Quadcopter Drone Project

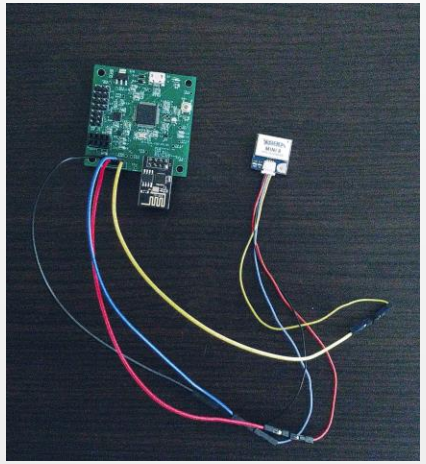
By: Zachary Heil, Lily Phu, Stephanie Phillips, Spencer Ward, Dishoungh White II

The Team

- CSCE Team
  - Zachary Heil: App Development
  - Lily Phu: App Development
  - Stephanie Phillips: ESP-8266 Backend Development
  - Spencer Ward: ESP-8266 Backend Development
  - Dishoungh White II: Peripheral Testing & Betaflight Configuration
- ELEG collaborated on this project

Overview

- The project is a collaborative project between the Computer Science/Computer Engineering (CSCE) and Electrical Engineering (ELEG) departments to create a feasible replacement for the CC3D drone controller.
- The EE team: schematics, printing/soldering boards, hover, battery protection unit, and control signals.
- The design and implementation on the CSCE side was broken down into 4 key phases:
  1. PCB Design
  2. Peripheral Testing
  3. Betaflight Configuration
  4. App development
- Secondary Objective: Create an app that can control the drone and receive sensor data.

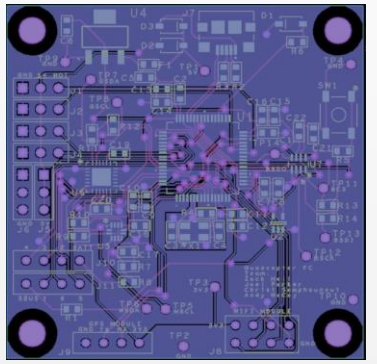


Purpose

- The flight controller our board is based on is the CC3D, which is a very inexpensive board. However, it does not currently support the more advanced features modern boards have such as:
  - Global Positioning System (GPS) module
  - Barometric capability to determine altitude
  - Magnetometer for magnetic calibration
- There is also a lack of important documentation on the official Betaflight repositories on how to configure flight controllers with custom pinouts. Completing this project will provide clearer documentation on that process for future drone hobbyists.

Implementation (PCB Design)

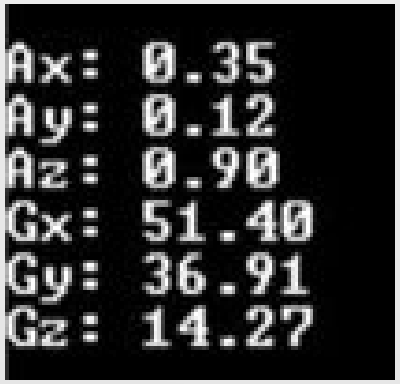
- The board consists of:
  - Microcontroller Unit (MCU): STM32F405RG
  - Barometer: BMP280
  - Inertial Measurement Unit (IMU): MPU6050
  - Magnetometer: LIS2MDLTR
  - 3.3V Voltage Regulation Circuit
  - Drone 3-cell and 4-cell Li battery detection circuit (for percentage remaining)
  - Data Lines for Universal Serial Bus (USB) Connection
  - Crystal Oscillator (16MHz): OSC-X322516MLB4SI
  - Optional WIFI Co-processor headers (ESP8266) for mobile app capability
  - Optional headers for user to add a GPS module
  - User-Assignable GPIO (for LEDs, alert buzzers, servos, etc)



Implementation (Peripheral Testing)

- We primarily used STM32 Cube Integrated Development Environment (IDE) to create test drivers for the microcontroller's processor and the peripherals.
- Before configuring our board with Betaflight, we tested its ability to communicate via serial port via USB communication functions. Then, we tested the BMP, MPU, LIS2, and ADC (Analog-to-Digital Converter) voltage readings in that order.
- Used the Olimex STM32-P405 (red board) as a test board to tinker with while ELEG incremented upon our production (green) boards.

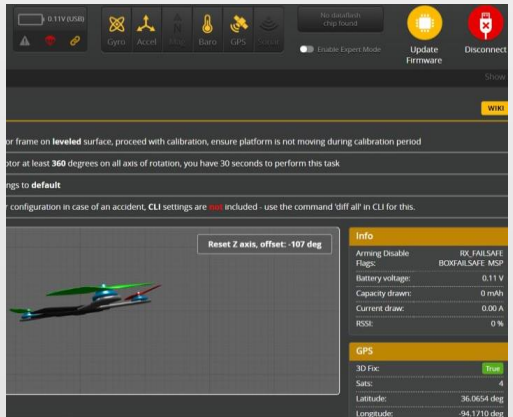
```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  while(!bmp280_is_measuring(&bmp280));

  while(!bmp280_read_float(&bmp280, &temperature, &pressure, &altitude))
  {
      sprintf((char *)data, "Temperature/Pressure Reading Failed\r\n\n");
      CDC_Transmit_FS(data, sizeof(data));
      memset(data, 0, sizeof(data));
  }

  sprintf((char *)data, "Pressure: %.2f Pa \r\nTemperature: %.2f C \r\nAltitude: %
  CDC_Transmit_FS(data, sizeof(data));
  memset(data, 0, sizeof(data));
  HAL_Delay(1000);
```



