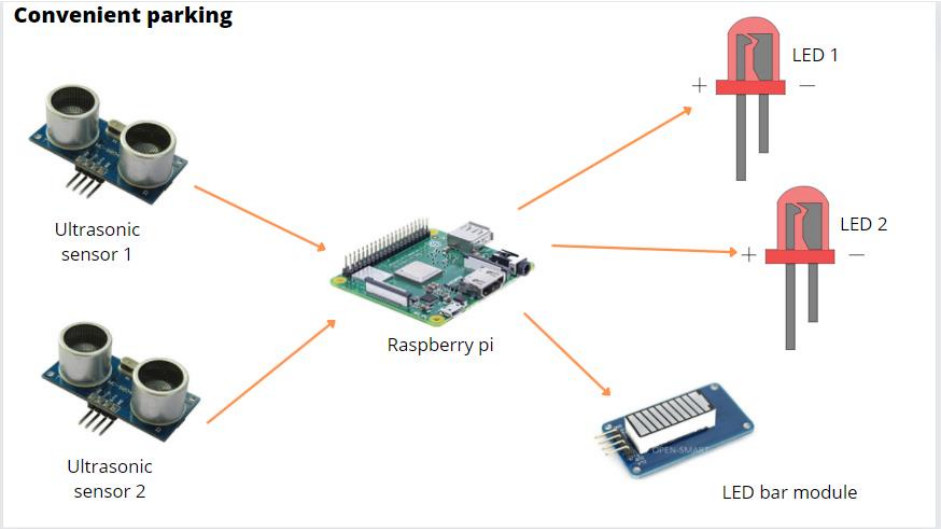
The Motion Sensor and LDR will trigger the Motor to turn clockwise or anti-clockwise. The accelerometer will start sensing the y-axis angle once motor is ON and will trigger the motor to OFF at a certain angle. When Motion is detected and LDR is covered *(Light value LOW)*, motor will turn anti-clockwise which will cause the gate to open. Motor will only stop when accelerometer detects up.

After that, once light is detected *(Light value HIGH)*, motor will spin clockwise which will cause the gate to close. The Motor will only stop once accelerometer detects down.



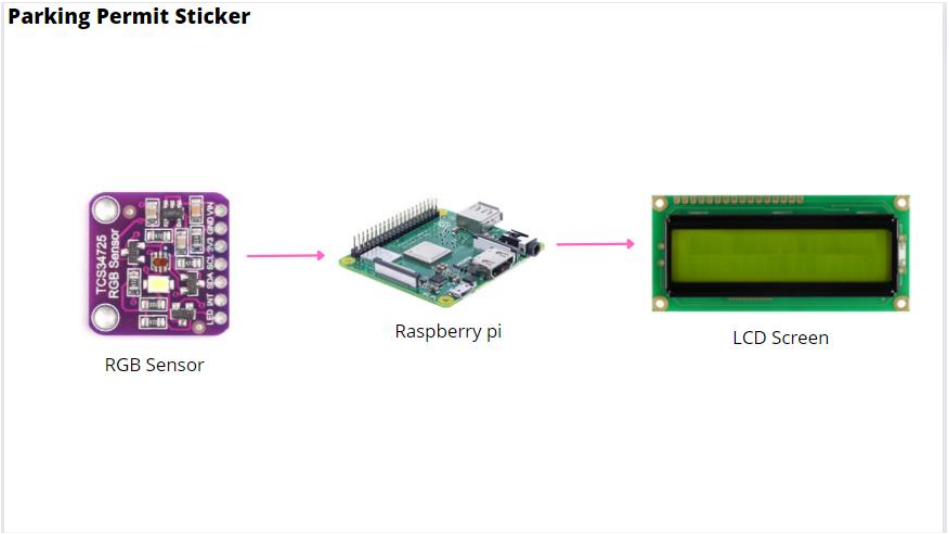
Convenient Parking:

2 Ultrasonic Sensors are used as an example. When Ultrasonic Sensor measure distance above 20cm, the single LED and 1 LED Bar from the LED Bar Module designated to the parking lot will be ON.
When Ultrasonic Sensor measure distance below 20cm, the designated LED and LED Bar will be OFF.



Parking Permit sticker:

When the RGB sensor detects the permit sticker given only to members, the LCD display at front will show “Hello Member, Free Parking”
When the RGB sensor didn’t detect the permit sticker, the LCD display will show “Hello Visitor, please pay $2.30”.

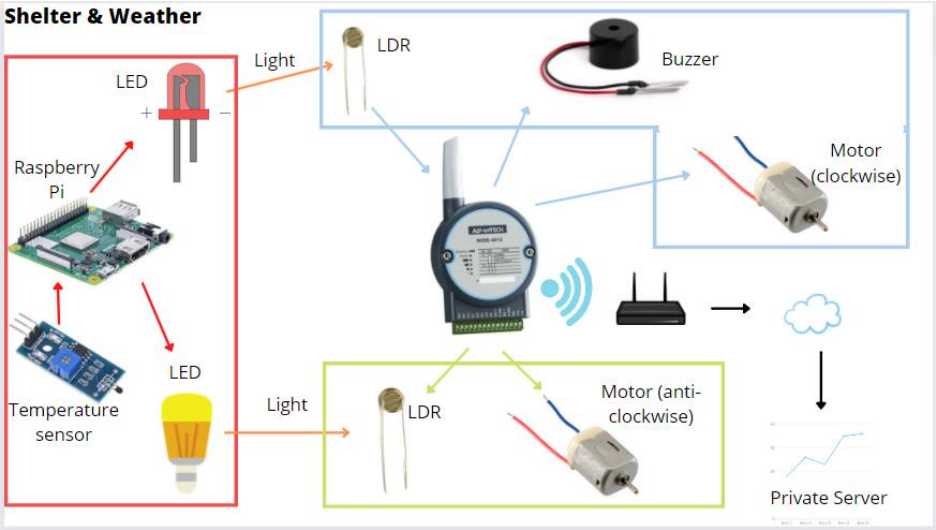


Shutter based on weather conditions:

Raspberry Pi and Arduino are connected and will communicate within each other through LEDs and LDRs.

The temperature sensor connected to the Raspberry Pi will check for sultry weather. Once temperature rises, Pi will send alert through a red LED which will be ON for 5 seconds. When the LDR from Advantech WISE detect this light, it will make the motor spin clockwise and a buzzer ON for 5 seconds.

