SMART HOUSE SYSTEM

Hardware and Software implementation of the system

ABSTRACT

The project focuses on developing a comprehensive smart home system capable of managing lighting, temperature, and audio volume based on sensor data and computer vision inputs. The implementation involves both hardware and software components. The hardware design is created using Altium Designer, while the software implementation is done using the Arduino IDE. Additionally, computer vision capabilities are integrated into the system using Python. The ultimate goal is to create an efficient and user-friendly smart home system that enhances convenience and automation in daily living.

Name	ID
Shrouk Abdelkhalek	58
Rewan Radwan Ibrahim Elbeshbeshy	53
Youssef Mohammed Salah	63
Yassen Islam Mohamed	60
Mahmoud Ibrahim Talaat	27

Smart House System

Contents

Introduction

- This technical report presents the design and development of a Printed Circuit Board (PCB) using Altium Designer, the implementation of software using C/Arduino, and the integration of computer vision tasks using Python. The objective of this project is to create a functional smart home system that can effectively control lighting, temperature, and audio volume based on sensor readings and computer vision inputs.

- The rapid advancement of technology has led to the emergence of smart home systems, which offer convenience, energy efficiency, and enhanced living experiences. Our project aims to contribute to this field by designing and developing a PCB that serves as the central control unit for the smart home system. The PCB will be responsible for processing sensor data, executing control algorithms, and communicating with various components of the system.

- **The software implementation will be carried out using C/Arduino**, a popular programming language and platform for embedded systems. Arduino provides a user-friendly and versatile environment for developing software that can interact with the hardware components of the smart home system. By leveraging the capabilities of Arduino, we will be able to efficiently control the lighting, temperature, and audio volume based on the sensor readings.

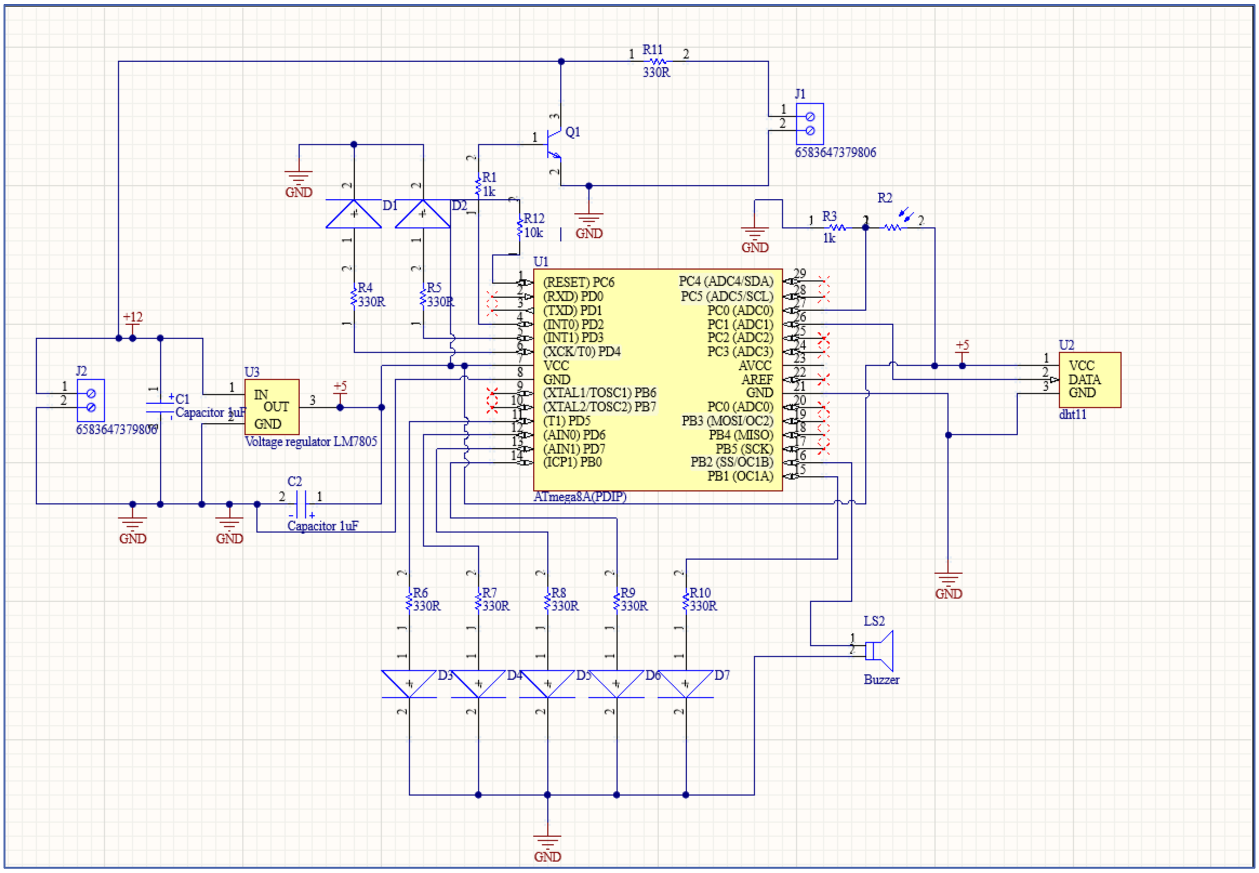
- **In addition to sensor-based control**, we will integrate computer vision tasks using Python. Computer vision enables the system to interpret visual inputs and make intelligent decisions based on them. By utilizing computer vision algorithms, we can enhance the functionality of the smart home system, allowing it to detect and respond to specific gestures or movements.

