MIND CLOUD FINAL PROJECT

SATELLITE SYSTEM SIMULATION

NAME	ID
Rewan Khaled Mahrous	1270E
Dina Adel Shiha	1327E
Sofian Ebrahim	1211E
Youssef Ezzat Ramadan	1201E

TABLE OF CONTENTS

1. Introduction:

1.1. Project Overview and Objectives

This project is with two main segments: a satellite design and a ground station design. The objective is to create a functional simulation of a satellite system, which includes both a space segment (the satellite) and a ground segment (the control and user stations).

Satellite Design (First Part)

The satellite, which represents the space segment, must be built using an STM32F103C6T6 32-Bit ARM microcontroller ("The Blue Pill"). The system must not use an Arduino. The design must incorporate four essential functions:

- A temperature measurement and monitoring system.
- A light intensity measurement and monitoring system.
- A radar system for protecting the satellite from nearby objects.
- A transmitter and receiver unit (TX, RX).

The system must operate in parallel. The temperature monitoring system uses three red LEDs to indicate different temperature ranges. The light intensity system uses a photoresistor and a white LED. The radar system, which includes a servo motor and an ultrasonic sensor, uses a green LED to indicate object detection. The entire system is to be implemented on a breadboard, and a bonus is available for designing a PCB.

Ground Station Design (Second Part)

The ground station, representing the ground segment, is a Graphical User Interface (GUI) developed in Python using the Tkinter library. It serves as the "brain on Earth". The GUI must include the following functionalities:

- **Object Detection Alert:** An alert notification will be displayed on the GUI when the satellite's ultrasonic sensor detects an object.
- **Voice-Controlled System Power:** The system can be turned on or off using the voice commands "on" and "off". A custom machine learning model, trained with recordings from team members' voices, will be used. External datasets are not permitted.
- **Sensor Readings:** The GUI will allow users to request current temperature and light intensity readings from the satellite.
- **Data Storage and Retrieval:** All retrieved sensor readings will be stored in a dataset. Users must have the option to access and review this historical data through the GUI.

2. Hardware Design:

2.1Detailed description of the satellite module.

The satellite module is the "space segment" of the project, designed to simulate a satellite's core functions and operations. The project specifies the use of an

STM32F103C8T6 Blue Pill 32-Bit ARM microcontroller, also known as "The Blue Pill". It is explicitly stated that an Arduino cannot be used for this implementation. The satellite system must be implemented on a breadboard and must contain four key components and systems:

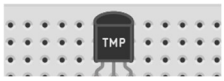
- **Temperature measuring and monitoring system**
- **Light intensity measuring and monitoring system**
- **Radar system** for object protection
- **Transmitter and receiver unit (TX, RX)**

2.2 Schematics and component list.

The following tools and components were used for the satellite design:

- **STM32F103C8T6 32-Bit ARM microcontroller** ("The Blue Pill")

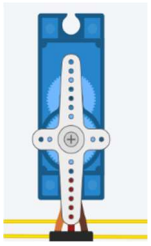
- **Temperature sensors**



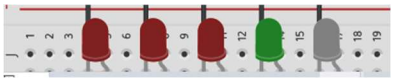
- **Light intensity sensor** (Photoresistor)

-
- **Ultrasonic sensor**

- **Servo motor**

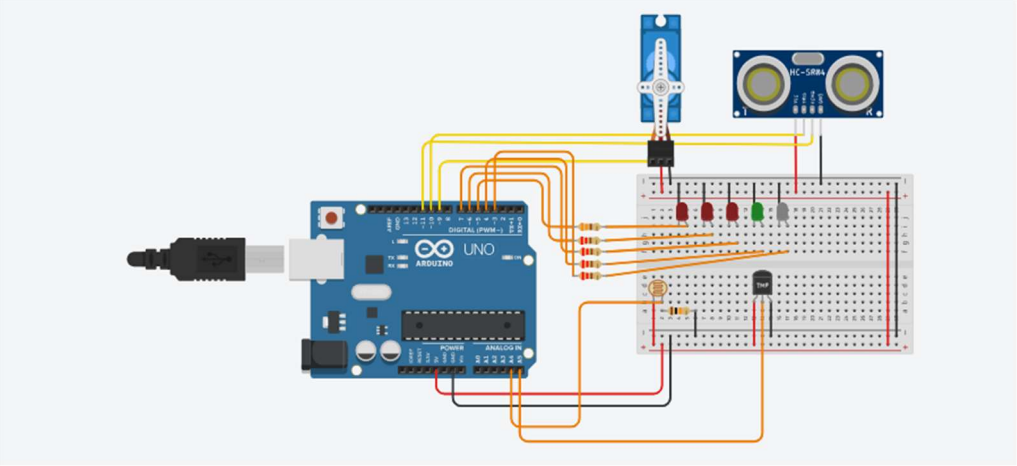


- **LEDs**: one white, one green, and three red



Schematic and link on Tinker cad:

https://www.tinkercad.com/things/ggbrBYqnIMn-mindcloudfinalproject/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard&sharecode=OMAS3ejBH6IO-nM3ctZCl0nmbksuNGY3Dsy42IuwUtw



2.3 Explanation of each sensor and its function.

2.3.1 Temperature Sensors

Explanation:

Temperature sensors measure the amount of heat or cold in an environment. They detect temperature changes and convert this information into data that can be read by electronic systems.

Function:

- Measure ambient temperature.
- Used in weather stations, smart thermostats, industrial machines, and electronic devices.
- Common types: **Thermistors**, **RTDs**, **Thermocouples**, and **digital temperature sensors** like the **DHT11** or **DS18B20**.

2.3.2 Light Intensity Sensor (Photoresistor)

Explanation:

A **photoresistor** (also known as a **Light Dependent Resistor** or **LDR**) is a sensor whose resistance changes based on the light intensity falling on it.

Function:

- Detect the level of light in the environment.
- Resistance decreases when light intensity increases (and vice versa)
- Used in:
    - Automatic street lights

- Light meters in cameras
- Solar garden lights
- Brightness control systems

2.3.3 Ultrasonic Sensor

Explanation:

An ultrasonic sensor measures distance using **ultrasound waves**. It emits a sound wave at a frequency higher than humans can hear and measures the time it takes for the echo to return after bouncing off an object.