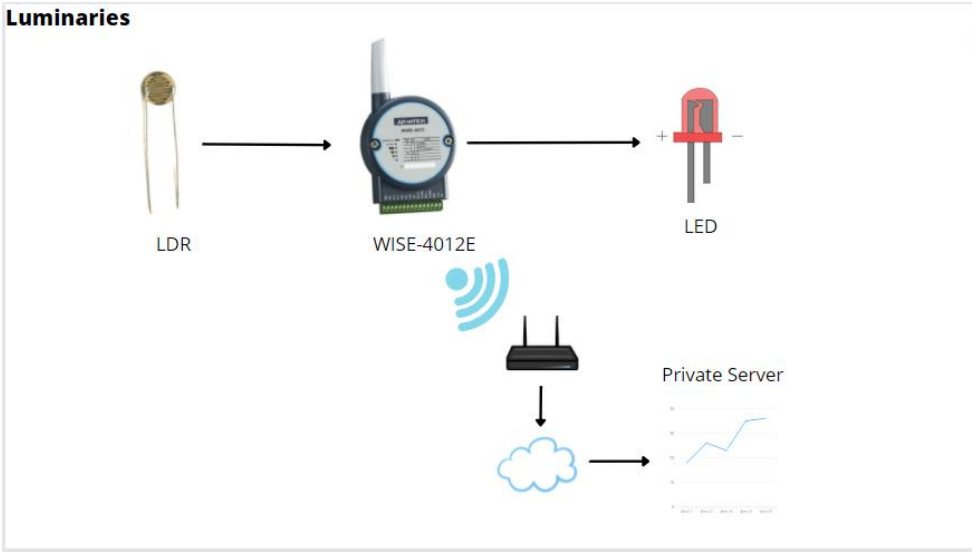
Once temperature sensor detects normal temperature after a hot period, Pi will send alert through a yellow LED which will ON for 5 seconds as well. When another LDR from Advantech WISE detect this light, it will make the motor turn anti-clockwise for 5 seconds.



Luminaires:

When LDR detects low brightness, the Luminaires will turn on.

When LDR detects brightness, Luminaires will turn off.



D) Source Code

Car Update (Abirami) :

```
#DVA Project
#Requirement: Car Update to LED Bar Module

import RPi.GPIO as GPIO             #import Raspberry Pi GPIO module
from time import sleep
GPIO.setmode(GPIO.BCM)              #set Broadcom SOC Channel
GPIO.setwarnings(False)             #Disable warnings


Entrance = 17                       #Insert LDR at Entrance to GPIO 17
GPIO.setup(Entrance, GPIO.IN)       #Set LDR at Entrance as Input
Exit = 27                           #Insert LDR at Exit to GPIO 27
GPIO.setup(Exit, GPIO.IN)           #Set LDR at Exit as Input

LED_CLK = 21                        #Insert SCL pin of LED BAR MODULE to GPIO 21
LED_DATA = 20                       #Insert SDA pin of LED BAR MODULE to GPIO 20
GPIO.setup(LED_CLK, GPIO.OUT)       #||Set SCL PIN as output
GPIO.setup(LED_DATA, GPIO.OUT)      #||Set SDA PIN as output
GPIO.output(LED_CLK, GPIO.HIGH)     #||Set SCL output initial value as HIGH
GPIO.output(LED_DATA, GPIO.HIGH)    #||Set SDA output initial value as HIGH

_state = 10 * [0]                   #Define array of 10 memory as _state and initial value as OFF for all 10 bars in LED_BAR modul


def setBits(count):                 #Function to Set LED ON value 0x20 and OFF value 0x00 in _state array of memory
    global _state

    for i in range(10):             #
        if (count>i):               #if _state memory array address < count value
            _state[i] = 0x20        #Set Led bar is ON
        else:                       #if not...
            _state[i] = 0x00        #Set Led bar is OFF

    setData()                       #Call I2C function to output the _state memory array value to LED_BAR


def setData():                      #Function to output _state memory content to LED bar through I2C ommunication
    global _state
    sendData(0)

    for i in range(10):
        sendData(_state[10 - i - 1])        #Last memory value to first output to match the LED_BAR requirement

    sendData(0)
    sendData(0)
    GPIO.output(LED_DATA, GPIO.LOW)         #Stop Condition of I2C
    sleep(0.01)                             #sleep 10 millisecond

    for i in range(10):
        sendData(_state[10 - i - 1])        #Last memory value to first output to match the LED_BAR requirement

    sendData(0)
    sendData(0)
    GPIO.output(LED_DATA, GPIO.LOW)         #Stop Condition of I2C
    sleep(0.01)                             #sleep 10 millisecond
    for i in range(4):
        GPIO.output(LED_DATA, GPIO.HIGH)
        GPIO.output(LED_DATA, GPIO.LOW)


def sendData(data):                         #Send data to LED_BAR using I2C communication protocol
    state=GPIO.LOW
    state1=GPIO.LOW

    for i in range(16):
        if (data & 0x8000):                 # Check MSB of the value in data
            state1 = GPIO.HIGH              # If MSB high set SDA value as High
        GPIO.output(LED_DATA, state1)       # set SDA pin output to SDA value
        state = 1 - state                   # Togle SCL value
        GPIO.output(LED_CLK, state)         # set SCL pin output to SCL value
        data = data << 1                    # bitwise shift left 1 bit to update MSB in data
        #print(data)


counter = 0                                 #Initialize counter as zero at first
Entrance_Statepre = False                   #Initialize Entrance_Statepre or previous Entrance State as False
Exit_Statepre = False                       #Initialize Exit_Statepre or previous Exit State as False

while True:
    Entrance_State = GPIO.input(Entrance)   #set Entrance_State
    Exit_State = GPIO.input(Exit)

    if Entrance_Statepre == True and Entrance_State == False:   #if car begin to block the LDR at Entrance ...
        counter += 1                                            #increment counter
        if counter >10:                                         #if number of cars inside is above 10...
            counter = 10                                        #counter will remain 10

        setBits(counter)

    elif Exit_Statepre == True and Exit_State == False:         #if car about to clear LDR at Exit
        counter -= 1                                            #decrement counter
        if counter <0:                                          #if count value is calculated to be below 0
            counter = 0                                         #counter will remain 0 in Led Bar Module

        setBits(counter)

    Entrance_Statepre = Entrance_State  #as it goes as a loop, update "Entrance_Statepre" as equal to current Entrance LDR state
    Exit_Statepre = Exit_State          #as it goes as a loop, update "Exit_Statepre" as equal to current Exit LDR state
```

