Drone

Software Architecture Design

SAD Version 3.0

Team #02
16 November 2021

Stephanus Huang
Sam Cox
Tatsuya Saito
Andy Thornhill
Steven Chen (Manager)

CS 460 Software Engineering

TABLE OF CONTENTS

1 Introduction

The Software Requirements Specification Report (SRS) formalized the software requirements of the drone. It is crucial for the software to be organized and designed in a clear, secure, and modular style while maintaining simplicity in order to ensure that all needs of the drone are met.

The objective of this report is to provide a complete and detailed architectural design of the software components used to control the drone. All components including LED, battery sensor, radio, altimeter, GPS, ESCs, motors, IMU, collision sensors, storage, and camera will be discussed.

The remainder of the report contains a detailed description of the software architecture. Section 2 provides a design overview with a design diagram of the software. Section 3 contains a detailed overview of all software components. Section 4 details sample use cases of the drone, and lastly Section 5 mentions some implementation constraints.

2 Design Overview

The software for the drone requires several components. Some of these components will communicate with each other in order to provide the necessary functionality for the drone. The following diagram illustrates the organization of the components and how they will interact with one another.

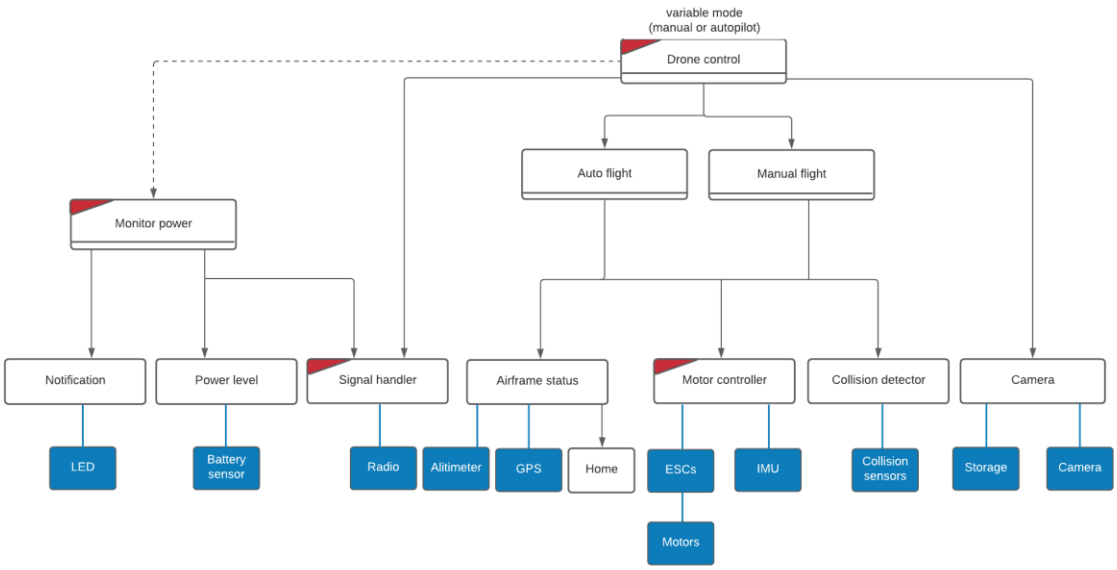

Figure 1
A design diagram of the Drone

The Drone control holds the “Auto-flight” and “Manual-flight” procedures. These procedures access the same set of objects: “airframe status”, “motor controller”, and “collision detector”. Manual-flight receives instructions through the signal handler object.

Additionally, the drone is able to take pictures using the camera object and the instructions received through the signal handler object.

The "Monitor power" procedure is required to monitor the battery level. A notification object under the "Monitor power" procedure indicates this information through LED's colors on the drone. Additionally, it notifies the remote controller of the battery level of the drone through the signal handler object.

- "Auto flight" and "Manual flight" handle all flying-related features of the drone with the "Motor controller", "Collision detector", and "Airframe status".
- "Airframe status" contains features pertaining to the status of the drone. It contains the altimeter which provides the current altitude of the drone, as well as the GPS which provides the coordinate information of the drone. Additionally, the Home object is contained here.
- "Motor controller" deals with the ESCs to set the speed of the motors and move the propellers to let the drone take off, hover, land, turn right and left, and rotate clockwise and counterclockwise. It also uses information from the IMU for the flying condition of the drone by utilizing the gyroscope, accelerometer, and magnetometer.
- "Collision detector" encompasses the directional collision sensors aboard the drone. These sensors will notify the drone if there is terrain around or under the drone.
- "Camera" handles the case when users want to take a photo. This will allow the camera to capture an image, and save the image to the SD card storage.
- "Notification" handles the LED color to indicate the current On/Off status and the battery status of the drone.

3 Component Specifications

Each component in the software for the drone has its own specific design and intended usage. In this section, each of the components will be described in further detail to ensure all requirements of the drone are met.

Notification
- void setLEDColor(boolean isBatLow)

PowerLevel
- boolean powerOK()
- boolean isStateChanged()

Signal handler
- void receiveSignal(String command)
- void sendSignal()

Airframe status
- Coordinate getCurrCoord()

- double getCurrAltitude()

Home
- void setHome()
- void returnHome(double latitude, double longitude, double altitude)

Motor controller
- void moveDrone(Vector vector)
- void holdCurrentVector(Vector vector)
- void rotateDrone(char direction)
- double getCurrFacingDirection()
- double getAngularDegree()

Collision Detector
- double notifyHorizontalTerrain()
- double notifyVerticalTerrain()

Camera
- Image captureImage()
- boolean storeImage(Image pic)

4 Sample Use Cases

The drone has many features and potential use cases. This section will provide some sample use cases of the drone software.

- When the user presses the capture button on the remote controller, the microcontroller on the remote controller causes the radio transceiver to send a signal to the drone. If storage is available, the drone will activate the camera and capture an image. The software will then take the image from the camera and place it into the onboard storage.
- The radio transceiver aboard both the drone and remote controller is capable of receiving and transmitting signals. Specifically, “sendSignal()” and “receiveSignal()” receive and transmit control signals from the drone to the remote controller.
- On initial takeoff when the drone is on the ground and the user gives the drone an “up” command once, the “moveDrone(up)” method of a motors controller object is called. This method makes the drone go up and hover there. In addition, the “setHome()” method is called. The coordinates of the take-off position are received from GPS and the altitude from the altimeter are both stored.

5 Implementation Constraints

The drone has some constraints in the implementation of all the features. This section will list the constraints known on the implementation of the features.

- The effect of wind is not considered when we implement the program.
- The implementation does not consider how high the drone can fly.
- The camera can only take a static picture and cannot take videos.