**< Radio-controlled data collecting plane/RCDCP>**

**Requirements analysis and idea formulation**

**Digital Media and Design course**

**Version 2.2.1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 29.10.2024 | 0.0.1 | Theoretical calculation, parts buying, getting requested environment, searching for the software | Aleksander Sem.  Aleksander Ser.  Alex  Misha |

# Problem research and analysis

The idea is to create an RC plane that can record and send telemetric data, including air pressure, wind speed, temperature, and others to a computer.

Research shows plenty of ways to create RC controls and data transmission in real time, depending on your financial situation. We’ve stuck to the ESP-32 WIFI transmission system. It’s the best in our situation due to the cheapness and flexibility of this technology. The cheapness is reached thanks to not so big cover area and few channel system. The flexibility is achieved due to compatibility with Arduino architecture. Due to this compatibility, all the other electronics are Arduino-compatible. We are using Micro Servos to control movable surfaces. We are using two three-bladed propellers on each side. Ideally, the plane should weigh not more than 2 kg. For that purpose, we are using highly efficient brushless motors. For mobility and repairability sake, our plane will consist of several parts, to be exact, two separatable wings, a body and a nose cone. The computer is connected using the transmitter with a connected antenna; the coverage distance is approximately 300 meters.

The software is designed to show all collected data on a screen, as well as give a user the ability to control the aircraft. Some of the assembly process will happen at our homes, but most of it will happen in the RTU scientific laboratory. The lab can provide us with most of the equipment, such as a 3D printer soldering-iron devices, Arduino development boards and for a short period of time in the beginning, power supplies for tests.

Our project may face with different types of problems starting with software issues and ending with a physical plane disability to fly. The list of hypothetical issues we may face with: Plane body overweight or lack of hardness – plane must be light and at the same time lasting enough to endure possible overload either positive or negative therefore we will need to find the best material for the plane body probably it with be some kind of plastic; Issues with the correct electronics assembling and also a risk of losing some part while testing systems, motors will consume a lot of energy and in case of incorrect assembling or calculations we may lose motors or other components; Problems with the transmitter or receiver, we will do telecommunication system for the first time and we probably will get some issues with it; Writing code for the Arduino could become a problem because we don’t have experience on C++ language; Also we don’t have experience of the creating own application and UI, we will need to learn photoshop. Of course will may face other problems at every stage of the project, but I hope that problems will only disappear or get resolved.

# Software specification

**1.1 Epic definition**

For explorers and normal/nerdy people.

Who want to get information about environmental qualities, about air and specific places, or just for fun!

The ability of getting into places and domains with high risks for humans or just to entertain.

It is a material flying object for research and an opportunity to get to know the world of aircraft with our own created software.

That allows our customers to easily achieve what they want.

Unlike others, we create our product, especially for each customer, which means that all parts of the plane are created with each of our customer’s needs in mind, which makes the plane special.

Our company can make planes suitable for any customer’s need.

**1.2. Business process model**

<Description of the main process – think about what processes take place in the project and who is involved in them.>

First of all, to start with our plane you need to visit our site Meteo-Plane.net. Order the plane and download the software to be able to control and monitor the plane telemetry and data collecting. To use the plane safety we recommend reading our instructions.

First, you need to charge up the plane. You will need to find a place for Takeoff and Landing as well, it must be from 30 meters of asphalt(smooth☺).

After you have found the place for takeoff and landings, you must check all aircraft control surfaces, basically rotating surfaces up and down.

**1.3. Process diagram**

A diagram of a plane

Description automatically generated

**1.4. Defining requirements**

Functional requirements:

