RAJALAKSHMI INSTITUTE OF TECHNOLOGY

(An AUTONOMOUS Institution and Affiliated to ANNA UNIVERSITY, Chennai)
KUTHAMBAKKAM, CHENNAI – 600 124

DEPARTMENT OF
B.E ELECTRONICS ENGINEERING
(VLSI DESIGN AND TECHNOLOGY)

SKYFI

A MINI PROJECT REPORT

Submitted by

SRIGAJALAKSHMI P - (2117230050046)

NAVEEN A - (2117230050026)

NAVEENRAJ J - (2117230050027)

Wi-Fi controlled Paper Plane using ESP 32 S3

ABSTRACT

The SKY-FI project focuses on developing a cost-effective, Wi-Fi enabled flying paper plane powered by the ESP32-S3 module. Unlike conventional paper planes, SKY-FI integrates IoT, embedded systems, and mobile application development to create an interactive and educational platform. A custom-designed PCB and 3D-printed aerodynamic frame ensure hardware efficiency and stability, while a Flutter-based mobile app enables real-time control through a tilt mechanism using UDP protocol. The system offers advantages such as affordability, longer flight duration, open-source adaptability, and strong educational value compared to commercial alternatives. Designed for RC hobbyists, DIY electronics enthusiasts, and learners in robotics and IoT, SKY-FI combines innovation and practicality by serving as both an engaging project and a proof-of-concept for wireless control systems. This project demonstrates how affordable technology can bridge creativity, learning, and real-world application in the field of IoT-enabled robotics.

COMPONENTS

Hardware Components:

- 3D frame
- Esp32 s3
- CW and CCW Propeller
- Coreless DC motor
- 30C 350mAH LiPo Battery
- Tp4056 module
- Motor driver
- Wires and connectors
- Paper

Software Tools:

- Arduino IDE v1.8.X (Legacy IDE)
- Flutter

WORKING METHODOLOGY

The SKY-FI prototype operates through the seamless integration of hardware design, wireless communication, and mobile-based control, enabling a cost-effective and innovative flying platform.

1. System Design and Integration:

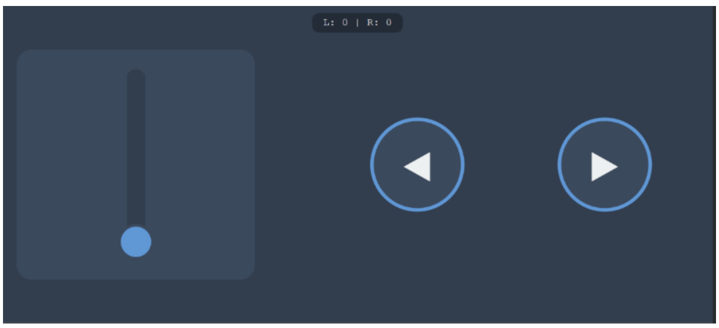
- A custom PCB is designed to integrate the ESP32-S3 microcontroller with the power system and motor driver.
- The 3D-printed plane frame, modeled using Autodesk, ensures aerodynamic stability and provides structural housing for the electronics.
- A lightweight coreless DC motor with propeller is mounted to generate thrust, powered by a compact Li-Po battery managed through a TP4056 charging module for safe recharging.

2. Embedded Control and Programming

- The ESP32-S3 is programmed using the Arduino IDE, where Wi-Fi libraries enable connectivity with the mobile application.
- A UDP protocol is implemented to ensure low-latency, real-time communication between the plane and the controller device.
- Control logic within the microcontroller interprets incoming signals and translates them into motor actuation for directional control.