Gantry Gate (Abirami) **:**

```
#DVA Project
#Requirement: Gantry

import RPi.GPIO as GPIO             #import Raspberry Pi GPIO module
import time                         #import time
import smbus                        #Open i2c bus 1 and read one byte from address 80
import math                         #To use mathematical fucntions and give acced to underlying C lib functions
GPIO.setwarnings(False)             #Disable warnings
GPIO.setmode(GPIO.BCM)              #set Broadcom SOC Channel

###Inputs###
MS = 26                             #insert Motion Sensor to GPIO 26
GPIO.setup (MS, GPIO.IN)            #set Motion Sensor as input
LDR = 22                            #insert LDR to GPIO 22
GPIO.setup(LDR, GPIO.IN)            #set Motion Sensor as input

###Outputs###
OpeningMotor = 25                   #insert A-IA(OpeningMotor) pin at GPIO 25
ClosingMotor = 23                   #insert A-IB(ClosingMotor) pin at GPIO 23
GPIO.setup(OpeningMotor, GPIO.OUT)  #set A-IA pin as Output
GPIO.setup(ClosingMotor, GPIO.OUT)  #set A-IB pin as Output

power_mgmt_1=0x6b   #Define Power Management 1 as 0x6b
power_mgmt_2=0x6c   #Define Power Managenenr 2 as 0x6c

def read_byte(reg):                             #Read byte value from accelerometer
    return bus.read_byte_data(address, reg)     #Read byte value from the address

def read_word(reg):                             #Read word value from accelerometer
    h=bus.read_byte_data(address, reg)          #Read High Side byte value from address
    l= bus.read_byte_data(address, reg+1)       #Read Low Side byte value from adress

    value= (h<<8)+1                             #Shift left high side 8 bits + add Low Side to form 16 bit
    return value

def read_word_2c(reg):                          #Convert 2s compliment if value is negative
    val=read_word(reg)
    if(val>=0x8000):                            # Check value is Negative number
        return-((65535 - val)+1)                # convert 2s compliment negative value
    else:
        return val                              # if positive value pass the value

bus= smbus.SMBus(1)                             #System Management Bus
address = 0x68                                  #Address is 0x68
bus.write_byte_data (address, power_mgmt_1, 0)


def Gate_off():                                 #Define Gate_off as...
    GPIO.output(OpeningMotor, False)
    GPIO.output(ClosingMotor, False)

def Gate_open():                                #Define Gate_open as...
    Accelerometer_yout = read_word_2c(0x3d)
    while Accelerometer_yout < 10000:           #Motor is ON until accelerometer detects gate is completely open
        Accelerometer_yout = read_word_2c(0x3d) #Y-axis calculation
        GPIO.output(OpeningMotor, True)         #Gate opening
        GPIO.output(ClosingMotor, False)        #^
        print (" Accelerometer_yout: ", ("%6d" % Accelerometer_yout))
    Gate_off()                                  #Motor stops once accelrometer detectd gate is open


def Gate_close():                               #Define Gate_close as...
    Accelerometer_yout = read_word_2c(0x3d)
    while Accelerometer_yout > 900:             #Motor is ON until accelerometer detects gate is completely close
        Accelerometer_yout = read_word_2c(0x3d) #Y-axis calculation
        GPIO.output(OpeningMotor, False)        #Gate closing
        GPIO.output(ClosingMotor, True)         #^
        print (" Accelerometer_yout: ", ("%6d" % Accelerometer_yout))
    Gate_off()                                  #Motor stops once accelrometer detectd gate is closed



Gate_state = 0  #Gate state is officially 0
Gate_off()      #Gate is orignally off


while True:
    Mdetected = GPIO.input(MS)        #define Mdetected as Motion Detection
    Ldetected = GPIO.input(LDR)       #define Ldetected as Light Detection

    Accelerometer_xout = read_word_2c(0x3b)    #X-axis calculation
    Accelerometer_yout = read_word_2c(0x3d)    #Y-axis calculation
    Accelerometer_zout = read_word_2c(0x3f)    #Z-axis calculation


    if Gate_state==0 and Mdetected == True and Ldetected == False: #if Gate state is 0, and Motion is detected, and Light is covered (by a car)
        Gate_open()                                               #Gate opens to let car in
        Gate_state = 1                                            #Gate state will become 0


    elif Gate_state==1 and Ldetected == True:                      #if Gate state is 1 and Light is detected, (car has passed through)
        time.sleep(1)                                             #will wait for 1 second
        Gate_close()                                              #Gate will close
        Gate_state = 0                                            #Gate state will become 0
```