1. ‘ **Remote Control Functionality**: The RC plane must respond to user inputs from a handheld remote controller within a range of 500 meters.
2. ‘**Takeoff and Landing**: The plane must be capable of smooth takeoff and landing, either manually controlled or have an assisted mode for beginners.
3. ‘**Stable Flight**: The plane must maintain stable flight with minimal corrections needed in calm weather conditions.
4. ‘**Speed Control**: Users must be able to adjust the speed of the plane during flight using the remote controller.
5. ‘**Altitude Adjustment**: The plane must allow users to control altitude changes, including steady climbs, descents, and hovering (if applicable for the model).
6. ‘**Wind Resistance**: The plane must be capable of operating in light to moderate wind conditions up to 15 km/h without losing stability.
7. ‘**Charge and Use Duration**: The plane must provide at least 20 minutes of flight time after a full battery charge, and the charging process must take no more than 2 hours.
8. ‘**Environmental Monitoring**: The plane must be equipped with sensors to monitor weather conditions, such as temperature, air pressure, and wind speed, and display this data on the controller or paired device.
9. ‘**GPS Tracking**: The plane must have GPS functionality to track its current location, altitude, and flight path, accessible via a paired app.
10. ‘**Weather Data Integration**: The plane must integrate weather data into flight controls, providing real-time recommendations or warnings based on environmental conditions (e.g., strong winds or storms).
11. **Repair possibility**: The app can provide an ability to send back the plane and request repair.

Non-functional requirements:

1. **Performance**: The plane must maintain consistent responsiveness to remote control commands with a latency of less than 100 milliseconds.
2. **Scalability**: The system must support integration with additional sensors, such as air quality or radiation detectors, without significant reconfiguration.
3. **Reliability**: The plane’s sensors and communication systems must function accurately for at least 95% of the operational flight duration under standard conditions.
4. **Durability**: The plane must withstand environmental factors like light rain, dust.
5. **Battery Efficiency**: The system's sensors and on board electronics must consume less than 20% of the total battery capacity during a standard 20-minute flight.
6. **Ease of Use**: The interface for monitoring sensor data (e.g., temperature, air pressure, GPS) must be intuitive and accessible to both beginners and advanced users.
7. **Data Accuracy**: The sensors must provide readings with a maximum deviation of ±2% for temperature, ±5% for air pressure, and ±1 meter for GPS coordinates.
8. **Data Transmission Security**: All communication between the plane and the remote controller must use encrypted protocols to prevent unauthorized access.
9. **System Scalability**: The software must support firmware updates to accommodate future functionality enhancements or sensor upgrades.
10. **Weather Resistance**: The plane must operate effectively in a temperature range of 0°C to 30°C and in winds up to 15 km/h.
11. **Signal Range**: The system must maintain a stable connection with the remote controller up to 500 meters in unobstructed conditions.
12. **Data Logging**: The plane must log flight data (e.g., altitude, speed, temperature) and store it for post-flight analysis.
13. **Startup Time**: The plane's onboard systems must initialize and be ready for operation within 10 seconds of being powered on.

**1.5. Selection and justification for initial programming tools, frameworks, languages**

Plane Coding:

* 1. Arduinp IDE
  2. C++

Libraries

* + - 1. Esp32 by Espressif Systems
      2. Accelerometer ADXL335 by Seeed-Studio
      3. Adafruit BME280 Library by Adafruit
      4. Adafruit BusIO by Adafruit
      5. Adafruit Unified Sensor by Adafruit
      6. BME280 by Tyler Glenn
      7. BME280\_Arduino\_I2C by Andrei Paramoshkin
      8. EasyEspNow by Henri Berisha
      9. EspSoftSerial by Dirk Kaar and Peter Lerup
      10. TinyGPSPlus by Mikal Hart
      11. TinyGPSPlus-ESP32 by Mikal Hart

Software Coding:

1. Python
2. TKinter software assembler

Libraries

* + 1. Serial
    2. Threading
    3. Sys
    4. Folium
    5. Os
    6. Webview

1. **Development Plan**

September:

Project Idea creation

October:

Theoretical research

Buy plane parts

Get access to the laboratory

Official project development start

GitHub repository creation

November:

Sensors test

Software development start

Coding sensors via Arduino IDE

December:

Connection between PC and esp32

Plane 3D model prototype

Motor, receiver and controlling surfaces test

January:

Camera and map implementation in software

February:

Print plane first model prototype

March:

Plane assembling

April:

First Takeoff and Landing

May:

Project presentation

Sum up results

# References

1. r/esp32 Sufficient-Market940 3 months ago <https://www.reddit.com/r/esp32/comments/1es32ut/doing_a_rc_car_from_scratch/>
2. Brian Lough’s YouTube video 6 years ago

<https://www.youtube.com/watch?v=0zs-A_fC3Yg&t=14s>

1. 3JWings’s YouTube video 6 months ago

<https://www.youtube.com/watch?v=dOqChqk8AAA>