Drone

Software Requirement Specification

SRS Version 2.0

Team #1

2nd November 2021

Saphal Karki [Manager]
Mausam Shrestha
Safal Poudel
Karan Aryal
Rajesh Upadhayaya

CS 460 Software Engineering

TABLE OF CONTENTS

1. Introduction

This document provides a detailed description of the features, interfaces involved, system design and design constraints for the control software of the Drone system.

1.1 Purpose

The purpose of this document is to provide the details on the software mechanics of the system and how it operates along with the environment interactions. It also provides a technical reformulation of the constraints to the software and acts as a baseline for all software development activities. This document can be used as a building framework for software design, testing and related technical studies.

1.2 Intended Audience

The nature of this document is highly technical in nature. It lays the foundation for the specifics of a working system design in a software level, and as such is intended for people with technical backgrounds in the team like software developers, and engineers.

1.3 Document Overview

This document is organized into 5 different sections to provide a concrete overview of the software system design. The 1st section is the Introduction that explains the nature and the purpose of this document. The 2nd section is the General Description that provides a block diagram of the physical interfaces involved, as well a short description of the involved physical components. The 3rd section is the Logical diagram which provides a visual representation of the logical interfaces involved in the system. The 4th section is the System State that provides the information about all of the states of the system which is necessary for decision making , and operating the drone system, The 5th section is the functional requirements which describes all the possible events generated as a result of user interaction and system operation.

2. General Description

This section provides a description of all the physical components involved in the system design. These physical interfaces are represented in a block diagram which provides a highlevel overview of how these different interfaces are laid out and connected concerning the construction of the drone system.

2.1 Block Diagram

This section consists of two diagrams that represent the physical interfaces involved in the drone and the remote controller. The first diagram represents the physical interfaces involved in the Drone and the second diagram represents the physical interfaces involved in the Remote Controller.

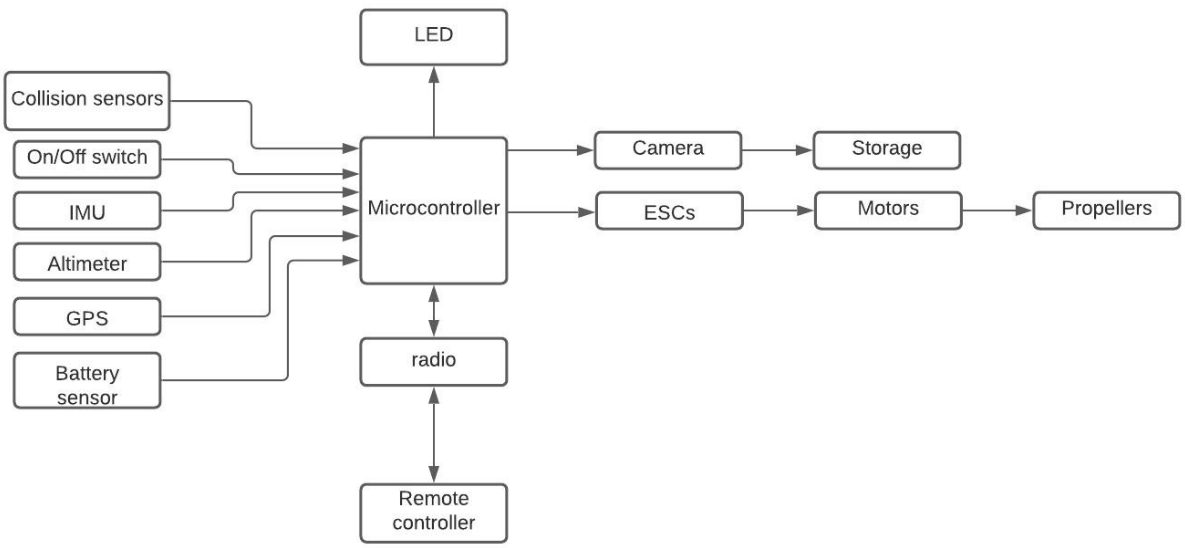

Figure 1: Block Diagram for Drone system

2.2 Components

The Drone system consists of different types of hardware components, which are the building blocks of system design. These physical components need to be utilized intelligently by the control software to provide a working mechanism for the drone system.