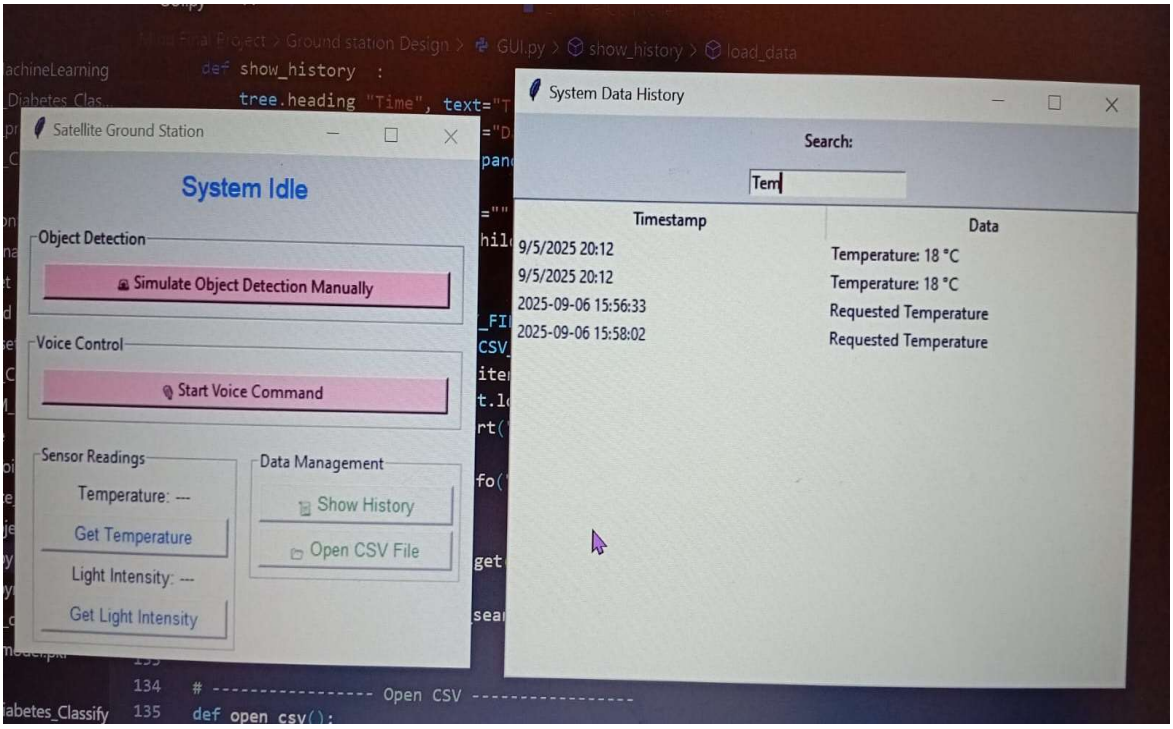
Function:

- Calculate the distance to an object (usually in cm or inches).
- Used in:
  - Obstacle detection in robotics
  - Car parking systems
  - Water level measurement
  - Security systems

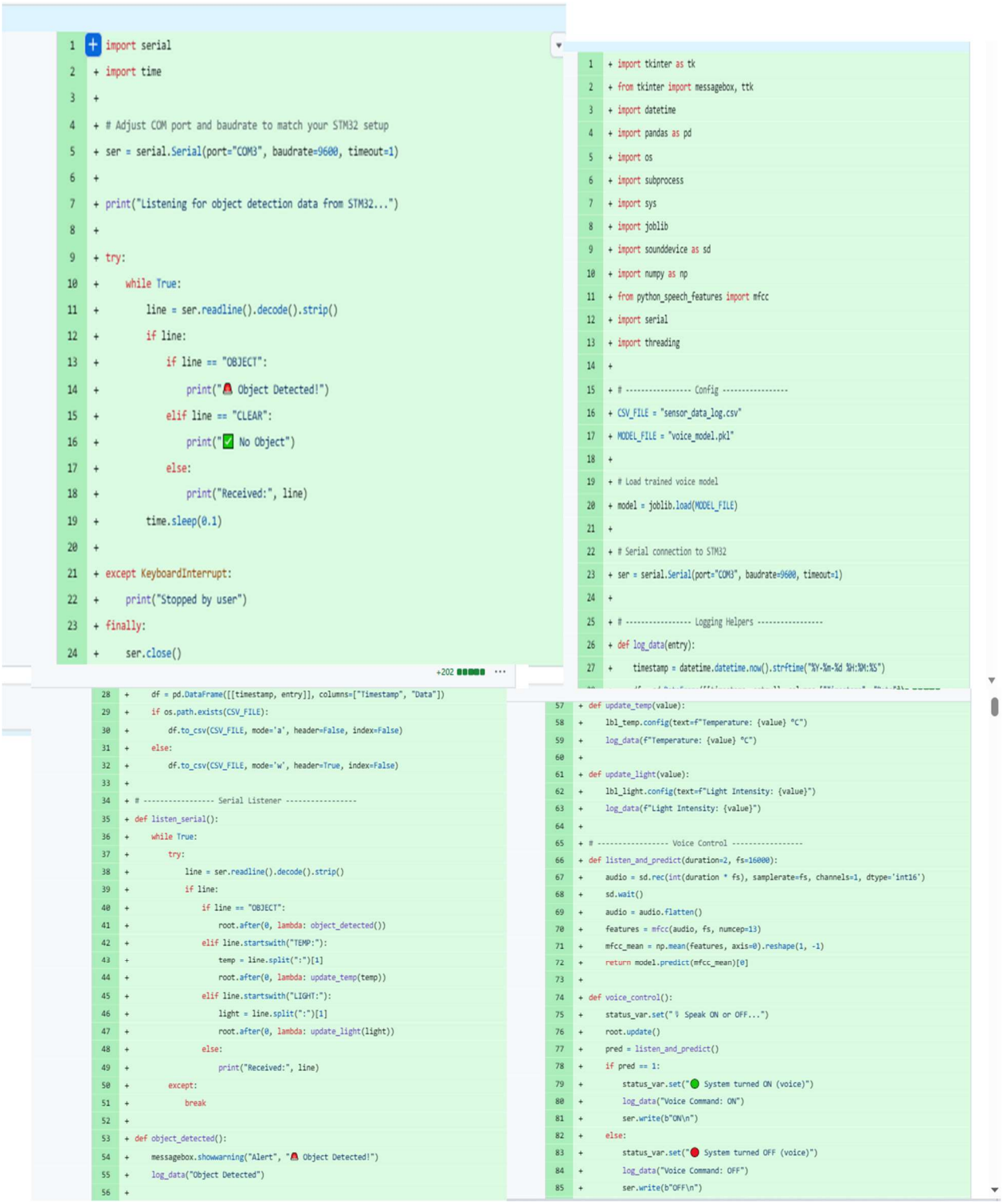
Popular example: HC-SR04 Ultrasonic Sensor