***Gantry Gate and Car Update** are originally combined* (Abirami) **:**

```
#DVA Project
#Requirement: Gantry & Car Update to LED Bar Module

import RPi.GPIO as GPIO             #import Raspberry Pi GPIO module
import time                         #import time
from time import sleep
import smbus                        #Open i2c bus 1 and read one byte from address 80
import math                         #To use mathematical fucntions and give acced to underlying C lib functions
GPIO.setwarnings(False)             #Disable warnings
GPIO.setmode(GPIO.BCM)              #set Broadcom SOC Channel

###Inputs###
MS = 26                             #insert Motion Sensor to GPIO 26
GPIO.setup (MS, GPIO.IN)            #set Motion Sensor as input
Entrance = 17                       #Insert LDR at Entrance to GPIO 17
GPIO.setup(Entrance, GPIO.IN)       #Set LDR at Entrance as Input
Exit = 27                           #Insert LDR at Exit to GPIO 27
GPIO.setup(Exit, GPIO.IN)           #Set LDR at Exit as Input

LED_CLK = 21                        #Insert SCL pin of LED BAR MODULE to GPIO 21
LED_DATA = 20                       #Insert SDA pin of LED BAR MODULE to GPIO 20
GPIO.setup(LED_CLK, GPIO.OUT)       #||Set SCL PIN as output
GPIO.setup(LED_DATA, GPIO.OUT)      #||Set SDA PIN as output
GPIO.output(LED_CLK, GPIO.HIGH)     #||Set SCL output initial value as HIGH
GPIO.output(LED_DATA, GPIO.HIGH)    #||Set SDA output initial value as HIGH

_state = 10 * [0]                   #Define array of 10 memory as _state and initial value as OFF for all 10 bars in LED_BAR module


def setBits(count):                 #Function to Set LED ON value 0x20 and OFF value 0x00 in _state array of memory
    global _state

    for i in range(10):             #
        if (count>i):               #if _state memory array address < count value
            _state[i] = 0x20        #Set Led bar is ON
        else:                       #if not...
            _state[i] = 0x00        #Set Led bar is OFF

    setData()                       #Call I2C function to output the _state memory array value to LED_BAR

def setData():                          #Function to output _state memory content to LED bar through I2C ommunication
    global _state
    sendData(0)

    for i in range(10):
        sendData(_state[10 - i - 1])        #Last memory value to first output to match the LED_BAR requirement

    sendData(0)
    sendData(0)
    GPIO.output(LED_DATA, GPIO.LOW)         #Stop Condition of I2C
    sleep(0.01)                             #sleep 10 millisecond
    for i in range(4):
        GPIO.output(LED_DATA, GPIO.HIGH)
        GPIO.output(LED_DATA, GPIO.LOW)


def sendData(data):                         #Send data to LED_BAR using I2C communication protocol
    state=GPIO.LOW
    state1=GPIO.LOW

    for i in range(16):
        if (data & 0x8000):                 # Check MSB of the value in data
            state1 = GPIO.HIGH              # If MSB high set SDA value as High
        GPIO.output(LED_DATA, state1)       # set SDA pin output to SDA value
        state = 1 - state                   # Togle SCL value
        GPIO.output(LED_CLK, state)         # set SCL pin output to SCL value
        data = data << 1                    # bitwise shift left 1 bit to update MSB in data
        #print(data)



###Outputs###
OpeningMotor = 25                   #insert A-IA(OpeningMotor) pin at GPIO 25
ClosingMotor = 23                   #insert A-IB(ClosingMotor) pin at GPIO 23
GPIO.setup(OpeningMotor, GPIO.OUT)  #set A-IA pin as Output
GPIO.setup(ClosingMotor, GPIO.OUT)  #set A-IB pin as Output

power_mgmt_1=0x6b   #defube Power Management 1 as 0x6b
power_mgmt_2=0x6c   #Define Power Managenenr 2 as 0x6c

def read_byte(reg):                             #Read byte value from accelerometer
    return bus.read_byte_data(address, reg)     #Read byte value from the address

def read_word(reg):                             #Read word value from accelerometer
    h=bus.read_byte_data(address, reg)          #Read High Side byte value from address
    l= bus.read_byte_data(address, reg+1)       #Read Low Side byte value from adress
    value= (h<<8)+1                         #Shift left high side 8 bits + add Low Side to form 16 bit
    return value

def read_word_2c(reg):                      #Convert 2s compliment if value is negative
    val=read_word(reg)
    if(val>=0x8000):                        # Check value is Negative number
        return-((65535 - val)+1)            # convert 2s compliment negative value
    else:
        return val                          # if positive value pass the value

bus= smbus.SMBus(1)                         #System Management Bus
address = 0x68                              #Address is 0x68
bus.write_byte_data (address, power_mgmt_1, 0)
```

```
def Gate_off():                                 #Define Gate_off as...
    GPIO.output(OpeningMotor, False)
    GPIO.output(ClosingMotor, False)

def Gate_open():                                #Define Gate_open as...
    Accelerometer_yout = read_word_2c(0x3d)
    while Accelerometer_yout < 10000:           #Motor is ON until accelerometer detects gate is completely open
        Accelerometer_yout = read_word_2c(0x3d) #Y-axis calculation
        GPIO.output(OpeningMotor, True)         #Gate opening
        GPIO.output(ClosingMotor, False)        #^
        print (" Accelerometer_yout: ", ("%6d" % Accelerometer_yout))
    Gate_off()                                  #Motor stops once accelrometer detectd gate is open


def Gate_close():                               #Define Gate_close as...
    Accelerometer_yout = read_word_2c(0x3d)
    while Accelerometer_yout > 900:             #Motor is ON until accelerometer detects gate is completely close
        Accelerometer_yout = read_word_2c(0x3d) #Y-axis calculation
        GPIO.output(OpeningMotor, False)        #Gate closing
        GPIO.output(ClosingMotor, True)         #^
        print (" Accelerometer_yout: ", ("%6d" % Accelerometer_yout))
    Gate_off()                                  #Motor stops once accelrometer detectd gate is closed



Gate_state = 0  #Gate state is officially 0
Gate_off()      #Gate is orignally off

counter = 0                                 #Initialize counter as zero at first
Entrance_Statepre = False                   #Initialize Entrance_Statepre or previous Entrance State as False
Exit_Statepre = False                       #Initialize Exit_Statepre or previous Exit State as False


while True:
    Mdetected = GPIO.input(MS)        #define Mdetected as Motion Detection
    Ldetected = GPIO.input(Entrance)       #define Ldetected as Light Detection

    Accelerometer_xout = read_word_2c(0x3b)    #X-axis calculation
    Accelerometer_yout = read_word_2c(0x3d)    #Y-axis calculation
    Accelerometer_zout = read_word_2c(0x3f)    #Z-axis calculation


    if Gate_state==0 and Mdetected == True and Ldetected == False: #if Gate state is 0, and Motion is detected, and Light is covered (by a car)
        Gate_open()                                               #Gate opens to let car in
        Gate_state = 1                                            #Gate state will become 0


   elif Gate_state==1 and Ldetected == True:                        #if Gate state is 1 and Light is detected, (car has passed through)
       time.sleep(1)                                               #will wait for 1 second
       Gate_close()                                                #Gate will close
       Gate_state = 0                                              #Gate state will become 0


   Entrance_State = GPIO.input(Entrance)   #set Entrance_State
   Exit_State = GPIO.input(Exit)

   if Entrance_Statepre == True and Entrance_State == False:   #if car begin to block the LDR at Entrance ...
       counter += 1                                            #increment counter
       if counter >10:                                         #if number of cars inside is above 10...
           counter = 10                                        #counter will remain 10

       setBits(counter)

   elif Exit_Statepre == True and Exit_State == False:         #if car about to clear LDR at Exit
       counter -= 1                                            #decrement counter
       if counter <0:                                          #if count value is calculated to be below 0
           counter = 0                                         #counter will remain 0 in Led Bar Module

       setBits(counter)

   Entrance_Statepre = Entrance_State  #as it goes as a loop, update "Entrance_Statepre" as equal to current Entrance LDR state
   Exit_Statepre = Exit_State          #as it goes as a loop, update "Exit_Statepre" as equal to current Exit LDR state
```

