|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | **EGE205**  **CONNECT SYSTEM DESIGN PROJECT**  **Temperature Control System**  **ASSIGNMENT**  **(INDIVIDUAL)** | | |  |
|  |  |  |  |  |
|  | Ong Zachary  221252K | | |  |
|  |  |  |  |  |

|  |
| --- |
| **Part 1: Assignment Criteria and Constraints** |
| *Instruction:*  *Write your assignment criteria and constraints as agreed in your team discussion.* |
| This assignment is to develop a prototype smart thermostat Control System as part of the subsystem of a Connected Home System. Below are the list of criteria and constraints that define this system.  **Criteria:**  The system consists of a PC web server and BBBW board web client to enable centralized control of multiple air conditioning units in a house. The key features of the system include:   1. Web-based application:  Allows users to control temperature and state of the air conditioning units, set timers for turning them on and off, and view room and unit temperatures. 2. Human presence detection:  Utilizes an IR Eclipse sensor to detect human presence and control the air conditioning unit, accordingly, enabling automatic turning on or off. 3. Voice control:  Incorporates voice recognition capabilities to control the air conditioning unit by voice commands, and differentiate users based on their unique voice and speech patterns. 4. Environmental sensing:  Utilizes an environment sensor click to detect room temperature and humidity, allowing the system to optimize energy usage and cooling by automatically adjusting the air conditioning unit based on predefined thresholds. 5. 7-segment display:  Displays the current state of the air conditioning unit (on or off) on a 7-segment display as well as temperature.   By implementing these functionalities, the system ensures efficient and convenient control of air conditioning units while promoting energy-saving practices.  **Constraints:**   * The system lacks the necessary hardware to control the temperature of the air conditioning unit as it does not have the required sensors. * The system lacks voice pattern recognition capabilities due to limitations in the microphone, which can only detect sound. * The temperature and state of the air conditioner are simulated since actual measurements are not available. * The IR Eclipse, due to its small form factor, is unsuitable for directly detecting humans. However, it will be utilized to simulate human detection. |

|  |
| --- |
| **Part 2: Block Diagram** |
| *Instruction:*  *Draw the block diagram of your system.*  *Resource:*  [*https://www.smartdraw.com/block-diagram/*](https://www.smartdraw.com/block-diagram/)  [*https://www.lucidchart.com/*](https://www.lucidchart.com/)  [*https://creately.com/*](https://creately.com/) |
|  |

|  |
| --- |
| **Part 3: Schematic / Circuit Diagram** |
| *Instruction:*  *Draw the schematic or circuit diagram of your system.*  *Resource:*  [*https://en.wikipedia.org/wiki/Schematic*](https://en.wikipedia.org/wiki/Schematic)  [*https://en.wikipedia.org/wiki/Circuit\_diagram*](https://en.wikipedia.org/wiki/Circuit_diagram) |
| Note: Some Click boards have swapped slots due to issues  7 Segment LED is now in slot 3  IR Eclipse is now slot 4 Environmental Sensor is now in slot 2 |

|  |
| --- |
| **Part 4: Flow Chart** |
| *Instruction:*  *Draw the flow chart of your program.*  *Resource:*  [*https://en.wikipedia.org/wiki/Flowchart*](https://en.wikipedia.org/wiki/Flowchart)  [*https://www.geeksforgeeks.org/an-introduction-to-flowcharts/*](https://www.geeksforgeeks.org/an-introduction-to-flowcharts/)  [*https://www.edrawsoft.com/create-programming-flowchart.html*](https://www.edrawsoft.com/create-programming-flowchart.html)  [*https://cacoo.com/examples/flowchart-software*](https://cacoo.com/examples/flowchart-software)  [*https://www.edrawsoft.com/flowchart-maker.html*](https://www.edrawsoft.com/flowchart-maker.html) |
| BBBW Web Client    PC Web Server    PC Web Page |