3. Software Design:

3.1 GUI Design: Layout and functionality.



3.2 Code GUI



```
+ import serial
+ import time
+
+ # Adjust COM port and baudrate to match your STM32 setup
+ ser = serial.Serial(port="COM3", baudrate=9600, timeout=1)
+
+ print("Listening for object detection data from STM32...")
+
+ try:
+     while True:
+         line = ser.readline().decode().strip()
+         if line:
+             if line == "OBJECT":
+                 print("🚨 Object Detected!")
+             elif line == "CLEAR":
+                 print("✅ No Object")
+             else:
+                 print("Received:", line)
+         time.sleep(0.1)
+
+ except KeyboardInterrupt:
+     print("Stopped by user")
+ finally:
+     ser.close()
```

```
+ import tkinter as tk
+ from tkinter import messagebox, ttk
+ import datetime
+ import pandas as pd
+ import os
+ import subprocess
+ import sys
+ import joblib
+ import sounddevice as sd
+ import numpy as np
+ from python_speech_features import mfcc
+ import serial
+ import threading
+
+ # ---------------- Config ----------------
+ CSV_FILE = "sensor_data_log.csv"
+ MODEL_FILE = "voice_model.pkl"
+
+ # Load trained voice model
+ model = joblib.load(MODEL_FILE)
+
+ # Serial connection to STM32
+ ser = serial.Serial(port="COM3", baudrate=9600, timeout=1)
+
+ # ---------------- Logging Helpers ----------------
+ def log_data(entry):
+     timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
+     df = pd.DataFrame([[timestamp, entry]], columns=["Timestamp", "Data"])
+     if os.path.exists(CSV_FILE):
+         df.to_csv(CSV_FILE, mode='a', header=False, index=False)
+     else:
+         df.to_csv(CSV_FILE, mode='w', header=True, index=False)
+
+ # ---------------- Serial Listener ----------------
+ def listen_serial():
+     while True:
+         try:
+             line = ser.readline().decode().strip()
+             if line:
+                 if line == "OBJECT":
+                     root.after(0, lambda: object_detected())
+                 elif line.startswith("TEMP:"):
+                     temp = line.split(":")[1]
+                     root.after(0, lambda: update_temp(temp))
+                 elif line.startswith("LIGHT:"):
+                     light = line.split(":")[1]
+                     root.after(0, lambda: update_light(light))
+                 else:
+                     print("Received:", line)
+         except:
+             break
+
+ def object_detected():
+     messagebox.showwarning("Alert", "🚨 Object Detected!")
+     log_data("Object Detected")
+
```

```
+ def update_temp(value):
+     lbl_temp.config(text=f"Temperature: {value} °C")
+     log_data(f"Temperature: {value} °C")
+
+ def update_light(value):
+     lbl_light.config(text=f"Light Intensity: {value}")
+     log_data(f"Light Intensity: {value}")
+
+ # ---------------- Voice Control ----------------
+ def listen_and_predict(duration=2, fs=16000):
+     audio = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')
+     sd.wait()
+     audio = audio.flatten()
+     features = mfcc(audio, fs, numcep=13)
+     mfcc_mean = np.mean(features, axis=0).reshape(1, -1)
+     return model.predict(mfcc_mean)[0]
+
+ def voice_control():
+     status_var.set("🎙 Speak ON or OFF...")
+     root.update()
+     pred = listen_and_predict()
+     if pred == 1:
+         status_var.set("🟢 System turned ON (voice)")
+         log_data("Voice Command: ON")
+         ser.write(b"ON\n")
+     else:
+         status_var.set("🔴 System turned OFF (voice)")
+         log_data("Voice Command: OFF")
+         ser.write(b"OFF\n")
```

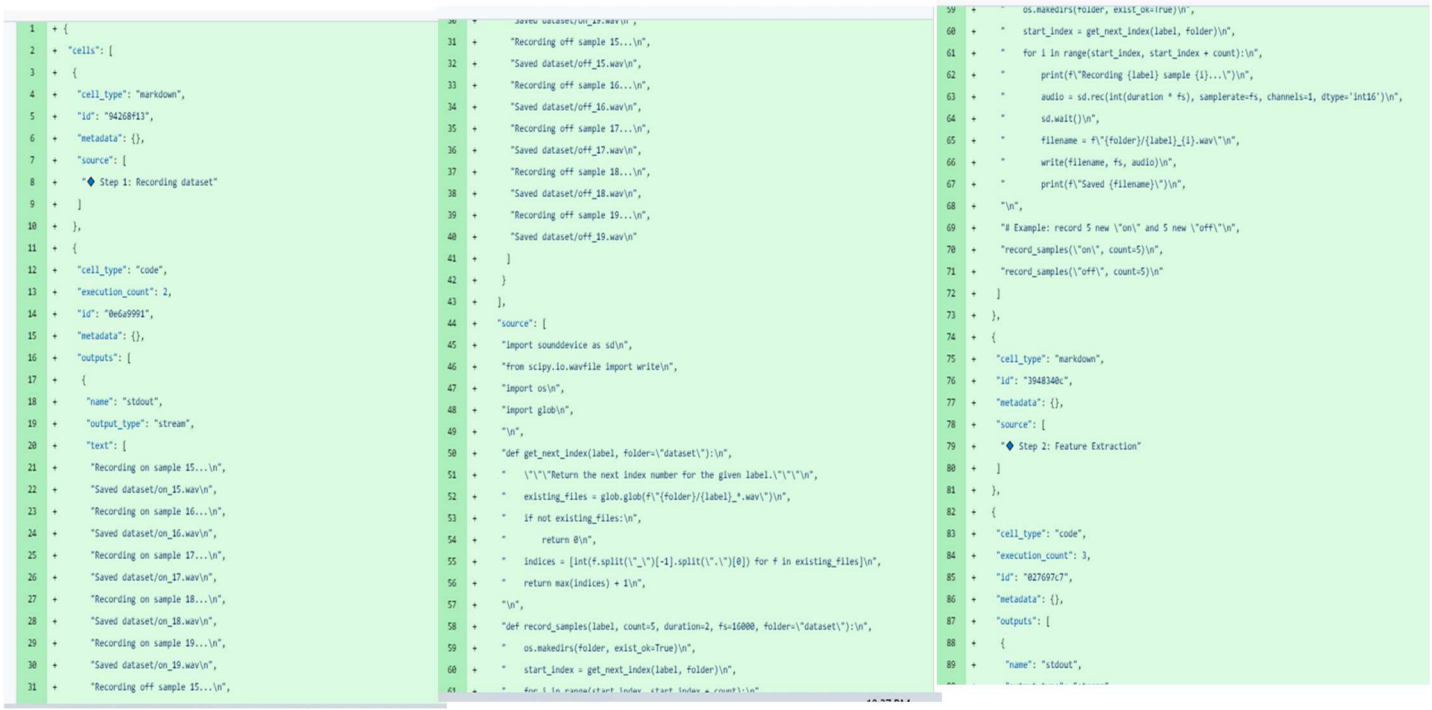
```
+
+ # ---------------- Manual Requests ----------------
+ def request_temp():
+     if ser:
+         ser.write(b"GET_TEMP\n")
+         log_data("Requested Temperature")
+
+ def request_light():
+     if ser:
+         ser.write(b"GET_LIGHT\n")
+         log_data("Requested Light")
+
+ # ---------------- History with Search ----------------
+ def show_history():
+     history_window = tk.Toplevel(root)
+     history_window.title("System Data History")
+     history_window.geometry("500x400")
+
+     # Search box
+     search_var = tk.StringVar()
+     tk.Label(history_window, text="Search:").pack(pady=5)
+     search_entry = tk.Entry(history_window, textvariable=search_var)
+     search_entry.pack(pady=5)
+
+     # Treeview
+     tree = ttk.Treeview(history_window, columns=("Time", "Data"), show="headings")
+     tree.heading("Time", text="Timestamp")
+     tree.heading("Data", text="Data")
+     tree.pack(fill="both", expand=True)
```

```
+
+     def load_data(filter_text=""):
+         for row in tree.get_children():
+             tree.delete(row)
+
+         if os.path.exists(CSV_FILE):
+             df = pd.read_csv(CSV_FILE)
+             for _, row in df.iterrows():
+                 if filter_text.lower() in str(row["Data"]).lower():
+                     tree.insert("", "end", values=(row["Timestamp"], row["Data"]))
+         else:
+             messagebox.showinfo("History", "No data found yet!")
+
+     def on_search(*args):
+         load_data(search_var.get())
+
+     search_var.trace("w", on_search)
+     load_data()
+
+ # ---------------- Open CSV ----------------
+ def open_csv():
+     if not os.path.exists(CSV_FILE):
+         messagebox.showerror("Error", "CSV file does not exist yet!")
+         return
+     try:
+         if sys.platform.startswith("win"):
+             os.startfile(CSV_FILE)
+         elif sys.platform.startswith("darwin"):
```

```
+         else:
+             subprocess.call(["xdg-open", CSV_FILE])
+     except Exception as e:
+         messagebox.showerror("Error", f"Could not open file: {e}")
+
+ # ---------------- GUI Layout ----------------
+ root = tk.Tk()
+ root.title("Satellite Ground Station")
+ root.geometry("370x360")
+
+ status_var = tk.StringVar(value="System Idle")
+ tk.Label(root, textvariable=status_var, font=("Arial", 14), fg="blue").pack(pady=10)
+
+ # Object Detection Box
+ frame_object = tk.LabelFrame(root, text="Object Detection", padx=10, pady=10)
+ frame_object.pack(fill="x", padx=10, pady=5)
+ btn_obj = tk.Button(frame_object, text="🚨 Simulate Object Detection Manually",
  command=object_detected, bg="pink")
+ btn_obj.pack(fill="x")
+
+ # Voice Control Box
+ frame_voice = tk.LabelFrame(root, text="Voice Control", padx=10, pady=10)
+ frame_voice.pack(fill="x", padx=10, pady=5)
+ btn_voice = tk.Button(frame_voice, text="🎙 Start Voice Command",
  command=voice_control, bg="pink")
+ btn_voice.pack(fill="x")
+
+ # Sensor Readings Box
+ frame_row = tk.Frame(root)
```