Convenient Parking (Phoebe) **:**

```
#DVA Project
#Requirement: Parking spots

import  RPi.GPIO as GPIO                        #import RPi.GPIO moduole
from time import sleep
import time                                     #import time
GPIO.setmode(GPIO.BCM)                          #set Broadcom SOC Channel
GPIO.setwarnings(False)

#LED Bar Module
LED_CLK = 21                                    #set SCL to GPIO21
LED_DATA = 20                                   #set SDA to GPIO20

#Ultrasonic 1
trig = 17                                       #set trig pin to GPIO17
echo = 27                                       #set echo pin to GPIO27
led = 22                                        #set LED to GPIO22

#Ultrasonic 2
TRIG = 26                                       #set TRIG pin to GPIO26
ECHO = 19                                       #set ECHO pin to GPIO19
Red = 13                                        #set Red LED to GPIO13

#LED Bar Module
GPIO.setup(LED_CLK, GPIO.OUT)                   #set SCL as output
GPIO.setup(LED_DATA, GPIO.OUT)                  #set SDA as output

#Ultrasonic 1
GPIO.setup(trig, GPIO.OUT)                      #set trig pin as output
GPIO.setup(echo, GPIO.IN)                       #set echo pin as input
GPIO.setup(led, GPIO.OUT)                       #set LED as output

#Ultrasonic 2
GPIO.setup(TRIG, GPIO.OUT)                      #set TRIG pin as output
GPIO.setup(ECHO, GPIO.IN)                       #set ECHO pin as input
GPIO.setup(Red, GPIO.OUT)                       #set Red LED as output

GPIO.output(LED_CLK, GPIO.HIGH)                 #SCL on
GPIO.output(LED_DATA, GPIO.HIGH)                #SDA on
_state = 10 * [0x20]                            #all LED in LED Bar Module is on

def setData():
    global _state
    sendData(0)

    for i in range(10):
        sendData(_state[10 - i - 1])            #last memory value to first output to match  the LED bar module requirement

    sendData(0)
    sendData(0)
    GPIO.output(LED_DATA, GPIO.LOW)
    sleep(0.01)                                 #sleep 10 millisecond
    for i in range(4):
        GPIO.output(LED_DATA, GPIO.HIGH)
        GPIO.output(LED_DATA, GPIO.LOW)

def sendData(data):
    state=GPIO.LOW
    state1=GPIO.LOW

    for i in range(16):
        if (data & 0x8000):                     #check MSB of the value in data
            state1 = GPIO.HIGH                  #if MSB high, set SDA value as HIGH
        GPIO.output(LED_DATA, state1)           #set SDA pin output to SDA value
        state = 1 - state                       #togle SCL value
        GPIO.output(LED_CLK, state)             #set SCL pin output to SCL value
        data = data << 1                        #bitwise shift left 1 bit to update MSB in data

def c():
    GPIO.output(led, 0)                         #LED off
    GPIO.output(Red, 1)                         #Red LED on
    global _state
    _state[2] = 0x00                            #2nd LED in Bar Module off
    _state[7] = 0x20                            #7th LED in Bar Module on
    setData()

def u():
    GPIO.output(led, 1)                         #LED on
    GPIO.output(Red, 0)                         #Red LED off
    global _state
    _state[2] = 0x20                            #2nd LED in Bar Module on
    _state[7] = 0x00                            #7th LED in Bar Module off
    setData()


def both():
    GPIO.output(led, 0)                         #LED off
    GPIO.output(Red, 0)                         #Red LED off
    global _state
    _state[2] = 0x00                            #2nd LED in Bar Module off
    _state[7] = 0x00                            #7th LED in Bar Module off
    setData()

def gone():
    GPIO.output(led, 1)                         #LED on
    GPIO.output(Red, 1)                         #Red LED on
    global _state
    _state[2] = 0x20                            #2nd LED in Bar Module on
    _state[7] = 0x20                            #7th LED in Bar Module on
    setData()
```

```
while True:
    #Ultasonic 1
    GPIO.output(trig, True)                     #trig on
    time.sleep(0.00002)                         #delay for 0.02 millisecond
    GPIO.output(trig, False)                    #trig off

    while GPIO.input(echo)==0:                  #when echo does not receive any signal
        pulse_start = time.time()               #record time

    while GPIO.input(echo)==1:                  #when echo receives signal
        pulse_end = time.time()                 #record time

    pulse_duration = pulse_end - pulse_start    #duration between the time recorded
    d = pulse_duration *17150                   #find distance
    d = round(d, 2)                             #round distance to 2 decimal places
    print("Lot 1: ", d, " cm")                  #print distance from car to Ultrasonic 1

    #Ultrasonic 2
    GPIO.output(TRIG, True)                     #TRIG on
    time.sleep(0.00002)                         #delay for 0.02 millisecond
    GPIO.output(TRIG, False)                    #TRIG off

    while GPIO.input(ECHO)==0:                  #when ECHO does not receive any signal
        pulse_star = time.time()                #record time

    while GPIO.input(ECHO)==1:                  #when ECHO receive signal
        pulse_en = time.time()                  #record time

    pulse_duratio = pulse_en - pulse_star       #duration between the time recorded
    D = pulse_duratio *17150                    #find distance
    D = round(D, 2)                             #round distance to 2 decimal places
    print("Lot 2: ", D, " cm")                  #print distance from car to Ultrasonic 2

    if d < 20 and D > 20:                       #car parked under Ultrasonic 1
        c()
        print("Lot 1 is taken")                 #print
    elif d > 20 and D < 20:                     #car parked under Ultrasonic 2
        u()
        print("Lot 2 is taken")                 #print
    elif d < 20 and D < 20:                     #cars parked under both Ultrasonic
        both()
        print("Both lots are taken")            #print
    elif d > 20 and D > 20:                     #no cars parked under both Ultrasonic
        gone()
        print("All lots are empty")             #print
```

Parking Permit Sticker (Abirami) **:**

```
#DVA Project
#Requirement: Parking Permit Sticker

import smbus            # Open SystemManagement(i2c) bus
import time
bus = smbus.SMBus(1)    # Open i2c bus 1 and read one byte from address 80
from signal import signal, SIGTERM, SIGHUP, pause
from rpi_lcd import LCD #import LCD, adress is 7x27
lcd = LCD()

# I2C address 0x29
# Register 0x12 has device ver.
# Register addresses must be OR'ed with 0x80

bus.write_byte(0x29,0x80|0x12)  #for RGB sensor
ver = bus.read_byte(0x29)       #for RGB sensor
# version # should be 0x44

def safe_exit(signum, frame):   #point to the frame that was interrupted by the signal
    exit(1)

signal(SIGTERM, safe_exit)      #for LCD screen
signal(SIGHUP, safe_exit)       #for LCD screen

if ver == 0x44:
    print ("Device found\n")
    bus.write_byte(0x29, 0x80|0x00) # 0x00 = ENABLE register
    bus.write_byte(0x29, 0x01|0x02) # 0x01 = Power on, 0x02 RGB sensors enabled
    bus.write_byte(0x29, 0x80|0x14) # Reading results start register 14, LSB then MSB
    while True:
        data = bus.read_i2c_block_data(0x29, 0)
       # clear = clear = data[1] << 8 | data[0]
        red = data[3] << 8 | data[2]            #Calculation for Red Color
        green = data[5] << 8 | data[4]          #Calculation for Green Color
        blue = data[7] << 8 | data[6]           #Calculation for Blue Color
        rgb = (" R: %s, G: %s, B: %s\n") % (red, green, blue) #print values for checking purpose
      #  print (crgb)
        print (rgb)

        if red < green > blue:  #if green color is detected     ################################
            lcd.text("Hello Member,", 1) #LINE 1                ##| LCD display will say    |##
            lcd.text("free parking", 2)  #LINE 2                ##|    "free parking"       |##
            print("Member") #print update for checking purpose  ################################

        else:               #if green color not detected        ################################
            lcd.text("Hello visitor,", 1)  #LINE 1              ##| LCD display will say    |##
            lcd.text("please pay $2.30", 2)#LINE 2              ##|  "Parking Payment"      |##
            print("Visitor")#print update for checking purpose  ################################
        time.sleep(2)
else:
    print ("Device not found\n")
```