3. Mobile Application and User Interaction

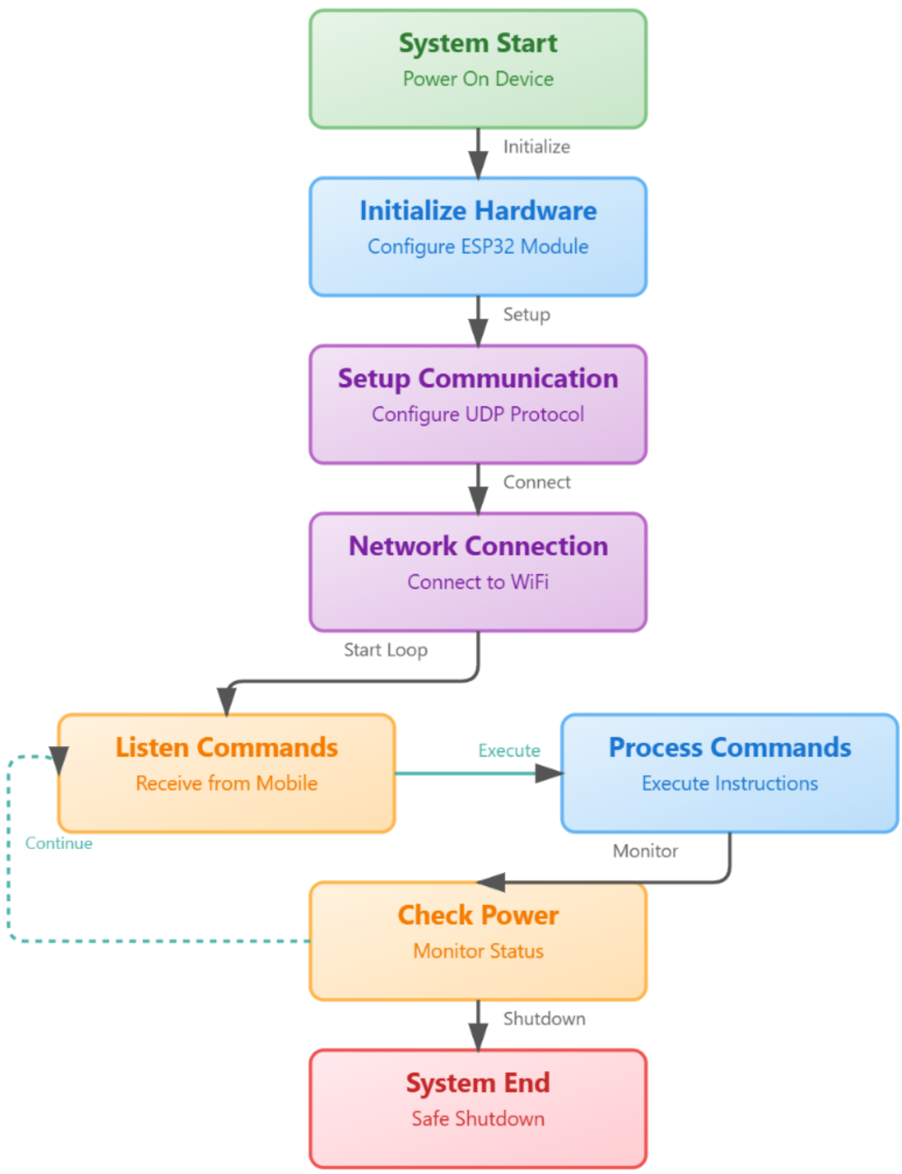
- A custom Flutter-based mobile application is developed to provide an intuitive user interface.



4. Operational Workflow

- The user powers on the system, establishing a Wi-Fi link between the smartphone and ESP32-S3.
- As the phone is tilted, real-time control signals are transmitted to the microcontroller.
- The ESP32-S3 adjusts the motor's thrust accordingly, producing the required movement of the plane.
- Throughout flight, the system maintains responsiveness and stability, showcasing reliable performance within its operating range.