1. On/Off Switch – turns the system on or off.
2. Propeller – provides flight capabilities.
3. Motor – helps to spin the propellers
4. Electronic Speed Controller(ESC) – helps to adjust the rotational speed of the motors.
5. Collision Sensor – detects objects in the collision radius.
6. Inertial Measurement Unit(IMU) –
    - Gyroscope – measures the rotational attributes of the drone (Roll, Pitch & Yaw).
    - Accelerometer – determines the acceleration of the drone in all three dimensions.
    - Magentometer – establishes cardinal direction for the drone.
7. Altimeter – detects the current altitude of the drone.
8. GPS – detects the location of the drone.
9. Battery Sensor – tracks the power status of the battery.

10. LED – signals different types of events.
11. Radio – transceiver module to send and receive signals.
12. Camera – used only for taking pictures.
13. Microcontroller – flight controller that manages the navigation of the drone.
14. Remotecontroller – controller for users to operate the drone.

3. Logical Diagram

This section shows the logical diagram for the Drone system. This diagram shows the logical interfaces involved in the system, and how it interacts with the Drone controller.

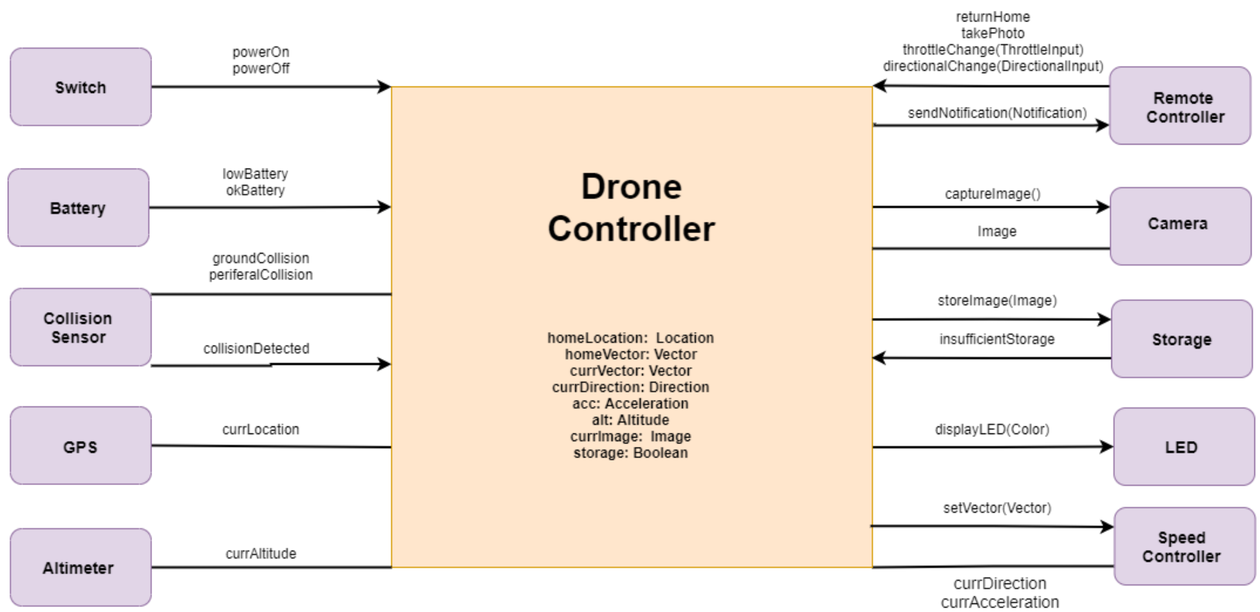

Figure 2: Logical Diagram

3.1 Logical Interfaces

This section lists the logical interfaces involved in the system design, along with a brief overview of their primary use and functionality.