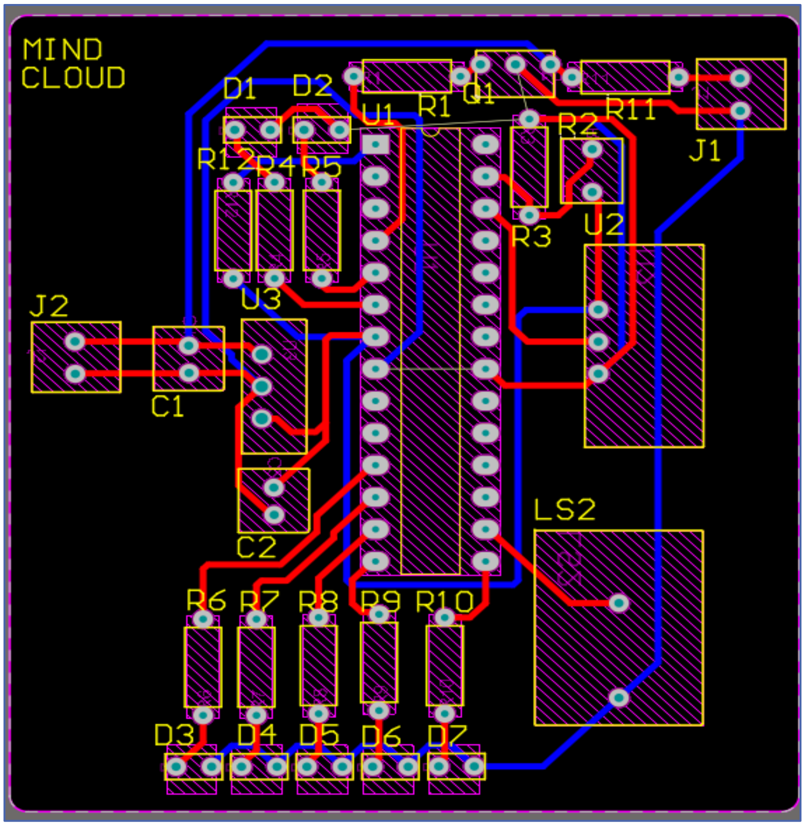
- **Throughout this report**, we will discuss the hardware design process, including the selection of components and the creation of the PCB using Altium Designer. We will also delve into the software implementation, detailing the programming techniques used to control the system based on sensor readings and computer vision inputs. Furthermore, we will explore the integration of computer vision tasks using Python and the overall user experience of the smart home system.

