TEAM 6

Final Project Report

Satellite System Simulation

Team Member Name	ID
Ibrahim Mohamed Ibrahim	1216E
Mahmoud Fathy Mohamed	1214E
Omar Fathi Younis	1254E
Fatma Mostafa Mohamed	1265E
Malak Ibrahim Mahmoud	1346E

1 Introduction

The objective of this project was to design and implement a complete satellite and ground station simulation. The system consists of an STM32 micro-controller to simulate a satellite that gathers sensor data, and a ground station for displaying the data, controlling the satellite system and data logging.

Key Features

- Real-time monitoring of temperature, light intensity, and object proximity.
- A continuously sweeping radar system for collision avoidance that works in parallel with the other functions.
- A Python-based Graphical User Interface (GUI) for the ground control station.
- Voice-activated on/off control using a custom-trained model.
- Sensor data logging.

2 System Design

2.1 Communication

The communication between the STM32 and the python scripts is via a serial port over a USB connection at a baud rate of 9600.

- **Satellite to Ground Station:** The STM32 sends a every interval of time the comma-separated data of the sensors in the format: `TEMPERATURE,DISTANCE,LIGHT_STATE`. For example: `24.5,15.2,1`.
- **Ground Station to Satellite:** The Python application sends single-character commands to control the satellite's operational state:
  - `'1'`: Activates the STM32's system.
  - `'0'`: Deactivates the STM32's system.

3 Part 1: Satellite Subsystem (Hardware & Firmware)

3.1 Hardware Components

- STM32F103C6 "Blue Pill" Microcontroller
- DHT11 Temperature and Humidity Sensor
- Photoresistor (LDR) for light intensity measurement
- HC-SR04 Ultrasonic Sensor for object detection
- SG90 Micro Servo Motor for the radar scanner
- Various LEDs for status indication (Red, Green, White)
- USB TO TTL FTD to program the micro-controller

3.2 Hardware circuit

The sensors and actuators are connected to the STM32 as shown in the schematic below. The sensors are powered by 3.3V and the servo motor is powered by an external 5V supply with a common ground.

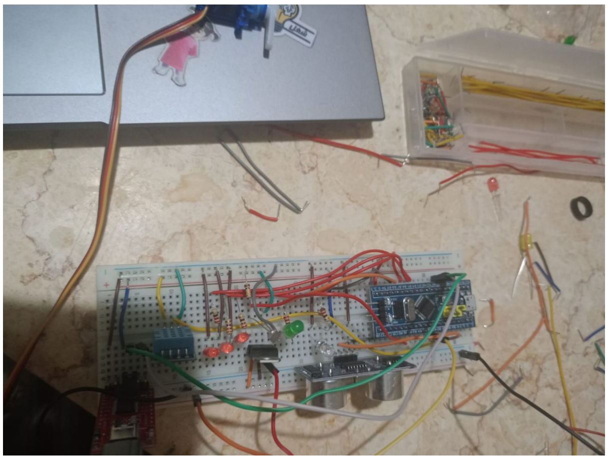
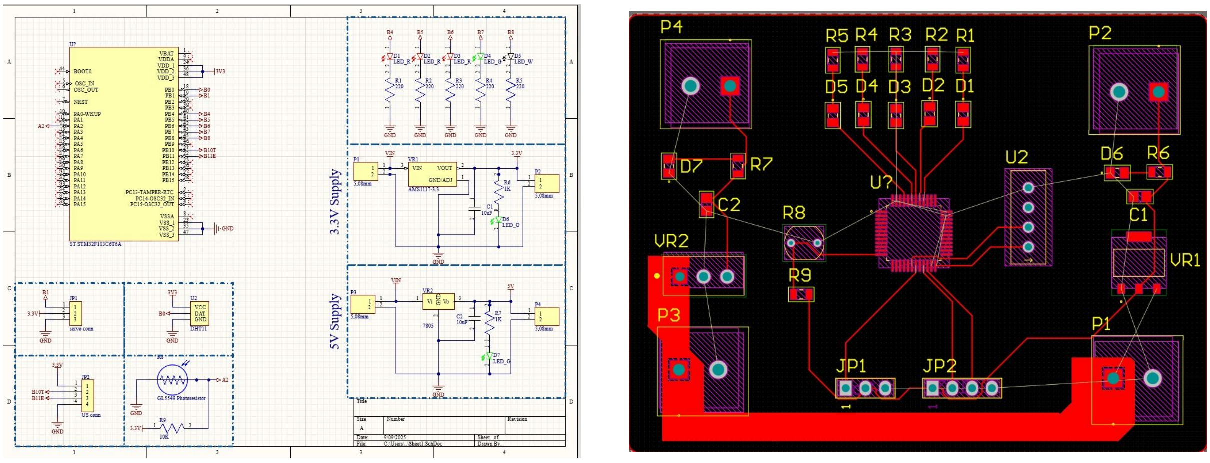


Figure 1: Hardware Circuit

3.3 PCB Design



Schematic of the system.