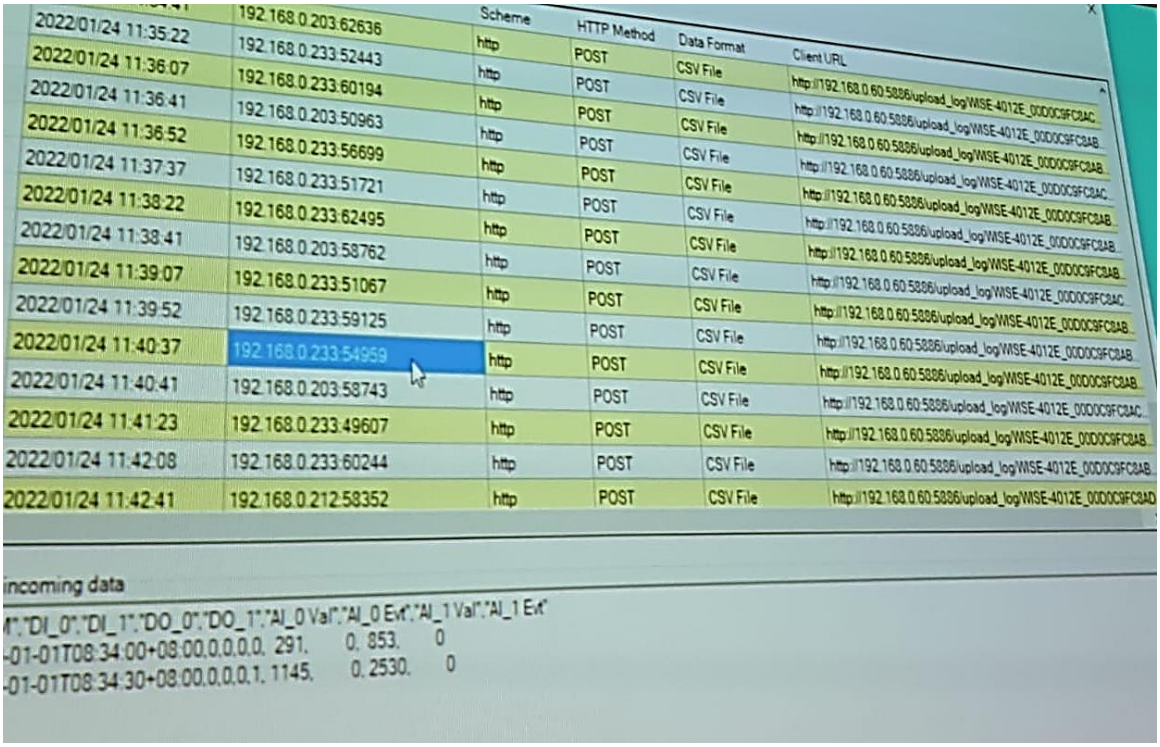
Shelter & weather Conditions (Abirami and Phoebe) **:**

```
#DVA project
#Requirement: Shelter & Weather Conditions

import RPi.GPIO as GPIO                          #import Raspberry Pi GPIO module
import time                                      #import time
GPIO.setwarnings(False)                          #Disable warnings
GPIO.setmode(GPIO.BCM)                           #set Broadcom SOC Channel

TempSensor = 16                                  #insert TempSensor at GPIO 16
HotWeather = 17                                  #insert LED(red clr) alert for HotWeather at GPIO 17
BackToNormal = 22                                #insert LED(yellow clr) alert for BackRToNormal at GPIO 22

GPIO.setup(TempSensor, GPIO.IN)                  #set TempSensor as input
GPIO.setup(HotWeather, GPIO.OUT)                 #set HotWeather as output
GPIO.setup(BackToNormal, GPIO.OUT)               #set BackToNormal as output

def HotWeather():                                #define HotWeather condition as...
    GPIO.output(HotWeather, True)                #Red Led On
    GPIO.output(BackToNormal, False)             #Yellow Led Off

def NormalWeather():                             #define NormalWeather condition as...
    GPIO.output(HotWeather, False)               #Red Led Off
    GPIO.output(BackToNormal, True)              #Yellow Led On

def All_off():                                   #define All_off condition as...
    GPIO.output(HotWeather, False)               #Red Led Off
    GPIO.output(BackToNormal, False)             #Yellow Led Off

WeatherState = 0                                 #WeatherState is originally 0

while True:
    tempstate = GPIO.input(TempSensor)           #define input value of TempSensor as tempstate

    #Temperatue sensor detecting HOT weather
    if WeatherState == 0 and tempstate == False:   #if temperature sense HOT,
        HotWeather()                               #Led for "Hot Weather" turns ON
        time.sleep(5)                              #Led remains ON for 5 seconds
        All_off()                                  #All Leds will turn OFF
        WeatherState == 1                          #WeatherState will become 1, once HOT temperature is detected and LED has given alert

    elif WeatherState == 1 and tempstate == False: #if temperature sense hot *again*
        All_off()                                  #nothing will happen, Leds will remain OFF
        WeatherState = 1                           #WeatherState remains 1


    #Temperatue sensor detecting NORMAL weather
    elif WeatherState == 1 and tempstate == True:  #if temperature sense normal *after sensing HOT*
        NormalWeather()                            #Led for "Back to Normal Weather" turns ON
        time.sleep(5)                              #Led remains ON for 5 seconds
        All_off()                                  #All Leds will turn OFF
        WeatherState = 0                           #WeatherState will be back to 0, once Normal temperature is detected after a Hot weather.


    elif WeatherState == 0 and tempstate == True:  #if temperature sense NORMAL *again* or is *orginally* NORMAL
        All_off()                                  #nothing happens, Leds remain OFF
        WeatherState = 0                           #WeatherState remains 0

        ##Goes as a loop
```

LDR1 to V0+
LDR2 to V1+
  enable “High Alarm” *for both*
  Voltage Value: 2.5v

Buzzer to RL0+
Motor1 to RL0+
Motor2 to RL1+
  RL0+ mapping channel is V0+
  RL1+ mapping channel is V1+
  Trigger mode: ‘High Alarm’ *for both*

WISE data of shelter & weather conditions sent to private server…



		Scheme	HTTP Method	Data Format	Client URL
2022/01/24 11:35:22	192.168.0.203:62636	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AC...
2022/01/24 11:36:07	192.168.0.233:52443	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:36:41	192.168.0.233:60194	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:36:52	192.168.0.203:50963	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AC...
2022/01/24 11:37:37	192.168.0.233:56699	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:38:22	192.168.0.233:51721	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:38:41	192.168.0.233:62495	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:39:07	192.168.0.203:58762	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AC...
2022/01/24 11:39:52	192.168.0.233:51067	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:40:37	192.168.0.233:59125	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:40:41	192.168.0.233:54959	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:41:23	192.168.0.203:58743	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AC...
2022/01/24 11:42:08	192.168.0.233:49607	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
2022/01/24 11:42:41	192.168.0.233:60244	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AB...
	192.168.0.212:58352	http	POST	CSV File	http://192.168.0.60:5886/upload_log/WISE-4012E_00D0C9FC8AD...