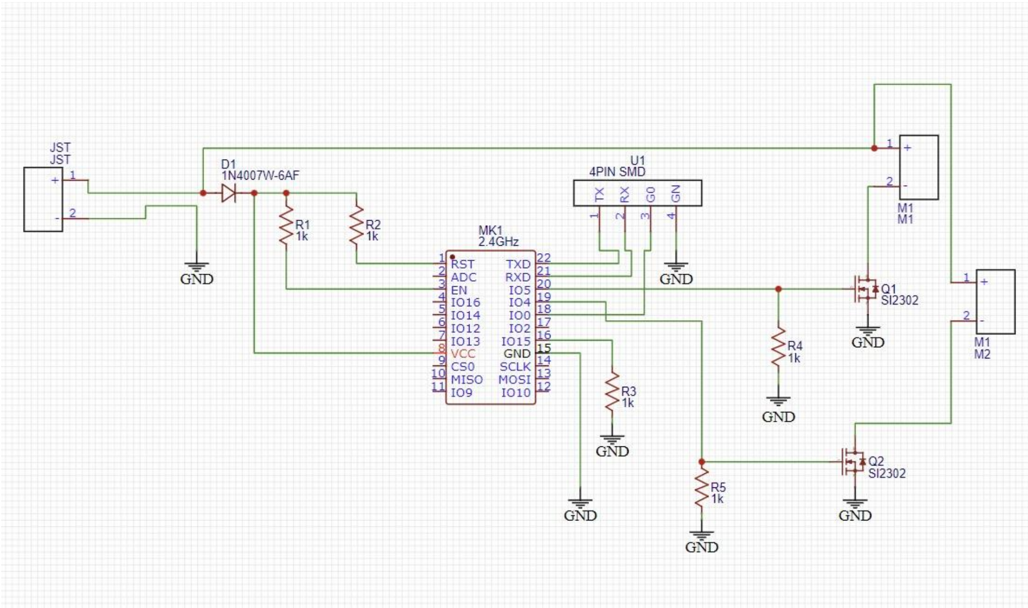
FLOWCHART



STEPS TO IMPLEMENT

1. Hardware Design

- Design a custom PCB from scratch to integrate the ESP32-S3, motor driver, battery, and supporting circuitry.
- Assemble the PCB with necessary components to ensure proper power and signal flow.



2. Mechanical Structure

- Design and fabricate a 3D-printed plane frame using Autodesk to achieve lightweight, aerodynamic stability.
- Mount the motor, propeller, and electronic components securely on the frame.

3. Power Management

- Connect a Li-Po battery as the primary power source.
- Add a TP4056 charging module to enable safe charging and protection of the battery.

4. Embedded Programming

- Program the ESP32-S3 using the Arduino IDE.
- Implement Wi-Fi libraries and configure the UDP protocol for low-latency communication.
- Develop control logic to translate incoming commands into motor actuation.