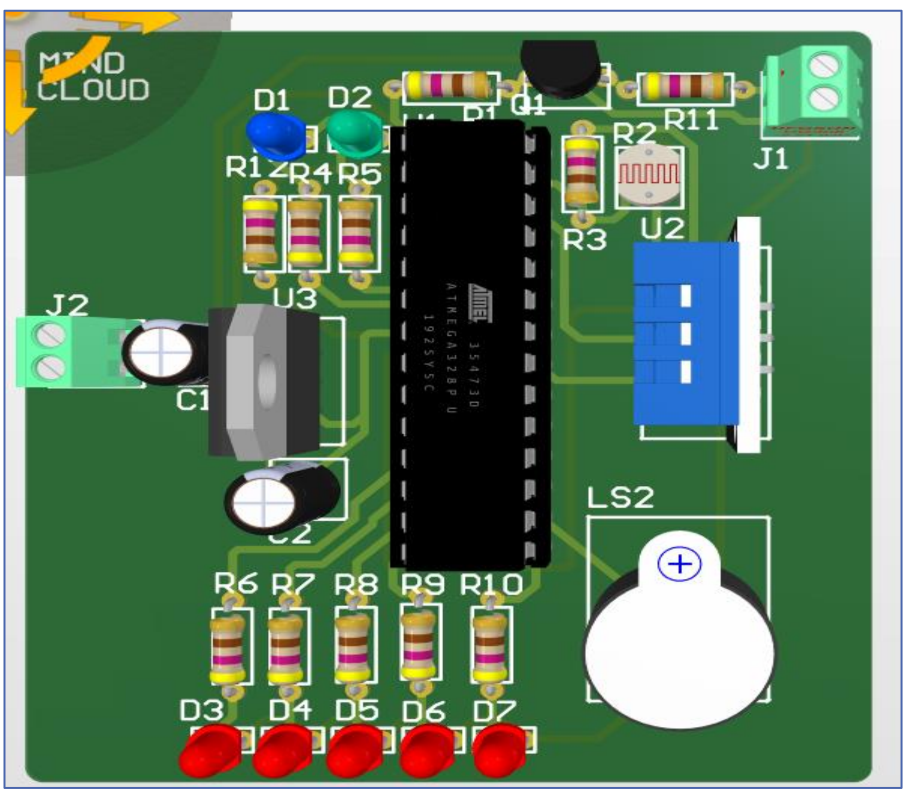
- **By the end of this project**, we aim to demonstrate a fully functional smart home system that effectively controls lighting, temperature, and audio volume based on sensor readings and computer vision inputs. This report will provide a comprehensive overview of the design, development, and integration processes, as well as insights into the future potential and improvements of the system.

Hardware Implementation

The Schematic Diagram in Altium Designer




MIND
CLOUD
D1
D2
U1
R1
Q1
R11
R2
J1
R12
R4
R5
R3
U2
J2
U3
C1
C2
LS2
R6
R7
R8
R9
R10
D3
D4
D5
D6
D7


MIND
CLOUD
D1
D2
Q1
R11
R2
J1
R12
R4
R5
R3
U2
J2
U3
C1
ATMEGA328P U
LS2
R6
R7
R8
R9
R10
D3
D4
D5
D6
D7

Design Rule Check

Designer

Design Rule Verification Report

Date:	10/6/2023
Time:	3:26:04 PM
Elapsed Time:	00:00:01
Filename:	C:\Users\shrouk qassem\AppData\Local\Temp\Temp8d8da5be-1732-4bc1-8466-56d9febabf47_Smart Home System(Final project).zip\Smart Home System(Final project)\PCB2.PcbDoc

Warnings:	0
Rule Violations:	116

Summary

Warnings	Count
Total	**0**

Rule Violations	Count
Clearance Constraint (Gap=0.6mm) (All),(All)	0
Short-Circuit Constraint (Allowed=No) (All),(All)	0
Un-Routed Net Constraint ((All))	3
Modified Polygon (Allow modified: No), (Allow shelved: No)	0
Width Constraint (Min=0.6mm) (Max=3mm) (Preferred=1mm) (All)	0
Power Plane Connect Rule(Relief Connect)(Expansion=0.508mm) (Conductor Width=0.254mm) (Air Gap=0.254mm) (Entries=4) (All)	0
Hole Size Constraint (Min=0.025mm) (Max=2.54mm) (All)	0
Hole To Hole Clearance (Gap=0.254mm) (All),(All)	0