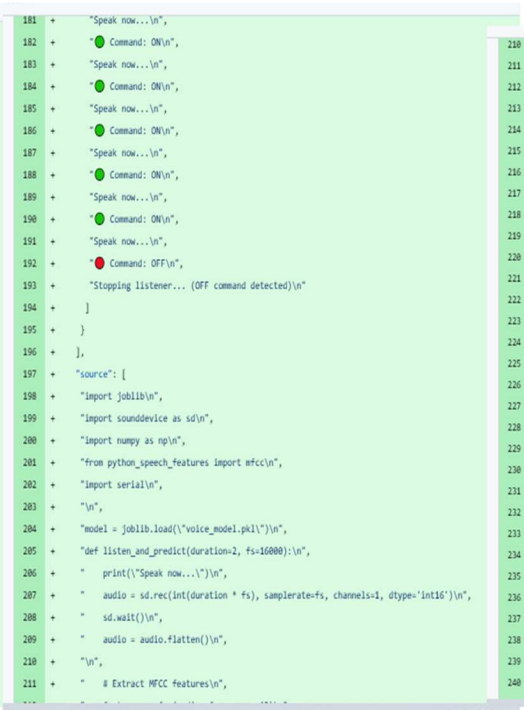
```
+ # Sensor Readings Box
+ frame_row = tk.Frame(root)
+ frame_row.pack(padx=10, pady=5, fill="x")
+ frame_sensors = tk.LabelFrame(frame_row, text="Sensor Readings", padx=5, pady=5)
+ frame_sensors.grid(row=0, column=0, padx=5, pady=5, sticky="n")
+ lbl_temp = tk.Label(frame_sensors, text="Temperature: ---", font=("Arial", 10))
+ lbl_temp.pack(pady=2)
+ btn_temp = tk.Button(frame_sensors, text="Get Temperature", command=request_temp,
+                       font=("Arial", 10), width=18, height=1, fg="blue",)
+ btn_temp.pack(pady=2)
+ lbl_light = tk.Label(frame_sensors, text="Light Intensity: ---", font=("Arial", 10))
+ lbl_light.pack(pady=2)
+ btn_light = tk.Button(frame_sensors, text="Get Light Intensity", command=request_light,
+                       font=("Arial", 10), width=18, height=1, fg="blue",)
+ btn_light.pack(pady=2)
+
+ # Data Management Box
+ frame_data = tk.LabelFrame(frame_row, text="Data Management", padx=5, pady=5)
+ frame_data.grid(row=0, column=1, padx=5, pady=5, sticky="n")
+
+ btn_history = tk.Button(frame_data, text="📜 Show History", command=show_history,
+                         font=("Arial", 10), width=18, height=1, fg="green")
+ btn_history.pack(pady=2)
+
+ btn_open_csv = tk.Button(frame_data, text="📂 Open CSV File", command=open_csv,
+                          font=("Arial", 10), width=18, height=1, fg="green")
+ btn_open_csv.pack(pady=2)
+
+ # ---------------- Run Serial Listener in Background ----------------
+ threading.Thread(target=listen_serial, daemon=True).start()
+
+ root.mainloop()
+ ser.close()
```