incoming data

"DI_0","DI_1","DO_0","DO_1","AI_0 Val","AI_0 Ext","AI_1 Val","AI_1 Ext"
-01-01T08:34:00+08:00,0,0,0,0, 291, 0, 853, 0
-01-01T08:34:30+08:00,0,0,0,1, 1145, 0, 2530, 0

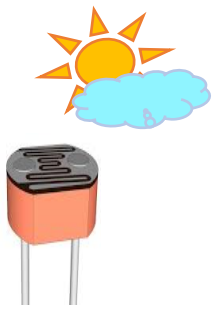
Parking Permit Sticker, Convenient Parking and Shelter & Weather Conditions Combined (Abirami & Phoebe):

```
#DVA Project
#Requirement: Gantry & Car Update to LED Bar Module

import RPi.GPIO as GPIO             #import Raspberry Pi GPIO module
import time                         #import time
from time import sleep
import smbus                        #Open i2c bus 1 and read one byte from address 80
import math                         #To use mathematical fucntions and give acced to underlying C lib functions
GPIO.setwarnings(False)             #Disable warnings
GPIO.setmode(GPIO.BCM)              #set Broadcom SOC Channel

###Inputs###
MS = 26                             #insert Motion Sensor to GPIO 26
GPIO.setup (MS, GPIO.IN)            #set Motion Sensor as input
Entrance = 17                       #Insert LDR at Entrance to GPIO 17
GPIO.setup(Entrance, GPIO.IN)       #Set LDR at Entrance as Input
Exit = 27                           #Insert LDR at Exit to GPIO 27
GPIO.setup(Exit, GPIO.IN)           #Set LDR at Exit as Input

LED_CLK = 21                        #Insert SCL pin of LED BAR MODULE to GPIO 21
LED_DATA = 20                       #Insert SDA pin of LED BAR MODULE to GPIO 20
GPIO.setup(LED_CLK, GPIO.OUT)       #||Set SCL PIN as output
GPIO.setup(LED_DATA, GPIO.OUT)      #||Set SDA PIN as output
GPIO.output(LED_CLK, GPIO.HIGH)     #||Set SCL output initial value as HIGH
GPIO.output(LED_DATA, GPIO.HIGH)    #||Set SDA output initial value as HIGH

_state = 10 * [0]                   #Define array of 10 memory as _state and initial value as OFF for all 10 bars in LED_BAR module


def setBits(count):                 #Function to Set LED ON value 0x20 and OFF value 0x00 in _state array of memory
    global _state

    for i in range(10):             #
        if (count>i):               #if _state memory array address < count value
            _state[i] = 0x20        #Set Led bar is ON
        else:                       #if not...
            _state[i] = 0x00        #Set Led bar is OFF

    setData()                       #Call I2C function to output the _state memory array value to LED_BAR


def setData():                      #Function to output _state memory content to LED bar through I2C communication
    global _state
    sendData(0)

    for i in range(10):
        sendData(_state[10 - i - 1])        #Last memory value to first output to match the LED_BAR requirement

    sendData(0)
    sendData(0)
    GPIO.output(LED_DATA, GPIO.LOW)         #Stop Condition of I2C
    sleep(0.01)                             #sleep 10 millisecond
    for i in range(4):
        GPIO.output(LED_DATA, GPIO.HIGH)
        GPIO.output(LED_DATA, GPIO.LOW)


def sendData(data):                         #Send data to LED_BAR using I2C communication protocol
    state=GPIO.LOW
    state1=GPIO.LOW

    for i in range(16):
        if (data & 0x8000):                 # Check MSB of the value in data
            state1 = GPIO.HIGH              # If MSB high set SDA value as High
        GPIO.output(LED_DATA, state1)       # set SDA pin output to SDA value
        state = 1 - state                   # Togle SCL value
        GPIO.output(LED_CLK, state)         # set SCL pin output to SCL value
        data = data << 1                    # bitwise shift left 1 bit to update MSB in data
        #print(data)
```

```
###Outputs###
OpeningMotor = 25                  #insert A-IA(OpeningMotor) pin at GPIO 25
ClosingMotor = 23                  #insert A-IB(ClosingMotor) pin at GPIO 23
GPIO.setup(OpeningMotor, GPIO.OUT)  #set A-IA pin as Output
GPIO.setup(ClosingMotor, GPIO.OUT)  #set A-IB pin as Output

power_mgmt_1=0x6b   #defube Power Management 1 as 0x6b
power_mgmt_2=0x6c   #Define Power Managenenr 2 as 0x6c

def read_byte(reg):                            #Read byte value from accelerometer
    return bus.read_byte_data(address, reg)    #Read byte value from the address

def read_word(reg):                            #Read word value from accelerometer
    h=bus.read_byte_data(address, reg)         #Read High Side byte value from address
    l= bus.read_byte_data(address, reg+1)      #Read Low Side byte value from adress

    value= (h<<8)+1                            #Shift left high side 8 bits + add Low Side to form 16 bit
    return value

def read_word_2c(reg):                         #Convert 2s compliment if value is negative
    val=read_word(reg)
    if(val>=0x8000):                           # Check value is Negative number
        return-((65535 - val)+1)               # convert 2s compliment negative value
    else:
        return val                             # if positive value pass the value

bus= smbus.SMBus(1)                            #System Management Bus
address = 0x68                                 #Address is 0x68
bus.write_byte_data (address, power_mgmt_1, 0)



def Gate_off():                                #Define Gate_off as...
    GPIO.output(OpeningMotor, False)
    GPIO.output(ClosingMotor, False)

def Gate_open():                               #Define Gate_open as...
    Accelerometer_yout = read_word_2c(0x3d)
    while Accelerometer_yout < 10000:          #Motor is ON until accelerometer detects gate is completely open
        Accelerometer_yout = read_word_2c(0x3d) #Y-axis calculation
        GPIO.output(OpeningMotor, True)        #Gate opening
        GPIO.output(ClosingMotor, False)       #^
        print (" Accelerometer_yout: ", ("%6d" % Accelerometer_yout))
    Gate_off()                                 #Motor stops once accelrometer detectd gate is open


def Gate_close():                              #Define Gate_close as...
    Accelerometer_yout = read_word_2c(0x3d)
    while Accelerometer_yout > 900:            #Motor is ON until accelerometer detects gate is completely close
        Accelerometer_yout = read_word_2c(0x3d) #Y-axis calculation
        GPIO.output(OpeningMotor, False)       #Gate closing
        GPIO.output(ClosingMotor, True)        #^
        print (" Accelerometer_yout: ", ("%6d" % Accelerometer_yout))
    Gate_off()                                 #Motor stops once accelrometer detectd gate is closed



Gate_state = 0  #Gate state is officially 0
Gate_off()      #Gate is orignally off

counter = 0                                #Initialize counter as zero at first
Entrance_Statepre = False                  #Initialize Entrance_Statepre or previous Entrance State as False
Exit_Statepre = False                      #Initialize Exit_Statepre or previous Exit State as False


while True:
    Mdetected = GPIO.input(MS)        #define Mdetected as Motion Detection
    Ldetected = GPIO.input(Entrance)       #define Ldetected as Light Detection

    Accelerometer_xout = read_word_2c(0x3b)    #X-axis calculation
    Accelerometer_yout = read_word_2c(0x3d)    #Y-axis calculation
    Accelerometer_zout = read_word_2c(0x3f)    #Z-axis calculation
```