CODE:

```
#include <WiFi.h>
#include <WebServer.h>

// Wi-Fi credentials
const char* ssid = "poco";
const char* password = "12345678";

// Motor control pins (PWM-capable)
const int gpLm = 5; // Left Motor GPIO
const int gpRm = 43; // Right Motor GPIO

// PWM settings
const int pwmFreq = 5000;          // Hz
const int pwmResolution = 8;        // 0–255

WebServer server(80);

// HTML UI - Manual Steering with Vertical Throttle
const char htmlPage[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html lang="en">
<head>
  <title>FPV RC Plane Control</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
  <style>
    :root {
      --bg-color: #2c3e50;
      --widget-bg: #34495e;
      --accent-color: #3498db;
      --text-color: #ecf0f1;
      --button-color: #3498db;
      --button-active-color: #2980b9;
```

```css
}
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background-color: var(--bg-color);
    color: var(--text-color);
    margin: 0;
    overflow: hidden; /* Prevent scrolling */
    -webkit-user-select: none; /* Safari */
    -ms-user-select: none; /* IE 10+ */
    user-select: none; /* Standard */
}
.main-container {
    display: flex;
    justify-content: space-between;
    align-items: center;
    height: 100vh;
    padding: 20px;
    box-sizing: border-box;
}
/* --- Throttle (Left Side) --- */
.throttle-container {
    width: 35%;
    height: 80%;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    background-color: var(--widget-bg);
    border-radius: 20px;
}
#throttleSlider {
    -webkit-appearance: none;
    appearance: none;
    width: 80%; /* Adjust width of the slider track */
```

```
    height: 25px;
    background: #2c3e50;
    outline: none;
    border-radius: 15px;
    cursor: pointer;
    transform: rotate(-90deg); /* Make it vertical */
}
#throttleSlider::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 50px;
    height: 50px;
    background: var(--accent-color);
    border-radius: 50%;
    border: 4px solid var(--widget-bg);
}
/* --- Steering (Right Side) --- */
.steering-container {
    width: 55%;
    height: 80%;
    display: flex;
    justify-content: space-around;
    align-items: center;
    flex-wrap: wrap;
}
.steer-button {
    width: 120px;
    height: 120px;
    border-radius: 50%;
    border: 5px solid var(--button-color);
    background-color: var(--widget-bg);
    color: var(--text-color);
    font-size: 3rem;
    font-weight: bold;
```

```
            display: flex;
            justify-content: center;
            align-items: center;
            cursor: pointer;
            touch-action: manipulation; /* Prevents zoom on double tap */
        }
        .steer-button:active {
            background-color: var(--button-active-color);
            transform: scale(0.95);
        }
        /* --- Debug Info (Top Center) --- */
        .debug-info {
            position: absolute;
            top: 10px;
            left: 50%;
            transform: translateX(-50%);
            font-family: 'Courier New', Courier, monospace;
            background-color: rgba(0,0,0,0.3);
            padding: 5px 15px;
            border-radius: 10px;
            font-size: 0.9rem;
        }
    </style>
</head>
<body>

    <div class="debug-info" id="debugInfo">
        L: 0 | R: 0
    </div>

    <div class="main-container">
        <div class="throttle-container">
            <input type="range" min="0" max="255" value="0" id="throttleSlider">
        </div>
```

```
        <div class="steering-container">
            <div class="steer-button" id="leftButton">&#9664;</div> <div class="steer-button"
id="rightButton">&#9654;</div> </div>
    </div>

    <script>
        const throttleSlider = document.getElementById('throttleSlider');
        const leftButton = document.getElementById('leftButton');
        const rightButton = document.getElementById('rightButton');
        const debugInfo = document.getElementById('debugInfo');

        // --- Configuration ---
        const UPDATE_INTERVAL_MS = 50; // Send updates faster for manual control
        const STEERING_SENSITIVITY = 0.6; // 60% power difference when steering

        let throttle = 0;
        let steerDirection = 0; // -1 for left, 0 for straight, 1 for right

        // Update throttle from slider
        throttleSlider.addEventListener('input', (event) => {
            throttle = parseInt(event.target.value, 10);
        });

        // --- Steering Button Event Listeners ---
        // We need to handle both mouse and touch events for wide compatibility

        // Left Button
        leftButton.addEventListener('mousedown', () => { steerDirection = -1; });
        leftButton.addEventListener('touchstart', (e) => { e.preventDefault(); steerDirection = -1;
});
        leftButton.addEventListener('mouseup', () => { steerDirection = 0; });
        leftButton.addEventListener('touchend', () => { steerDirection = 0; });
        leftButton.addEventListener('mouseleave', () => { steerDirection = 0; }); // Failsafe
```

```
// Right Button
rightButton.addEventListener('mousedown', () => { steerDirection = 1; });
rightButton.addEventListener('touchstart', (e) => { e.preventDefault(); steerDirection = 1;
});
rightButton.addEventListener('mouseup', () => { steerDirection = 0; });
rightButton.addEventListener('touchend', () => { steerDirection = 0; });
rightButton.addEventListener('mouseleave', () => { steerDirection = 0; }); // Failsafe

// --- Main Control Loop ---
setInterval(() => {
  let leftMotor = throttle;
  let rightMotor = throttle;

  if (steerDirection !== 0) {
    const steerEffect = throttle * STEERING_SENSITIVITY;
    if (steerDirection === -1) { // Turning Left
      leftMotor -= steerEffect;
      rightMotor += steerEffect;
    } else if (steerDirection === 1) { // Turning Right
      rightMotor -= steerEffect;
      leftMotor += steerEffect;
    }
  }

  // Clamp values to the 0-255 range
  leftMotor = Math.max(0, Math.min(255, Math.round(leftMotor)));
  rightMotor = Math.max(0, Math.min(255, Math.round(rightMotor)));

  // Send commands to ESP32
  fetch(`/set?motor=left&value=${leftMotor}`);
  fetch(`/set?motor=right&value=${rightMotor}`);

  // Update debug info
```

```
            debugInfo.innerText = `L: ${leftMotor} | R: ${rightMotor}`;
        }, UPDATE_INTERVAL_MS);
    </script>
</body>
</html>
)rawliteral";
// Handle root page
void handleRoot() {
  server.send_P(200, "text/html", htmlPage);
}

// Handle motor value update
void handleSetMotor() {
  if (server.hasArg("motor") && server.hasArg("value")) {
    String motor = server.arg("motor");
    int value = server.arg("value").toInt();
    value = constrain(value, 0, 255);

    if (motor == "left") {
      ledcWrite(gpLm, value);
    } else if (motor == "right") {
      ledcWrite(gpRm, value);
    }

    server.send(200, "text/plain", "OK");
  } else {
    server.send(400, "text/plain", "Missing args");
  }
}

void setup() {
  Serial.begin(115200);
  ledcWrite(gpLm, 0);  // Stop Left motor at start
  ledcWrite(gpRm, 0);  // Stop Right motor at start
```

```
  // Attach motors to PWM
  ledcAttach(gpLm, pwmFreq, pwmResolution);
  ledcAttach(gpRm, pwmFreq, pwmResolution);
  //ledcWrite(gpLm, 0);  // Stop Left motor at start
  //ledcWrite(gpRm, 0);  // Stop Right motor at start

  // Connect to Wi-Fi
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  // Start web server
  server.on("/", handleRoot);
  server.on("/set", handleSetMotor);
  server.begin();
  Serial.println("Web server started.");
}
```