3.2 Voice Control: Machine learning model details, training data, and implementation code

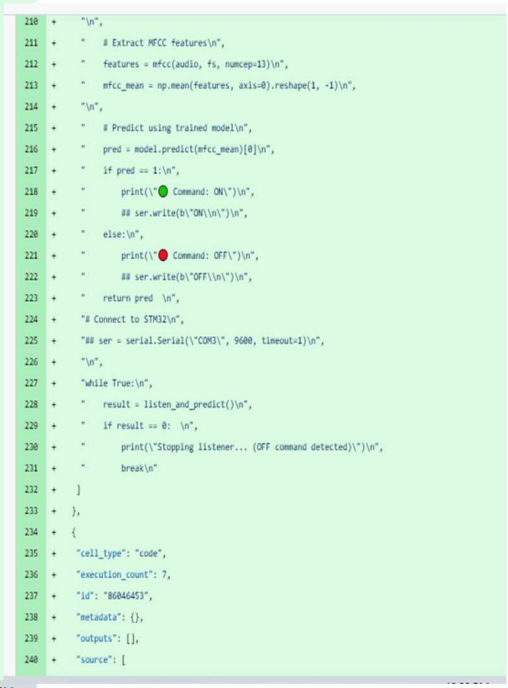


```
+ {
+  "cells": [
+   {
+    "cell_type": "markdown",
+    "id": "94268f13",
+    "metadata": {},
+    "source": [
+     "🔷 Step 1: Recording dataset"
+    ]
+   },
+   {
+    "cell_type": "code",
+    "execution_count": 2,
+    "id": "0e6a9991",
+    "metadata": {},
+    "outputs": [
+     {
+      "name": "stdout",
+      "output_type": "stream",
+      "text": [
+       "Recording on sample 15...\n",
+       "Saved dataset/on_15.wav\n",
+       "Recording on sample 16...\n",
+       "Saved dataset/on_16.wav\n",
+       "Recording on sample 17...\n",
+       "Saved dataset/on_17.wav\n",
+       "Recording on sample 18...\n",
+       "Saved dataset/on_18.wav\n",
+       "Recording on sample 19...\n",
+       "Saved dataset/on_19.wav\n",
+       "Recording off sample 15...\n",
+       "Saved dataset/off_15.wav\n",
+       "Recording off sample 16...\n",
+       "Saved dataset/off_16.wav\n",
+       "Recording off sample 17...\n",
+       "Saved dataset/off_17.wav\n",
+       "Recording off sample 18...\n",
+       "Saved dataset/off_18.wav\n",
+       "Recording off sample 19...\n",
+       "Saved dataset/off_19.wav\n"
+      ]
+     }
+    ],
+    "source": [
+     "import sounddevice as sd\n",
+     "from scipy.io.wavfile import write\n",
+     "import os\n",
+     "import glob\n",
+     "\n",
+     "def get_next_index(label, folder=\"dataset\"):\n",
+     "    \"\"\"Return the next index number for the given label.\"\"\"\n",
+     "    existing_files = glob.glob(f\"{folder}/{label}_*.wav\")\n",
+     "    if not existing_files:\n",
+     "        return 0\n",
+     "    indices = [int(f.split(\"_\")[-1].split(\".\")[0]) for f in existing_files]\n",
+     "    return max(indices) + 1\n",
+     "\n",
+     "def record_samples(label, count=5, duration=2, fs=16000, folder=\"dataset\"):\n",
+     "    os.makedirs(folder, exist_ok=True)\n",
+     "    start_index = get_next_index(label, folder)\n",
+     "    for i in range(start_index, start_index + count):\n",
+     "        print(f\"Recording {label} sample {i}...\")\n",
+     "        audio = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')\n",
+     "        sd.wait()\n",
+     "        filename = f\"{folder}/{label}_{i}.wav\"\n",
+     "        write(filename, fs, audio)\n",
+     "        print(f\"Saved {filename}\")\n",
+     "\n",
+     "# Example: record 5 new \"on\" and 5 new \"off\"\n",
+     "record_samples(\"on\", count=5)\n",
+     "record_samples(\"off\", count=5)\n"
+    ]
+   },
+   {
+    "cell_type": "markdown",
+    "id": "3948340c",
+    "metadata": {},
+    "source": [
+     "🔷 Step 2: Feature Extraction"
+    ]
+   },
+   {
+    "cell_type": "code",
+    "execution_count": 3,
+    "id": "027697c7",
+    "metadata": {},
+    "outputs": [
+     {
+      "name": "stdout",
+      "output_type": "stream",
+      "text": [
+       "Features shape: (40, 13)\n"
+      ]
+     }
+    ],
+    "source": [
+     "import glob\n",
+     "import numpy as np\n",
+     "from scipy.io import wavfile\n",
+     "from python_speech_features import mfcc\n",
+     "\n",
+     "X, y = [], []\n",
+     "\n",
+     "for file in glob.glob(\"dataset/*.wav\"):\n",
+     "    label = 1 if \"on\" in file else 0\n",
+     "    sr, audio = wavfile.read(file)\n",
+     "    features = mfcc(audio, sr, numcep=13)\n",
+     "    mfcc_mean = np.mean(features, axis=0)\n",
+     "    X.append(mfcc_mean)\n",
+     "    y.append(label)\n",
+     "\n",
+     "X = np.array(X)\n",
+     "y = np.array(y)\n",
+     "\n",
+     "print(\"Features shape:\", X.shape)\n"
+    ]
+   },
+   {
+    "cell_type": "markdown",
+    "id": "135a4b95",
+    "metadata": {},
+    "source": [
+     "🔷 Step 3: Train a Simple Model"
+    ]
+   },
+   {
+    "cell_type": "code",
+    "execution_count": 4,
+    "id": "2a8b0379",
+    "metadata": {},
+    "outputs": [
+     {
+      "name": "stdout",
+      "output_type": "stream",
+      "text": [
+       "Accuracy: 1.0\n",
+       "Model saved as voice_model.pkl\n"
+      ]
+     }
+    ],
+    "source": [
+     "from sklearn.linear_model import LogisticRegression\n",
+     "from sklearn.model_selection import train_test_split\n",
+     "from sklearn.metrics import accuracy_score\n",
+     "\n",
+     "X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)\n",
+     "\n",
+     "model = LogisticRegression(max_iter=1000)\n",
+     "model.fit(X_train, y_train)\n",
+     "\n",
+     "y_pred = model.predict(X_test)\n",
+     "print(\"Accuracy:\", accuracy_score(y_test, y_pred))\n",
+     "\n",
+     "import joblib\n",
+     "joblib.dump(model, \"voice_model.pkl\")\n",
+     "print(\"Model saved as voice_model.pkl\")\n"
+    ]
+   },
+   {
+    "cell_type": "markdown",
+    "id": "9ae02017",
+    "metadata": {},
+    "source": [
+     "🔷 Step 4: Real-time Voice Command Detection"
+    ]
+   },
+   {
+    "cell_type": "code",
+    "execution_count": 5,
+    "id": "09394ad4",
+    "metadata": {},
+    "outputs": [
+     {
+      "name": "stdout",
+      "output_type": "stream",
+      "text": [
+       "Speak now...\n",
+       "🟢 Command: ON\n",
+       "Speak now...\n",
+       "🟢 Command: ON\n",
```