Rule Violations	Count
Clearance Constraint (Gap=0.6mm) (All),(All)	0
Short-Circuit Constraint (Allowed=No) (All),(All)	0
Un-Routed Net Constraint ((All))	3
Modified Polygon (Allow modified: No), (Allow shelved: No)	0
Width Constraint (Min=0.6mm) (Max=3mm) (Preferred=1mm) (All)	0
Power Plane Connect Rule(Relief Connect)(Expansion=0.508mm) (Conductor Width=0.254mm) (Air Gap=0.254mm) (Entries=4) (All)	0
Hole Size Constraint (Min=0.025mm) (Max=2.54mm) (All)	0
Hole To Hole Clearance (Gap=0.254mm) (All),(All)	0
Minimum Solder Mask Sliver (Gap=0.254mm) (All),(All)	2
Silk To Solder Mask (Clearance=0.254mm) (IsPad),(All)	98
Silk to Silk (Clearance=0.254mm) (All),(All)	13
Net Antennae (Tolerance=0mm) (All)	0
Height Constraint (Min=0mm) (Max=25.4mm) (Prefered=12.7mm) (All)	0
Total	**116**

Layout Diagram

Physical Connection

Software Implementation

Arduino IDE

The provided code is an implementation for adjusting the intensity of light in a room using an LDR (Light Dependent Resistor) sensor and an Arduino microcontroller. Here is a brief explanation of the code:

1. Importing Libraries:

- **DHT**: for interfacing with the DHT11 temperature and humidity sensor.

```

// Import DHT11 sensor Library
#include <DHT.h>
```

2. Pin Assignments:

- **ldrPin**: the analog pin connected to the LDR sensor.

- **fanPin**: the digital pin connected to control the fan.

- **DHTPIN**: the analog pin connected to the DHT11 sensor.

- **greenLedPin**: the digital pin connected to the green LED.

- **blueLedPin:** the digital pin connected to the blue LED.

- **buzzerPin**: the digital pin connected to control the buzzer.

- **redLedPin**: the digital pin connected to the red LED.

```
// Pin assignments
#define ldrPin A0   // LDR pin
#define fanPin 2    // Fan control pin
#define DHTPIN A1   //DHT Temperature sensor pin
#define DHTTYPE DHT11
#define greenLedPin 3   // Green LED pin
#define blueLedPin 4    // Blue LED pin
#define buzzerPin 5     // Buzzer control pin
#define redLedPin 6     // Red LED pin
```

3. Variable Initialization:

- **ldrValue**: stores the value read from the LDR sensor.

- **temperature**: stores the temperature value read from the DHT11 sensor.

- **dht**: an instance of the DHT class to handle DHT data.

- **serialObject**: an instance of the SoftwareSerial class for serial communication.

```

// Variables
int ldrValue = 0;          // LDR value
float temperature = 0;     // Temperature value
DHT dht(DHTPIN, DHTTYPE);  // variable to handle to DHT data
```

4. Setup Function:

- Initializes serial communication with the computer and the microcontroller.

- Initializes the DHT sensor.

- Sets the pin modes for the fan, LEDs, and buzzer.

```
void setup() {
  // Initialize serial communication
  Serial.begin(9600);
  dht.begin();

  // Set pin modes
  pinMode(fanPin, OUTPUT);
  digitalWrite(fanPin, LOW);  // Turn off the fan

  pinMode(greenLedPin, OUTPUT);
  pinMode(blueLedPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  pinMode(redLedPin, OUTPUT);
}
```

Functions Implemented

```
void loop() {
  //LDR reading and adjusting green LED function
  readLDR();
  // Blink the blue LED every 1 second
  blueBlink();

  // Read temperature value
  getTemperature();

  // Control buzzer volume  based on data from Python vision tasks
  ControlBuzzer();

  // Control  red LEDs based on data from Python vision tasks
  ControlRedLed();

  delay(1000);  // Delay for 1 second
}
```

readLDR()

```
void readLDR() {
  // Read LDR value
  ldrValue = analogRead(ldrPin);

  // Adjust brightness of green LED based on LDR readings
  int brightness = map(ldrValue, 0, 1023, 0, 255);
  analogWrite(greenLedPin, brightness);
  //Provide LDR data to the computer via serial communication
  Serial.print("°C, LDR Value: ");
  Serial.println(ldrValue);
}
```

blueBlink()

```
void blueBlink() {
  // Blink the blue LED every 1 second
  static unsigned long previousMillis = 0;
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= 1000) {
    previousMillis = currentMillis;
    digitalWrite(blueLedPin, !digitalRead(blueLedPin));
  }
}
```

getTemperature()

```
void getTemperature() {
  // Read temperature value
  temperature = dht.readTemperature();
  if (isnan(temperature)) {
    Serial.println("Failed to read temperature from DHT sensor!");
  } else {
    // Turn on/off the fan based on temperature
    if (temperature > 25) {
      digitalWrite(fanPin, HIGH);  // Turn on the fan
    } else {
      digitalWrite(fanPin, LOW);  // Turn off the fan
    }
    // Provide temperature to the computer via serial communication
    Serial.print("Temperature: ");
    Serial.print(temperature);
  }
}
```