5. Mobile Application Development

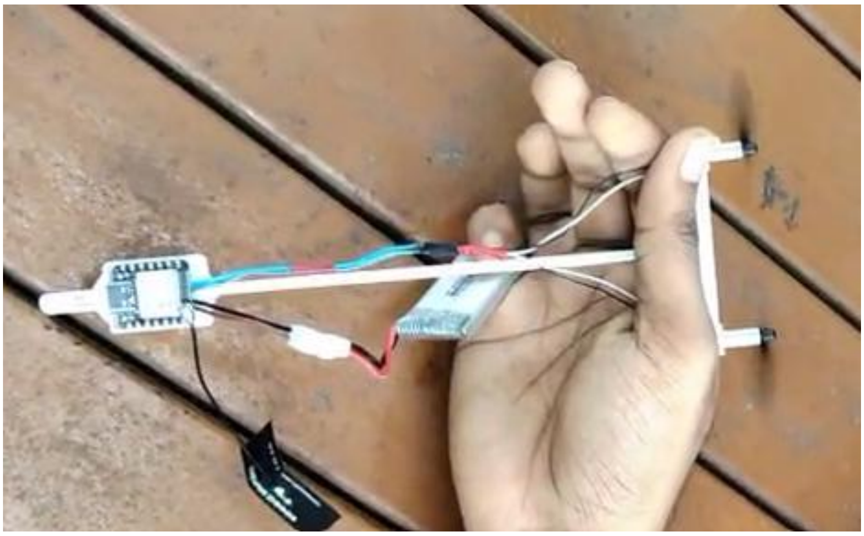
- Build a Flutter-based mobile application with a tilt-control mechanism using the smartphone’s accelerometer.
- Establish Wi-Fi connectivity between the app and ESP32-S3 to transmit real-time control signals.

6. System Integration

- o Test the wireless connection between the mobile app and the ESP32-S3.
- o Verify signal reception, motor responsiveness, and thrust generation.

7. Flight Testing and Optimization

- o Conduct trial flights to evaluate stability, responsiveness, and control reliability.
- o Optimize PCB layout, motor speed, and app sensitivity for smoother performance.



FINAL OUTCOME

The SKY-FI prototype is a lightweight dual-propeller flying model powered by the ESP32-S3 microcontroller. It uses a 3D-printed frame, coreless DC motors, and a Li-Po battery with TP4056 charging module. A custom PCB integrates the electronics, while a Flutter-based mobile app provides real-time tilt control via Wi-Fi (UDP), enabling stable and responsive flight.

FUTURE SCOPE

• Extended Range & Reliability: Upgrade from UDP to more reliable protocols like MQTT/TCP to improve control stability and reduce packet loss.

• Camera Integration: Add a lightweight ESP32-S3 camera module to enable live video streaming for FPV (First-Person View) flight experience.

• AI-based Flight Assistance: Implement basic AI/ML algorithms for self-stabilization, obstacle avoidance, or autonomous flight modes. 13

• Enhanced Power System: Use high-capacity Li-Po batteries or solar-assisted charging for longer flight durations.

CONCLUSION

The SKY-FI project successfully demonstrates how low-cost hardware and IoT technologies can be combined to create an innovative flying prototype. By integrating the ESP32-S3 microcontroller, custom PCB design, 3D-printed frame, and a Flutter based mobile app, the system achieves real-time wireless control with enhanced learning value. The prototype not only proves the feasibility of building a cost-effective alternative to commercial solutions but also serves as a powerful educational tool for students and hobbyists interested in embedded systems, robotics, and wireless communication.