1. Switch – This interface communicates if the system is on or off. It fires the following events.
    - powerOn
    - powerOff
2. Battery – Tracks the battery status, and fires events signifying the power status of the battery. It fires the following events:
    - lowBattery
    - okBattery
3. Collision Sensor – This interface consists of two values that the drone controller can read to find if any collision sensors have possible collision. It also fires a collision detection event if any of the collision sensors detect foreign object or terrain in their proximity radius.

- groundCollision
- periferalCollision
- collisionDetected

4. GPS – interface from which the controller can read the current location of the drone. It contains the following value:
   - currLocation
5. Altimeter – interface from which the controller can read the current altitude of the drone. It contains the following value:
   - currAltitude
6. Camera – This interface recieves command from the drone controller to capture an image. The captured image can then be read by the controller. It contains the following values:
   - Image
7. Storage – This interfaces recieves commands from the drone controller to store the captured images. It then fires an event in the case of failure to store the image due to insufficient storage.
   - insufficientStorage
8. LED – This interface recieves commands from the controller to display different LED lights corresponding to different events.
9. Speed Controller – This interface recieves commands from the controller to make necessary adjustments to adjust the speed of the electric motors to match the vector input.
10. Remote Controller – Interface that transmits different types of commands as requested by the drone operator (or user). It also recieves notification signals from the drone controller. It issues the following commands:
    - returnHome
    - takePhoto
    - throttleChange(ThrottleInput)
    - directionalChange(DirectionalInput)
11. Drone Controller –This interface is the brain of the drone system and is responsible for proper navigation, and operation of the Drone. It is reponsible for intelligently interpreting events generated in system operation, and for issuing necessary commands to the different logical interfaces to maintain the requested system state by the user. It issues the following commands:
    - sendNotification(Notification)
    - captureImage()
    - storeImage(Image)
    - displayLED(Color)
    - setVector(Vector)

4. System State

The Drone system consists of different states that describe the behavior of the system at a particular time. These states capture the information about the drone which is required by the microcontroller to make control decisions.

1. **State:** battery of type *Int*

   This state represents the current battery power of the drone (0 - 100). The following events can change this state:

   Event: powerOn
   - --Battery until the event powerOff is fired.
   - The battery continues to decrease until the drone is switched off.

2. **State:** homeLocation of type *Location*

   This state represents the location where the drone was switched on. The following events can change the state:

   Event: powerOn
   - homeLocation = currLocation
   - The drone controller reads the value of currLocation from the GPS, and stores the value in homeLocation.

   Event: powerOff
   - homeLocation = null
   - The drone resets the value of homeLocation to null, when the power is turned off.

3. **State:** acc of type *Acceleration*

   This state gives information about the acceleration of the drone in all 3 dimensions. The following events can change this state:

   Event: throttleChange(ThrottleInput)
   - acc = currAcceleration
   - The drone controller reads the value of currAcceleration from the speed controller and sets the value of Acc to currAcceleration.

   Event: directionalChange(DirectionalInput)
   - acc = currAcceleration
   - The drone controller reads the value of currAcceleration from the speed of the controller and sets the value of Acc to currAcceleration.

4. **State:** alt of type Altitude

   This states gives information about the altitude of the drone. The following events can change this state:

Event: throttleChange(ThrottleInput)

- Alt = currAltitude
- The drone controller reads the value of currAltitude from the Altimeter and sets the value of alt to currAltitude.

5. **State**: currentDirection of type *Direction*

This state represents the current direction the head of the drone is facing. The following events can change this state:

Event: directionalChange(DirectionalInput)

- If (Direction != currDirection) currentDirection = currDirection
- The drone controller reads the value of currDirection from the speed controller and then sets the value of currentDirection to currDirection.

6. **State:** homeVector of type Vector

This state represents a pointer to the homeLocation. The following events can change this state:

Event: powerOn

- homeVector = new Vector(homeLocation)
- The drone controller generates a new vector to keep track of the homelocation by pointing a towards homeLocation.

7. **State:** currentVector of type Vector

This state represents the point where the drone is currently headed to. The following events can change this state:

Event: setVector(Vector)

- currentVector = Vector
- The drone commands the speed controller to make the necessaru adjustments to match the new vector.