PCB layout of the system.

Figure 2: PCB design of the system.

3.4 Firmware Design

The firmware was developed in the Arduino IDE.

- **Non-Blocking Timers:** The code avoids the use of the `delay()` function. Instead, it uses the `millis()` function to create independent timers for reading the slow DHT11 sensor, sweeping the radar servo, and sending serial data packets. This ensures the main that every function works in parallel.

4 Part 2: Ground Station Subsystem (Software)

The ground station is a Python application that provides a graphical interface for monitoring and controlling the satellite.

4.1 Graphical User Interface (GUI)

A GUI was developed using Python's built-in Tkinter library. The interface provides real-time display of all sensor data and a visual alert for object proximity.

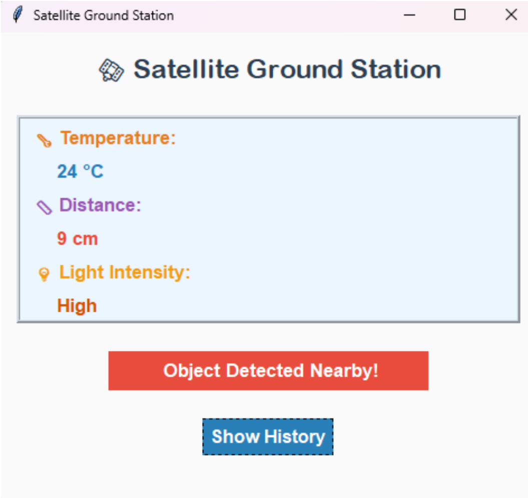


Figure 3: GUI of ground station

4.2 Voice Command System

A key feature is the ability to turn the satellite on and off using voice commands.

- **Custom Dataset:** A custom dataset was created by recording team members' voices saying "ON" and "OFF".
- **Feature Extraction:** The Librosa library was used to process the raw audio files and extract Mel-Frequency Cepstral Coefficients (MFCCs), which are numercal values that describe the shape of an audio wave, so that the model can understand them.
- **Model Training:** A Support Vector Machine (SVM) classifier from the Scikit-learn library was used to recognize the spoken words "ON" and "OFF".

4.3 Data Storage and Display

All incoming sensor data is logged. The GUI includes a "Show History" button that opens a new window and displays the historical data in a clean, tabular format.