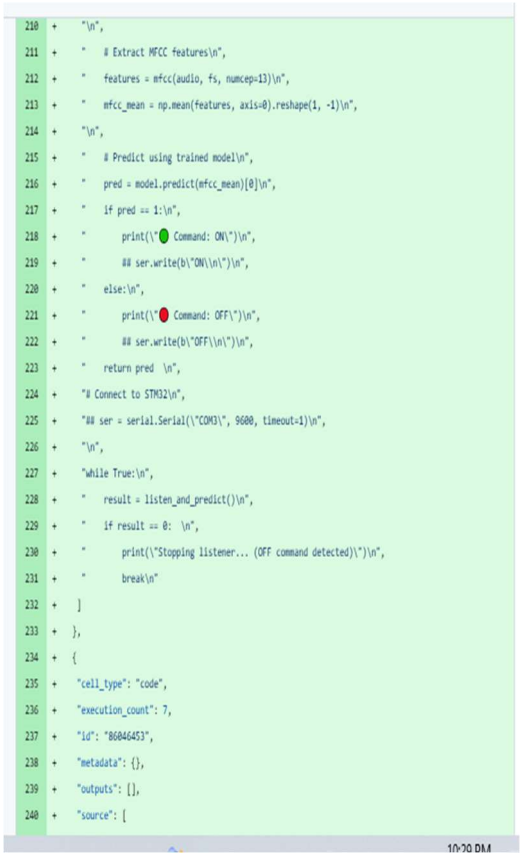
```
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🔴 Command: OFF\n",
      "Stopping listener... (OFF command detected)\n"
     ]
    }
   ],
   "source": [
    "import joblib\n",
    "import sounddevice as sd\n",
    "import numpy as np\n",
    "from python_speech_features import mfcc\n",
    "import serial\n",
    "\n",
    "model = joblib.load(\"voice_model.pkl\")\n",
    "def listen_and_predict(duration=2, fs=16000):\n",
    "    print(\"Speak now...\")\n",
    "    audio = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')\n",
    "    sd.wait()\n",
    "    audio = audio.flatten()\n",
    "\n",
    "    # Extract MFCC features\n",
```



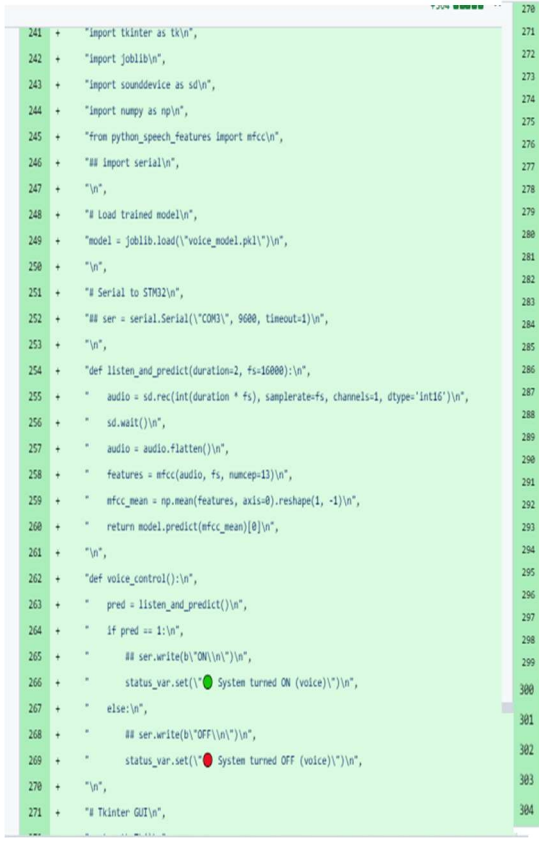
```
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🟢 Command: ON\n",
      "Speak now...\n",
      "🔴 Command: OFF\n",
      "Stopping listener... (OFF command detected)\n"
     ]
    }
   ],
   "source": [
    "import joblib\n",
    "import sounddevice as sd\n",
    "import numpy as np\n",
    "from python_speech_features import mfcc\n",
    "import serial\n",
    "\n",
    "model = joblib.load(\"voice_model.pkl\")\n",
    "def listen_and_predict(duration=2, fs=16000):\n",
    "    print(\"Speak now...\")\n",
    "    audio = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')\n",
    "    sd.wait()\n",
    "    audio = audio.flatten()\n",
    "\n",
    "    # Extract MFCC features\n",
```