|  |
| --- |
| **Part 5: Program Code** |
| *Instruction:*  *Include the code of your system.* |
| BBBW Web Client  import time  import board  import socketio  import adafruit\_bme680 as ENV  import Adafruit\_BBIO.ADC as ADC  import Adafruit\_BBIO.GPIO as GPIO  from Adafruit\_BBIO.SPI import SPI  sio = socketio.Client()  def SevenSegInit():  GPIO.setup("P9\_16", GPIO.OUT)  GPIO.setup("P9\_23", GPIO.OUT)  GPIO.output("P9\_16", GPIO.HIGH)  GPIO.output("P9\_23", GPIO.HIGH)  L\_Spi0 = SPI(0, 0)  L\_Spi0.mode = 0  return L\_Spi0  def SevenSegDisplay(L\_Spi0, L\_Number):  DigitList = [0x7E, 0x0A, 0xB6, 0x9E, 0xCA, 0xDC, 0xFC, 0x0E, 0xFE, 0xDE]  OnesDigit = L\_Number % 10  TensDigit = L\_Number / 10  L\_Spi0.writebytes([DigitList[int(OnesDigit)], DigitList[int(TensDigit)]])  def SETUP():  ADC.setup()  board.I2C()  GPIO.setup("P8\_17", GPIO.IN)  GPIO.setup('USR0', GPIO.OUT)  GPIO.setup('USR1', GPIO.OUT)  GPIO.setup('USR2', GPIO.OUT)  GPIO.setup('USR3', GPIO.OUT)  def BIG\_X():  x = ADC.read("P9\_38")  BG\_NOISE = x \* 49.94007191  X = round(BG\_NOISE, 1)  return X  def VOICE(State):  Voice = 0  print("Digital Value: %f" % XX)  print("Background Noise: %f" % XX)  if XX >= 0.5:  Voice = XX  print("Voice: %f" % Voice)  if not State and 0.6 <= Voice <= 0.7:  print("Turn on Air Conditioner")  State = True  if State:  print("Air Conditioner is on")  time.sleep(5)  elif State and 0.7 <= Voice <= 0.8:  print("Turn off Air Conditioner")  State = False  if not State:  print("Air Conditioner is off")  time.sleep(5)  time.sleep(0.5)  return State  def T\_NUM():  T\_Offset = 0  T = ENVIRONMENT.temperature + T\_Offset  t = round(T)  try:  sio.emit('Room', {'data': t})  print('Data sent!')  except:  pass  print('Unable to transmit data.')  return t  def H\_NUM():  H\_Offset = 0  H = ENVIRONMENT.relative\_humidity + H\_Offset  h = round(H)  return h  def T\_Detect(State):  try:  print("Temperature of room: %i°C" % TT)  print(f"Humidity of room: {HH}%")  if not State and HH >= 40 and TT >= 25:  print("Turn on Air Conditioner")  State = True  if State:  print("Air Conditioner is on")  sio.emit('State', {'data': State})  print('Data sent!')  time.sleep(3)  elif State and HH <= 40 and TT <= 25:  print("Turn on Eco mode")  cold = True  if cold:  print("Set Air Conditioner to Eco mode")  sio.emit('Eco State', {'data': cold})  time.sleep(3)  time.sleep(0.5)  except:  pass  print('Unable to transmit data.')  return State  def TRIP(Count):  try:  y = GPIO.input("P8\_17")  if y == 1:  Count += 1  print("Human is Detected")  print(f"No. of people in room: {Count}")  sio.emit('Count', {'data': Count})  print('Data sent!')  time.sleep(2)  if Count == 1:  State = True  print("Turn on Air Conditioner")  if State:  print("Air Conditioner is on")  sio.emit('State', {'data': State})  print('Data sent!')  except:  pass  print('Unable to transmit data.')  return Count  @sio.event  def connect():  print('Connection established.')  GPIO.output('USR0', GPIO.HIGH)  GPIO.output('USR1', GPIO.HIGH)  GPIO.output('USR2', GPIO.HIGH)  GPIO.output('USR3', GPIO.HIGH)  @sio.event  def connect\_error(data):  print("The connection failed!")  @sio.event  def disconnect():  print('Disconnected from server.')  GPIO.output('USR0', GPIO.LOW)  GPIO.output('USR1', GPIO.LOW)  GPIO.output('USR2', GPIO.LOW)  GPIO.output('USR3', GPIO.LOW)  while True:  try:  sio.connect('http://192.168.0.103:5000')  break  except:  print("Try to connect to the server.")  pass  SETUP()  Count = 0  State = False  G\_Number = 0  G\_Spi0 = SevenSegInit()  while True:  I2C = board.I2C()  ENVIRONMENT = ENV.Adafruit\_BME680\_I2C(I2C, 0x77)  XX = BIG\_X()  TT = T\_NUM()  HH = H\_NUM()  Count = TRIP(Count)  G\_Number = TT  if Count == 0:  print(f"No. of people in room: {Count}")  print("AIR CONDITIONER IS OFF")  time.sleep(5)  elif Count >= 1:  State = T\_Detect(State) and VOICE(State)  SevenSegDisplay(G\_Spi0, G\_Number)  PC Web Server  from flask import Flask  from flask import render\_template  from flask\_socketio import SocketIO  from flask\_socketio import emit  app = Flask(\_\_name\_\_)  socketio = SocketIO(app)  @app.route('/')  def index():  return render\_template('index.html')  @socketio.event  def Room(RxData):  socketio.emit('Web\_Room', RxData)  print('Receive Data from Room')  @socketio.event  def State(RxData):  socketio.emit('Web\_state', RxData)  @socketio.event  def Eco\_State(RxData):  socketio.emit('Web\_Eco\_State', RxData)  @socketio.event  def Count(RxData):  socketio.emit('Web\_Count', RxData)  if \_\_name\_\_ == '\_\_main\_\_':  app.run(host='192.168.0.103')  PC Web Page  <html>  <head>  <meta charset="UTF-8">  <title>Smart Thermostat</title>  <meta http-equiv="x-ua-compatible" content="IE=edge">  <meta name="viewport" content="width=device-width, initial-scale=1" />  <link rel='stylesheet' href="{{ url\_for('static', filename='css/style.css') }}">  <!Use of Javascript library socket.io.js to connect to the python web server––>  <script src='https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js'></script>  <script src='https://cdnjs.cloudflare.com/ajax/libs/socket.io/3.1.2/socket.io.js'></script>  <script type='text/javascript'>  $(document).ready(function(){  var socket = io.connect('http://192.168.0.103:5000');  socket.on('Web\_Room', function(RxData) {  $(Tempreture).text(RxData.data);  });  socket.on('Web\_state', function(RxData) {  $(State).text(RxData.data)  });  socket.on('Web\_Eco\_State', function(RxData) {  $(Eco).text(RxData.data)  })  socket.on('Web\_Count', function(RxData) {  $(Count).text(RxData.data)  })  });  </script>  </head>  <body>  <main>  <div class="HEADER">  <h1>Smart Thermostat</h1>  </div>  <p class="DDD">Temperature of room is: <Tempreture>???</Tempreture></p>  <p>Air Conditioner: <State>???</State></p>  <p>Eco Mode: <Eco>???</Eco></p>  <p>No. of people in room: <Count>???</Count></p>  </main>  </body> |