STM32 CubeIDE

Implementation (Betaflight Configuration)

- After the main peripherals on the production (green) board were complete, we began configuring Betaflight, which is a flight controller firmware.
- It seemed like a daunting task to tinker with Betaflight's real-time operating system (RTOS) and peripheral drivers, especially with our big challenge of accessing the green boards with COVID-19 restrictions.
- Fortunately, Betaflight developed a graphical user interface (GUI) alongside with a command-line to make the configuration process easier.



Implementation (App)

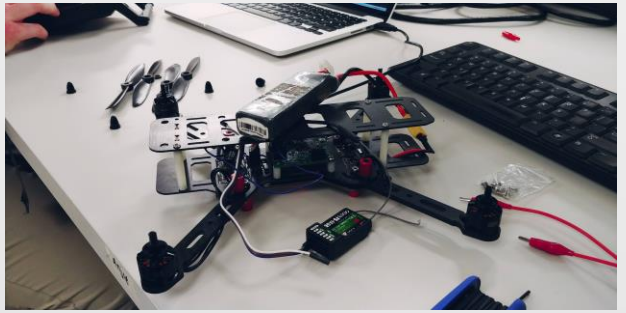
- App Stacks
  - React Native, Expo, Typescript
- Limitations on creating the App
  - Phone issues as well as computer web not being able to utilize location data
- Implementations
  - HTTP requests to send/receive to/from a mock drone HTTP server / ESP8266 on the drone. This includes
    - Problem: HTTP Requests are too slow and utilize too much information size
  - TCP/UDP sockets and let the ESP8266 utilize MSP protocol to translate data to betaflight
    - Problem: Time limitation

Demo (Drone Flight)

Demo (App)

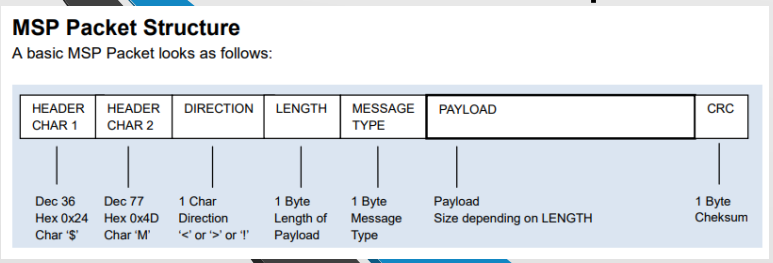
Future Work (Drone)

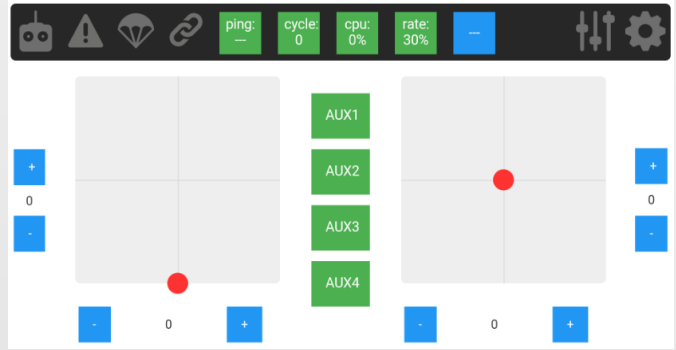
- Not a major problem, but every time the board loses power, the Betaflight firmware doesn't persist unless if we reset the board
  - Future work could be done to have Betaflight persist in the controller board instead of reconfiguring back to the previous state each time.
- One other feature set that could be added in the future is a Light Emitting Diode (LED) strip and a beeper to notify the pilot of critical statuses.
  - This could simply be added with more tweaks in the Betaflight configurator.



Future Work (App)

- Originally, we wanted to create an app to control the drone and get sensor data through the ESP8266 chip from/to our app.
- There is a Cordova-based app on Github that is like how we envisioned our app. The app does connect to the Wi-Fi chip and the drone can be controlled from the app, which proves the implementation is possible.
- Due to time constraints, we decided to create an HTTP mock server that connects to a react-native client.
- MSP protocol could be used to send sensor data back





References

- UARK Quadcopter Flight Controller Github Org, https://github.com/UARK-Quadcopter-Flight-Controller
- Project Website, https://wordpressua.uark.edu/capstone/fall-spring-2020-2021/teams-1-5-f20/quadcopter_drone_controller/
- ESP8266 Wifi-Serial Bridge (esp-link), https://github.com/jeelabs/esp-link
- MSP/TCP Controller App, https://github.com/cs8425/msp-controller
- Drone Demo Video, https://youtu.be/y8RX5VHPSSM
- App Demo Video, https://youtu.be/L9fV5bQ89GM