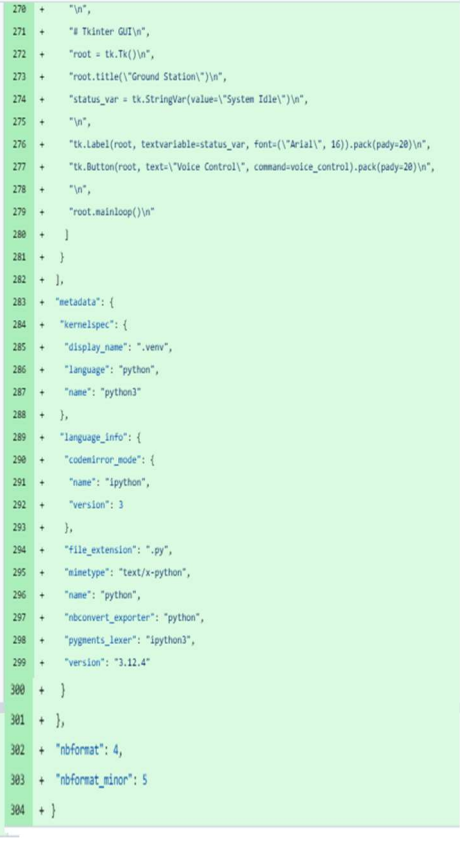
```
    "\n",
    "    # Extract MFCC features\n",
    "    features = mfcc(audio, fs, numcep=13)\n",
    "    mfcc_mean = np.mean(features, axis=0).reshape(1, -1)\n",
    "\n",
    "    # Predict using trained model\n",
    "    pred = model.predict(mfcc_mean)[0]\n",
    "    if pred == 1:\n",
    "        print(\"🟢 Command: ON\")\n",
    "        ## ser.write(b\"ON\\n\")\n",
    "    else:\n",
    "        print(\"🔴 Command: OFF\")\n",
    "        ## ser.write(b\"OFF\\n\")\n",
    "    return pred  \n",
    "# Connect to STM32\n",
    "## ser = serial.Serial(\"COM3\", 9600, timeout=1)\n",
    "\n",
    "while True:\n",
    "    result = listen_and_predict()\n",
    "    if result == 0:  \n",
    "        print(\"Stopping listener... (OFF command detected)\")\n",
    "        break\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 7,
   "id": "86046453",
   "metadata": {},
   "outputs": [],
   "source": [
```



```
    "\n",
    "    # Extract MFCC features\n",
    "    features = mfcc(audio, fs, numcep=13)\n",
    "    mfcc_mean = np.mean(features, axis=0).reshape(1, -1)\n",
    "\n",
    "    # Predict using trained model\n",
    "    pred = model.predict(mfcc_mean)[0]\n",
    "    if pred == 1:\n",
    "        print(\"🟢 Command: ON\")\n",
    "        ## ser.write(b\"ON\\n\")\n",
    "    else:\n",
    "        print(\"🔴 Command: OFF\")\n",
    "        ## ser.write(b\"OFF\\n\")\n",
    "    return pred  \n",
    "# Connect to STM32\n",
    "## ser = serial.Serial(\"COM3\", 9600, timeout=1)\n",
    "\n",
    "while True:\n",
    "    result = listen_and_predict()\n",
    "    if result == 0:  \n",
    "        print(\"Stopping listener... (OFF command detected)\")\n",
    "        break\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 7,
   "id": "86046453",
   "metadata": {},
   "outputs": [],
   "source": [
```



```
    "import tkinter as tk\n",
    "import joblib\n",
    "import sounddevice as sd\n",
    "import numpy as np\n",
    "from python_speech_features import mfcc\n",
    "## import serial\n",
    "\n",
    "# Load trained model\n",
    "model = joblib.load(\"voice_model.pkl\")\n",
    "\n",
    "# Serial to STM32\n",
    "## ser = serial.Serial(\"COM3\", 9600, timeout=1)\n",
    "\n",
    "def listen_and_predict(duration=2, fs=16000):\n",
    "    audio = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')\n",
    "    sd.wait()\n",
    "    audio = audio.flatten()\n",
    "    features = mfcc(audio, fs, numcep=13)\n",
    "    mfcc_mean = np.mean(features, axis=0).reshape(1, -1)\n",
    "    return model.predict(mfcc_mean)[0]\n",
    "\n",
    "def voice_control():\n",
    "    pred = listen_and_predict()\n",
    "    if pred == 1:\n",
    "        ## ser.write(b\"ON\\n\")\n",
    "        status_var.set(\"🟢 System turned ON (voice)\")\n",
    "    else:\n",
    "        ## ser.write(b\"OFF\\n\")\n",
    "        status_var.set(\"🔴 System turned OFF (voice)\")\n",
    "\n",
    "# Tkinter GUI\n",
```



```
    "\n",
    "# Tkinter GUI\n",
    "root = tk.Tk()\n",
    "root.title(\"Ground Station\")\n",
    "status_var = tk.StringVar(value=\"System Idle\")\n",
    "\n",
    "tk.Label(root, textvariable=status_var, font=(\"Arial\", 16)).pack(pady=20)\n",
    "tk.Button(root, text=\"Voice Control\", command=voice_control).pack(pady=20)\n",
    "\n",
    "root.mainloop()\n"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": ".venv",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.12.4"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

3.4 Data Storage:

Link for it present in the repo in the dataset

https://github.com/RewanKhaled/Mega_Project

4. Implementation and Testing:

4.1 Challenges faced and solutions implemented.

Everything was working perfectly on tinker cad, GUI, Altium designer until we tried to upload the code on STM it using Arduino IDE software it was uploading, but the code was not running so we used another laptop. The red led of STM was on, but the green was not. We thought maybe the green led of STM itself was burned, so we tried a different code from the examples and we found that was working. Now we know that problem is not from the STM, we changed the breadboard and even used Avometer to make sure wires and components are working and not damaged and they were okay. Last but not least, we tried downloading STM cube on three laptops and finally it downloaded on the third Laptop. After downloading it code was running perfectly and STM was shinning with its green light.

4.2 Testing procedures for each module (temperature, light, radar).

Working perfectly

4.3 GUI testing

Working perfectly

5. Conclusion:

5.1 Future improvements and recommendations.

his project successfully developed a low-cost, STM based satellite system, providing a practical demonstration of radio detection, data storage, temperature sensing, voice controlling and ranging principles for object detection and distance measurement. The system's success in providing a functional and educational tool is evidenced by its ability to map detected objects, though limitations such as the inherent inaccuracies of ultrasonic sensors and restricted detection range were observed. Future work could involve the integration of more advanced sensors, such as those used in professional RF radar systems, and the development of more robust software for enhanced object tracking and data processing, ultimately expanding the project's potential for applications in navigation and surveillance.

6. Appendices:

6.1Code

```
#include <Servo.h>

// ================= Pins ==============
#define LDR_PIN    PA5
#define TEMP_PIN   PA6

#define TRIG_PIN   PA7
#define ECHO_PIN   PB0

#define SERVO_PIN   PB1

#define RED1_PIN    PB7
#define RED2_PIN    PB6
#define RED3_PIN    PB5

#define GREEN_PIN   PB4
#define WHITE_PIN   PB3

// ================= Objects ============
Servo radarServo;

// ================= Variables ===========
int servoAngle = 0;
int servoStep = 2;

// ================= Setup ==============
void setup() {
```

```
// ================= Setup ==============
void setup() {
  Serial.begin(9600);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  pinMode(RED1_PIN, OUTPUT);
  pinMode(RED2_PIN, OUTPUT);
  pinMode(RED3_PIN, OUTPUT);

  pinMode(GREEN_PIN, OUTPUT);
  pinMode(WHITE_PIN, OUTPUT);

  radarServo.attach(SERVO_PIN);
  radarServo.write(0);
}

// ================= Main Loop ==========
void loop() {
  // ----- Temperature -----
  float temperature = readTemp();

  if (temperature < 20) {
    digitalWrite(RED1_PIN, HIGH);
    digitalWrite(RED2_PIN, LOW);
    digitalWrite(RED3_PIN, LOW);
```

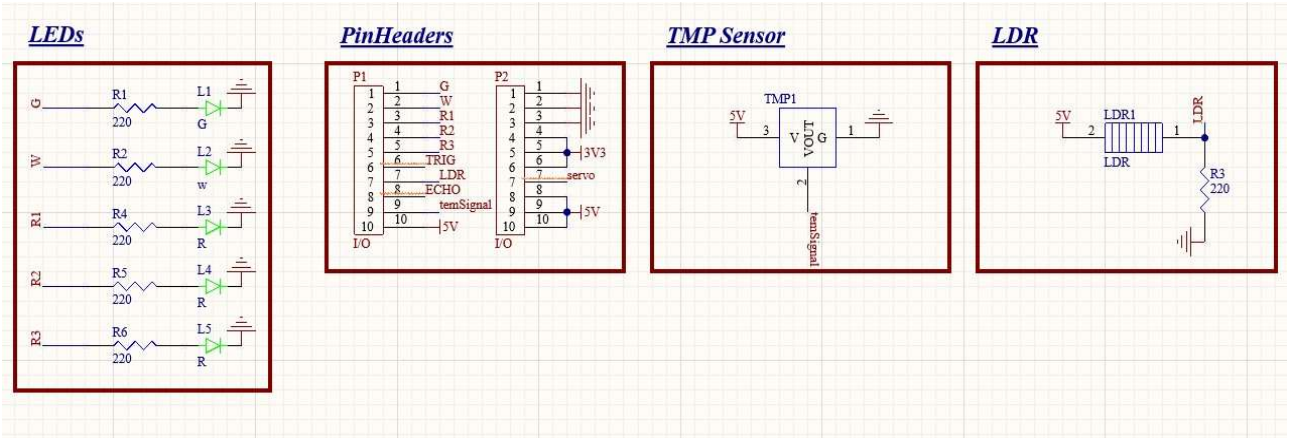
```
}
else if (temperature >= 20 && temperature <= 30) {
  digitalWrite(RED1_PIN, HIGH);
  digitalWrite(RED2_PIN, HIGH);
  digitalWrite(RED3_PIN, LOW);
}
else {
  digitalWrite(RED1_PIN, HIGH);
  digitalWrite(RED2_PIN, HIGH);
  digitalWrite(RED3_PIN, HIGH);
}