8. **State:** storage of type *Boolean*

This state represents if the drone controller has sufficient storage space to store any additional photos. The following events can change this state:

Event: insufficientStorage

- storage = false

- The drone controller sets Storage to false, and blocks any further commands to store an Image until space is made available.

5. Functional Requirements

This section describes the functional requirements of the different events generated during system operation, environment and user interaction. The events are described in detailed below:

1. powerOn

   This event is fired when the switch is turned on. This event signals the drone controller to configure the initial state of the drone.
   - Battery = battery power level
   - homeLocation = currLocation;
   - acc = null;
   - alt = currAltitude;
   - currentDirection = currDirection;
   - homeVector = new Vector(homeLocation);
   - currentVector = new Vectore(currDirection);
   - Storage = true;

2. powerOff

   This event is fired when the switch is turned off.

3. lowBattery

   This event is fired when the battery state is < 20. This event signals the drone controller to block user commands. The drone controller then sends the *sendNotification(Notification.LOW_BATTERY)* to the remote controller, which inturn automatically fires the returnHome event, to signal the drone to return to the homeLocation.

4. okBattery

   This event is fired as soon as the battery state is >= 20. This event signals the drone to unblock user commands.

5. collisionDetected

   This event is fired if any of the 5 collision sensors onboard detected any foreign object or terrain in their proximity radius. When the drone controller receives this event, it reads the value of groundProxmity and periferalProximity which returns the collision sensor detecting the event.

6. returnHome

   This event is fired by the remote Controller which is interpreted by the drone controller to make necessary adjustments to return to the homeLocation. The drone controller does this by first settting the currentVector to the homeVector(i.e. currentVector = homeVector), and then speeds up in the direction of the homeLocation until it reaches its destination.

7. takePhoto

This event is fired by the remote controller, when the user presses the button to take a picture. The drone controller will issue a *captureImage()* command to the camera to take a picture which is then read by the controller. After reading, the image the drone controller will attempt to store the Image by sending out a *storeImage(Image)* command to the Storage.

8. throttleChange(ThrottleInput)

The throttle control stick in the remote controller is mapped to ThrottleInput which is passed in as an argument to the *throttleChange(ThrottleInput)* event. This event is fired when the current position of the throttle control stick is changed by $1/5^{th}$ of the total available movement length. ThrotteInput is a one-dimensional Vector and has the following two properties.

- direction: up || down (signifies upward or downward movement)
- speed: speed in z-axis which is corresponds to the length of stick movement (+ve if direction = 'up' and -ve if direction = 'down')

9. directionalChange(DirectionalInput)

The directional control stick in the remote controller is mapped to DirectionalInput which is passed as an argument when the *directionalChange(DirectionalInput)* event is fired. This event is fired when the direction control rod changes its distance by $1/5^{th}$ of the diameter of the circle where the directional control rod operates. Directional Input has the following four values:

- xSpeed: speed in x-axis (+ve if xCoord is in +ve x-axis and negative otherwise)
- ySpeed: speed in y-axis (+ve is yCoord is in -ve y-axis and negative otherwise)
- xCoord: x-coordinate of directional control stick
- yCoord: y-coordinate of directional control stick

10. sendNotification(Notification)

This event is fired when the drone attempts to send a notification to the Remote Controller. The Notification to be sent is passed in as an argument to the event which has the following types:

- Notification.LOW_BATTERY
- Notification.OK_BATTERY

11. captureImage()

This event is fired after the drone controller recieves commands from the remote controller to take a picture. After listening this event, the drone commands the Camera to take a picture using *captureImage().*

12. storeImage(Image)

    This event is fired after the drone controller reads the captured image from the Camera. It then passes this Image as an argument to the *storeImage(Image)* event to store the Image in the Storage.

13. setVector(Vector)

    This event is generated after the drone controller makes the necessary adjustments to the current vector based on the inputs it recieves from the remote controller via *throttleChange(ThrottleInput)* and *directionalChange(DirectionalInput)* events. After the adjustments are made it commands the speed controllers to match this newly adjusted vector via the setVector(Vector) event where the new adjusted Vector is passed in as argument.