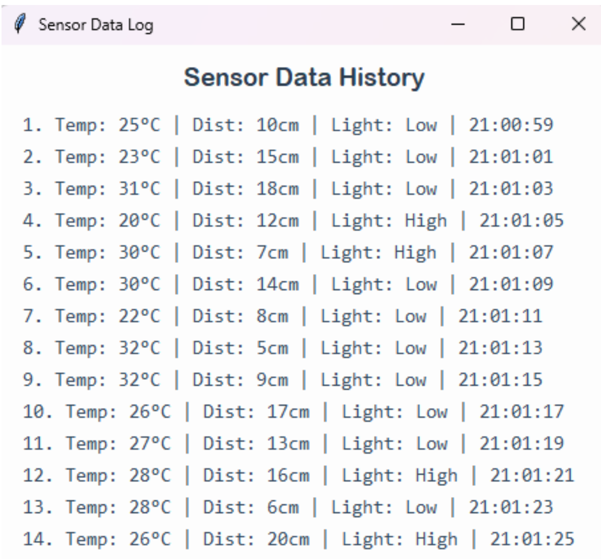


Figure 4: Sensor data history

A Source Code

A.1 STM32 Firmware (Arduino C++)

```
#include <Servo.h>
#include <DHT.h>


Servo radarservo;
#define Servo_signal PA7
#define trig PA8
#define echo PA9

#define red1 PB4
#define red2 PB5
#define red3 PB6
#define green PB7
#define white PB8

const int allPins[] = {red1, red2, red3, green, white, trig};

#define DHTPIN PB0
#define DHTTYPE DHT11
DHT dht11(DHTPIN, DHTTYPE);

#define LDR PA2

int system_state = 0; // 0: OFF, 1: ON

float distance_time ,distance;
float celsius = 0;
int light;


unsigned long radar_prev_time = 0;
int radar_index = 0, radar_step = 10;


unsigned long temp_prev_time = 0;
unsigned long data_prev_time = 0;

void setup(){
  radarservo.attach(Servo_signal);
  pinMode(trig,OUTPUT);
  pinMode(echo,INPUT);
  pinMode(red1,OUTPUT);
  pinMode(red2,OUTPUT);
  pinMode(red3,OUTPUT);
  pinMode(green,OUTPUT);
  pinMode(white,OUTPUT);
  pinMode(DHTPIN,INPUT);
  pinMode(LDR,INPUT);
  dht11.begin();
  Serial.begin(9600);

  shutdownAll();
}

void loop(){
  if (Serial.available() > 0) {
    char cmd = Serial.read();
    if (cmd == '1') {
      system_state = 1;
    } else if (cmd == '0') {
      system_state = 0;
      shutdownAll();
    }
  }

  if(system_state == 1){ // checks if the system is ON

    unsigned long current = millis();
```

```
    if(current - temp_prev_time >= 2000){ // Waits two seconds to read the temperature sensor
      temp_prev_time = current;
      float new_temp = dht11.readTemperature();

      if(!isnan(new_temp)){ // Handles temperature sensor error reading of NaN
        celsius = new_temp;
      }
    }

    distance = Read_Distance();
    light = light_intensity(); // 0: Low, 1: High

    control_led(celsius);

    Radar_system(distance);

    if (current - data_prev_time >= 1000) { // wait one second to send data
      data_prev_time = current;
      send_data(celsius, distance, light);
    }

  }

}



void control_led(float temp_val){
   if (temp_val<20){
    digitalWrite(red1,HIGH);
    digitalWrite(red2,LOW);
    digitalWrite(red3,LOW);
  }
  else if(temp_val>30){
    digitalWrite(red1,HIGH);
    digitalWrite(red2,HIGH);
    digitalWrite(red3,HIGH);
  }
  else{
    digitalWrite(red1,HIGH);
    digitalWrite(red2,HIGH);
    digitalWrite(red3,LOW);
  }
}



float light_intensity(){
  float light_val=analogRead(LDR);

  if(light_val<2045){
    digitalWrite(white,HIGH);
    light = 0; // low light intensity
  }else{
    digitalWrite(white,LOW);
    light = 1; // high light intensity
  }
  return light;
}

float Read_Distance(){
  digitalWrite(trig,LOW);
  delayMicroseconds(2);
  digitalWrite(trig,HIGH);
  delayMicroseconds(10);
  digitalWrite(trig,LOW);
  distance_time=pulseIn(echo,HIGH);
  distance=0.0343*(distance_time/2);
  return distance;

}

```

```
void Radar_system(float distance){

  unsigned long radar_current_time = millis();


  if(radar_current_time - radar_prev_time >= 100){
    radar_prev_time = radar_current_time;
    radarservo.write(radar_index);

    if(distance>0&&distance<10){
      digitalWrite(green,HIGH);
    }else{
      digitalWrite(green,LOW);
    }

    radar_index+=radar_step;
    if(radar_index >= 180 || radar_index <= 0) {
      radar_step = -1*radar_step;
    }


  }

}

void send_data(float celsius, float distance, int light_state){

  Serial.print(String(celsius));
  Serial.print(",");
  Serial.print(String(distance));
  Serial.print(",");
  Serial.println(light_state);

}



void shutdownAll() {
  // Turn off all pins
  for (int i = 0; i < sizeof(allPins)/sizeof(allPins[0]); i++) {
    digitalWrite(allPins[i], LOW);
  }
  radarservo.write(0);  // Return servo to initial position
}
```