// ----- LDR -----
int ldrVal = analogRead(LDR_PIN);
if (ldrVal < 500) { // threshold → tune as needed
  digitalWrite(WHITE_PIN, HIGH);
} else {
  digitalWrite(WHITE_PIN, LOW);
}

// ----- Ultrasonic Radar -----
long distance = readUltrasonic();
if (distance > 0 && distance < 50) {  // object detected closer than 50cm
  digitalWrite(GREEN_PIN, HIGH);
} else {
  digitalWrite(GREEN_PIN, LOW);
}
```

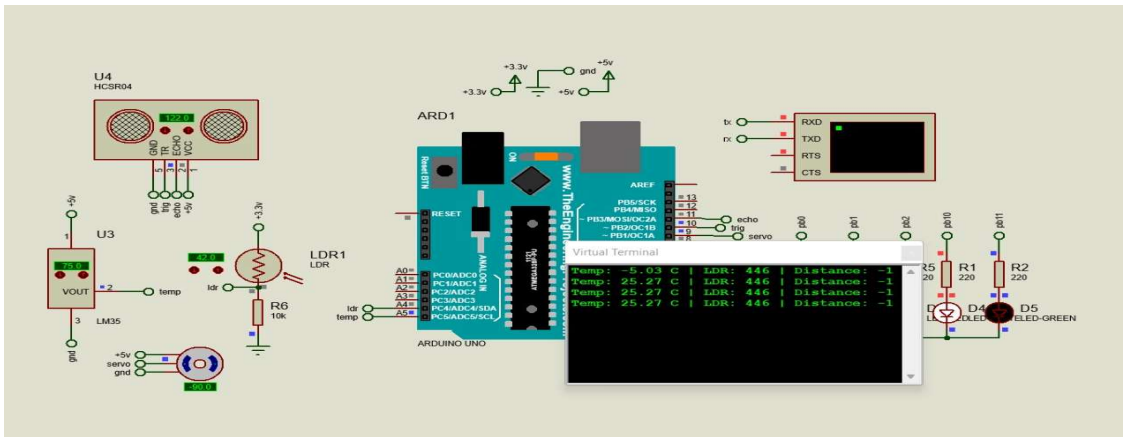
```
  // Servo sweep
  radarServo.write(servoAngle);
  servoAngle += servoStep;
  if (servoAngle >= 180 || servoAngle <= 0) {
    servoStep = -servoStep;
  }

  // ----- Serial Output -----
  Serial.print("Temp: ");
  Serial.print(temperature);
  Serial.print(" C | LDR: ");
  Serial.print(ldrVal);
  Serial.print(" | Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(100);
}

// ================= Helper Functions ================
float readTemp() {
  int val = analogRead(TEMP_PIN);
  float voltage = val * 5.0 / 1023.0;
  float tC = (voltage - 0.5) * 100.0; // TMP36
  return tC;
}
```

```
// ================= Helper Functions ==============
float readTemp() {
  int val = analogRead(TEMP_PIN);
  float voltage = val * 5.0 / 1023.0;
  float tC = (voltage - 0.5) * 100.0; // TMP36
  return tC;
}

long readUltrasonic() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(5);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH); // no timeout first
  if (duration == 0) return -1;          // no signal
  long distance = duration * 0.034 / 2;
  return distance;
}
```

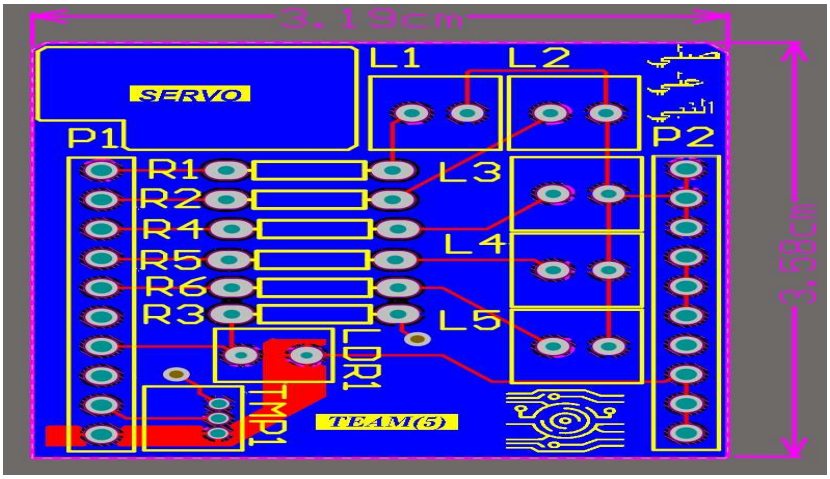
11

6.2Schematics

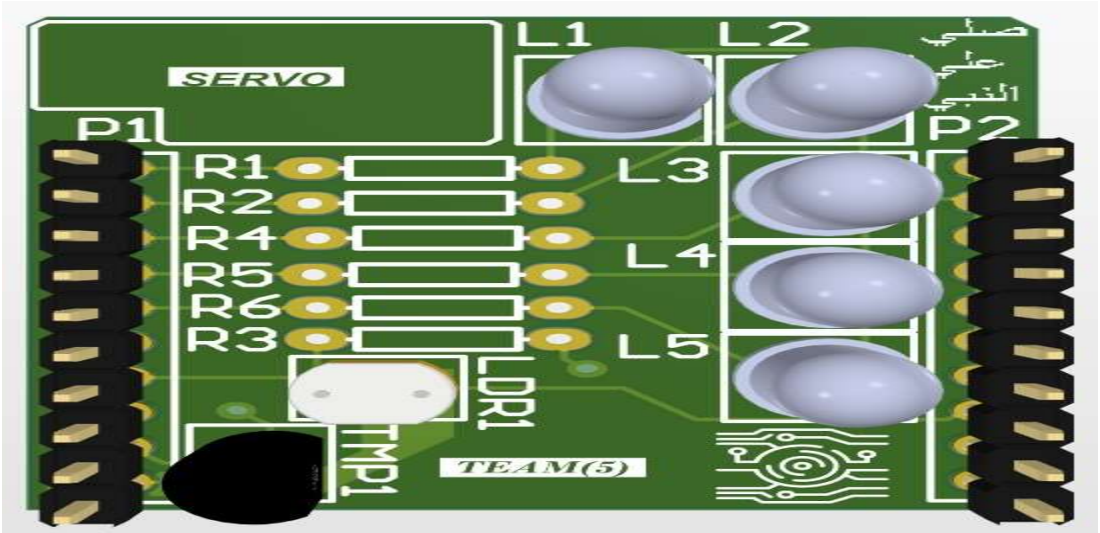




6.3 PCB in 2D



6.4 PCB in 3D



6.5 Link repohttps://github.com/RewanKhaled/Mega_Project