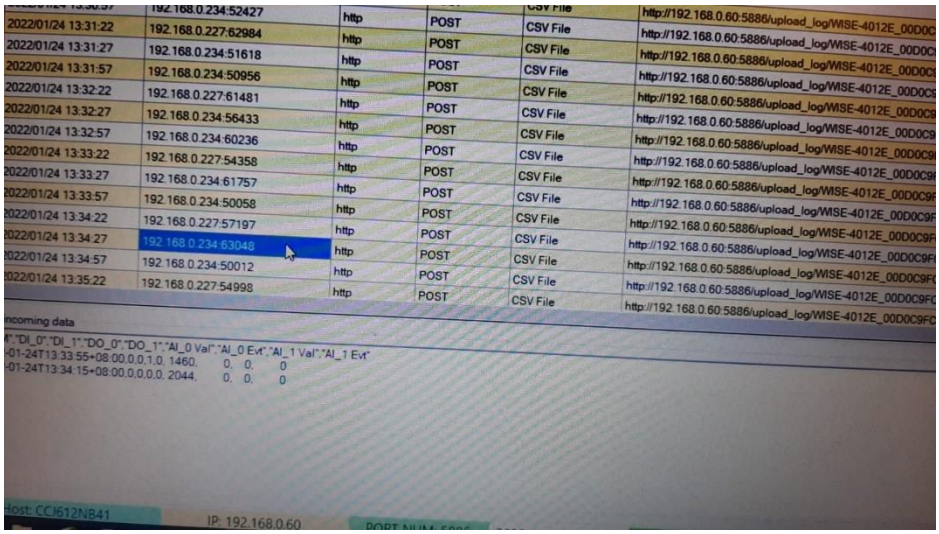
```
if Gate_state==0 and Mdetected == True and Ldetected == False: #if Gate state is 0, and Motion is detected, and Light is covered (by a car)
    Gate_open()                                                #Gate opens to let car in
    Gate_state = 1                                             #Gate state will become 0


elif Gate_state==1 and Ldetected == True:                      #if Gate state is 1 and Light is detected, (car has passed through)
    time.sleep(1)                                              #will wait for 1 second
    Gate_close()                                               #Gate will close
    Gate_state = 0                                             #Gate state will become 0


Entrance_State = GPIO.input(Entrance)    #set Entrance_State
Exit_State = GPIO.input(Exit)

if Entrance_Statepre == True and Entrance_State == False:   #if car begin to block the LDR at Entrance ...
    counter += 1                                           #increment counter
    if counter >10:                                        #if number of cars inside is above 10...
        counter = 10                                       #counter will remain 10

    setBits(counter)

elif Exit_Statepre == True and Exit_State == False:        #if car about to clear LDR at Exit
    counter -= 1                                           #decrement counter
    if counter <0:                                         #if count value is calculated to be below 0
        counter = 0                                        #counter will remain 0 in Led Bar Module

    setBits(counter)

Entrance_Statepre = Entrance_State  #as it goes as a loop, update "Entrance_Statepre" as equal to current Entrance LDR state
Exit_Statepre = Exit_State          #as it goes as a loop, update "Exit_Statepre" as equal to current Exit LDR state
```

Luminaires (Phoebe) **:**



LDR to V0+
- enable “Low Alarm”
- Voltage Value: 3v

LED to RL0+
- RL0+ mapping channel is V0+
- Trigger mode: ‘Low Alarm’



WISE data of Luminaires sent to private server…

E) Problems Encountered and Solutions

Problem: While setting up the circuit, we encountered some logical errors, especially for the scenario "Weather Conditions" and other requirements involving Led Bar Module.

Solution: Though we were worried, we made sure to not panic and focus on finding the issue. For codes with logical errors, I used the “debug” function to spot out the error. For codes involving Led Bar Module, I researched more to understand the basic functionality.

Problem: During the term break, when the lab was closed, I wasn’t able to test out all my coding as I only had limited sensors. Hence, not being able to test my coding was a struggle.

Solution: I found some websites that act as a raspberry pi simulator. Though the functions are different compared to working on actual raspberry pi, I was glad I got to check my code.

Problem: After ensuring the code works, we had to set up the actuators and sensors in a certain manner for recording. Though we didn’t have any logical errors, the output for “Gantry” didn’t work to our expectations due to the motor not operating properly. Since the speed of a motor depends on the weight, my group mate and I had a struggle to control the motor speed to our preference.

Solution: We kept rearranging the accelerometer, the motor and other elements attached to the motor. This reduced the weight the motor was carrying and allowed it to function properly. Though it took us more than an hour, I was glad my teammate and I finally managed to get the motor work according to our expectations and were satisfied with the result of the recording.

F) Conclusion

What I learnt:

✍ This project has really been an instructive experience for me. The brainstorming of ideas was not just enjoyable but also informative as I had to research on actuators, sensors, and Raspberry Pi before finalizing the ideas.
Moreover, I've also learnt more about Python programming, Raspberry Pi functions and controller unit connectivity during this project.

✍ Besides technological stuff, I've also learned teamwork and determination while working with my group mate Phoebe. Just the two of us working on a four-man project felt scary at first, however, we pushed through and support each other when one feels overwhelmed, and finally managed to finish the project.

Acknowledgement:

☺ Firstly, I would like to thank the lecturers who were involved in implementing this project. The experience has given us more confidence in Python programming and brainstorming technological ideas for IoT projects. I believe it will help us a lot in understanding IoT, solving logical programming problems and completing future projects successfully.

☺ Secondly, I would like to thank the lecturers in my class who guided us through this DVA module. To name, Mr. Lee Boon Liang and Mr. Melvin who taught us the operating system of some sensors and actuators.

☺ Secondly, I would like to thank my group mate Phoebe for helping me during difficult times. As we were overwhelmed with multiple projects, we shared each other's workload to keep a balance. Phoebe and I had gained insightful knowledge and confidence in controller unit circuitry.

Please watch this video to view the project operating

https://youtu.be/5HCgFc7b01A