Listing 1: Final STM32 Firmware Code

A.2 Python Ground Station GUI

```
import tkinter as tk
from tkinter import ttk
import time
import random
import threading
import os
import librosa
import numpy as np
import sounddevice as sd
import serial
import pickle
# ------------------------------
# Load trained voice model
# ------------------------------
with open("voice_model.pkl", "rb") as f:
    clf = pickle.load(f)

# Feature extraction
def extract_features(audio, sr=16000):
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
    return np.mean(mfcc.T, axis=0)
# ------------------------------
# Record voice and send command
```

```
# ------------------------------
def record_and_send():
    duration = 2  # 2 sec recording
    sr = 16000
    print("      Say ON or OFF...")
    audio = sd.rec(int(duration * sr), samplerate=sr, channels=1, dtype='float32')
    sd.wait()
    audio = audio.flatten()

    features = extract_features(audio, sr)
    pred = clf.predict([features])[0]

    if pred == 1:
        print("    Detected: ON     Activating System")
        ser.write(b'1')  # send '1' to STM32
    else:
        print("    Detected: OFF     Shutting Down System")
        ser.write(b'0')  # send '0' to STM32

# Run voice command in a separate thread to avoid blocking GUI
def voice_command_thread():
    threading.Thread(target=record_and_send, daemon=True).start()

# ------------------------------
# Data History
# ------------------------------
data_history = []

# ------------------------------
# Read and Update GUI
# ------------------------------
def read_data():
    if ser.in_waiting > 0:
        line = ser.readline().decode().strip()
        try:
            t, d, l = line.split(",")
            temp_value.config(text=f"{t}  C ")
            distance_value.config(text=f"{d} cm")
            light_value.config(text="Low" if int(l) == 0 else "High")

            # Save to history
            data_history.append({
                "Temperature": t,
                "Distance": d,
                "Light": "Low" if int(l) == 0 else "High",
                "Time": time.strftime("%H:%M:%S")
            })

            # Status feedback for object detection
            if int(d) < 10:
                distance_value.config(fg="#e74c3c")
                status_label.config(
                    text=" Object Detected Nearby!",
                    fg="white",
                    bg="#e74c3c"
                )
            else:
                distance_value.config(fg="#27ae60")
                status_label.config(
                    text="Clear Space",
                    fg="white",
                    bg="#27ae60"
                )

        except Exception as e:
            print("Error:", e)

    root.after(2000, read_data)

# ------------------------------
# Show Past Readings
# ------------------------------
def show_history():
```

```
    history_win = tk.Toplevel(root)
    history_win.title("Sensor Data Log")
    history_win.configure(bg="#fdfdfd")

    tk.Label(
        history_win,
        text="Sensor Data History",
        font=("Arial Rounded MT Bold", 14),
        fg="#2c3e50",
        bg="#fdfdfd"
    ).pack(pady=10)

    for idx, data in enumerate(data_history):
        tk.Label(
            history_win,
            text=f"{idx+1}. Temp: {data['Temperature']} C  | Dist: {data['Distance']}cm | Light: {data['Light']} | {data['Time']}",
            fg="#34495e",
            bg="#fdfdfd",
            font=("Consolas", 11)
        ).pack(anchor='w', padx=15)

# ------------------------------
# GUI Setup
# ------------------------------
root = tk.Tk()
root.title("Satellite Ground Station")
root.geometry("480x500")
root.configure(bg="#fafafa")

# Title
tk.Label(
    root,
    text="        Satellite Ground Station",
    font=("Arial Rounded MT Bold", 18),
    fg="#2c3e50",
    bg="#fafafa"
).pack(pady=15)

# Frame for sensor data
frame = tk.Frame(root, bg="#ecf6ff", bd=3, relief="ridge")
frame.pack(padx=15, pady=10, fill="x")

# Temperature
tk.Label(frame, text="        Temperature:", font=("Arial", 13, "bold"), fg="#e67e22", bg="#ecf6ff").pack(anchor="w", padx=10, pady=5)
temp_value = tk.Label(frame, text="--  C ", font=("Arial", 13, "bold"), fg="#2980b9", bg="#ecf6ff")
temp_value.pack(anchor="w", padx=30)

# Distance
tk.Label(frame, text="        Distance:", font=("Arial", 13, "bold"), fg="#9b59b6", bg="#ecf6ff").pack(anchor="w", padx=10, pady=5)
distance_value = tk.Label(frame, text="-- cm", font=("Arial", 13, "bold"), fg="#27ae60", bg="#ecf6ff")
distance_value.pack(anchor="w", padx=30)

# Light Intensity
tk.Label(frame, text="        Light Intensity:", font=("Arial", 13, "bold"), fg="#f39c12", bg="#ecf6ff").pack(anchor="w", padx=10, pady=5)
light_value = tk.Label(frame, text="--", font=("Arial", 13, "bold"), fg="#d35400", bg="#ecf6ff")
light_value.pack(anchor="w", padx=30)

# Status
status_label = tk.Label(root, text="Waiting for updates...", font=("Arial", 13, "bold"), fg="white", bg="#95a5a6", pady=6, width=28)
status_label.pack(pady=15)

# Buttons
style = ttk.Style()
style.theme_use("clam")
style.configure("TButton", font=("Arial", 12, "bold"), padding=6, foreground="#fff",
```

```
    background="#2980b9", borderwidth=0)
style.map("TButton", background=[("active", "#1f618d")])

ttk.Button(root, text="Show History", command=show_history).pack(pady=5)
ttk.Button(root, text="       Voice Command", command=voice_command_thread).pack(pady=5)

# Start loop
read_data()
root.mainloop()
```

Listing 2: Final Python GUI Code