Computer Vision

```
import math
import serial
import cv2
import mediapipe as mp
import numpy as np

# change the light bulb intensity
def Light_change(intensity,serialObject):
    serialObject.write(intensity)
```

```python
hand_draw = mp.solutions.drawing_utils
# Create a MediaPipe Hands object.
hands = mp.solutions.hands.Hands(
    max_num_hands=1,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5)

HandMin=50  # hand range (50,300)
HandMax=300
lighBulbMin=0  # 0Volt == 0
lighBulbMax=1023 #5Volt == 1023
# Create a video capture object.
cap = cv2.VideoCapture(0)
```

```python
while True:
    # Capture the next frame from the video capture object
    ret, frame = cap.read()

    # If the frame is empty, break the loop.
    if not ret:
        break

    # Convert the frame to RGB format.
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Process the frame with MediaPipe Hands.
    results = hands.process(rgb_frame)

```

```python

    # Get the landmarks for the index and thumb fingers.
    index_finger_landmark = results.multi_hand_landmarks[0].landmark[mp.solutions.hands.HandLandmark.INDEX_FINGER_TIP]
    middle_finger_landmark = results.multi_hand_landmarks[0].landmark[mp.solutions.hands.HandLandmark.MIDDLE_FINGER_TIP]

    # Draw a line between the two landmarks.
    cv2.line(frame, (index_finger_landmark.x, index_finger_landmark.y), (middle_finger_landmark.x, middle_finger_landmark.y), (0, 0, 255), 2)

    # If hand landmarks are detected, draw them on the frame.
    if results.multi_hand_landmarks is not None:
        for hand in results.multi_hand_landmarks:
            hand_draw.draw_landmarks(frame, hand, mp.solutions.hands.HAND_CONNECTIONS)
            # get the points of the two fingers
            finger1_landmarks = results.multi_hand_landmarks[0].landmark[ mp.solutions.hands.HandLandmark.INDEX_FINGER_TIP]

            finger2_landmarks = results.multi_hand_landmarks[0].landmark[mp.solutions.hands.HandLandmark.THUMB_TIP]

            # Draw a line between the two fingers.
            cv2.line(frame, (finger1_landmarks.x, finger1_landmarks.y), (finger2_landmarks.x, finger2_landmarks.y),(0, 0, 255), 2)

            # measure the length of the line between two points

            line_Length=math.hypot(finger1_landmarks,finger2_landmarks)

            # covert hand range intp the light output range
            Light_intensity=np.interp(line_Length,[HandMin,HandMax],[lighBulbMin,lighBulbMax])

            # change the light bulb intensity
            Light_change(Light_intensity, serialObject)

    # Display the frame.
    cv2.imshow('Hand Detection', frame)
```

```python
    if cv2.waitKey(30) & 0xFF == ord('x'):
        break

# Release the video capture object.
cap.release()

# Close all windows.
cv2.destroyAllWindows()

# Release the video capture object.
cap.release()

# Close all windows.
cv2.destroyAllWindows()
```

```python
# adjust the intensity of the light in the room using LDR

import serial

 # to use the serial module

import time

serialObject = serial.Serial("COM13", baudrate=9600, timeout=1)
 # making an object from the connected board

def getLight():

    MicroControllData = serialObject.readline().decode('ascii')

    # read the received data  and store the ascii code without symbols with the use of function decode


    return int(MicroControllData)


```

```
while (1):
    time.sleep(2)

    LightiItensity=getLight()

    print(f"intensity of light ion the room is ")

    if (LightiItensity < 100):
        # if the light intensity less than 100 the LED should be turned on

        serialObject.write(b'1')

        # turn on the LED

    if (LightiItensity>100):

        # if the light intensity more than 100 the LED should be turned off

        serialObject.write(b'0')

        #turnoff the LED

    break
```

```
while (1):
        time.sleep(2)

        temperature=((getTemp())*100)

        print(str(temperature)+"°C")

```

Integration and User control