Requirements Definition

MARC BIGLER

Tello Drone

1. Introduction and Context

The purpose of this project is to connect to a drone and be able to control it from the computer. To communicate between the drone and the computer a socket will be open to allow data transfer. Once the Communication Center (aka Computer) and the drone have a connection the Communication Center will direct the drone on specific missions. To start off there will be three pre-programmed missions. From the keyboard you type number “1”,”2”, or “3” to command the drone for the specific mission. If the connection is lost at any time the Tello drone has a 15 second window before is auto lands. This is built into the drone and the the Communication Center. The main function will have the objects needed to control the drone. There will be a CommunicationCenter that relays messages to the drone. The individual Missions are inherited from a virtual Mission class so that each mission has to have its own mission.

This program also has the ability to run a simulator that communicates with the flyer. There are two different ports that provide was to communicate with the simulator. The main port that takes and receives commands will be handled on port 8889. The simulator will start a thread that sends a continuous message of its state. This message will be sent from port 8890. This message will be sent every 100 ms.

This has been a challenging experience writing this program in c++ but knowing that I will be working in c++ after I graduate is the reason why I have chosen this path.

2. **Upgrades to the drone:**

#1 - Implement the tracking of periodic status messages from the drone, so the flier always has a relatively up-to-date snapshot of the drone's state

-Threads are send from the Simulator and Main

#2 - Use drone state information to improve the execution of missions.

-The done won't do the commands unless it is in the air

#3- Implement an internal estimation of the drone's position based on completed maneuvers.

-The message is returned to the flyer in a string every few seconds

#6 - Implement potential configuration values (e.g., port numbers, timeouts, maximum retries, etc.) so they can be easily changed at runtime.

-These are added in the command line arguments

#9 - Ensure that your component responsible for network communications can be used by both the Drone Flier and Simulator.

-The sim and main both us the "Communication Center"

#10 - Ensure that your components responsible for message serialization, de-serialization, and validation can be used by both the Drone Flier and Simulator.

-Both can use

#11 - Ensure that your component responsible for drone state can be used by both the Drone Flier and Simulator.

-Both can use

3. **Users and Their Goals**

Types of users

- **Communication Center** (See Figure)
    - Goals: Be able to control the drone and command drone to fly specific missions.
- **Drone** (See Figure)
    - Goals: Accept command given from the Communication Center and pass info back to the Communication Center depending if there was an “error” or if the command was “ok”
- **MissionCollection** (See Figure)
    - Goals: Hold a collection of all the different missions
- **Mission** (See Figure)
    - Goals: This is a virtual class that will be used for each mission
- **Mission1** (See Figure)
    - Goals: This is a specific mission inherited from the Mission
- **Mission2** (See Figure)
    - Goals: This is a specific mission inherited from the Mission
- **Mission3**(See Figure)
    - Goals: This is a specific mission inherited from the Mission
- **Socket Connection** (See Figure)
    - Goals: Pass information from the computer to the drone and allow the drone to pass information back though WIFI.
- **UserInterface**(See Figure)
    - Goals: Take control of the involvement with the user.
- **Status**(See Figure)

- Goals: The class is to contain the initial status of the drone, it will be used to improve the flyer.

- **Classes for each command**(See Figure)
  - Goals: be able to access any information on any given command to make everything in a centralized location

3. Functional Requirements

FLYER

1. Connect to Drone
2. The Communication center both need to send and receive messages.
3. receive message from sim/drone every 100ms

SIMULATOR

1. Connect to flyer
2. The Communication center both need to send and receive messages.
3. Send message to flyer every 100ms

4. Non-functional Requirements

a. Application-level logic will be tested .

b. Unit tests should cover 100% of logic classes

c. UX tests will be tested by me, so that the software design is working as intended (not including all possible fringe cases)

5. Insight into construction the code.

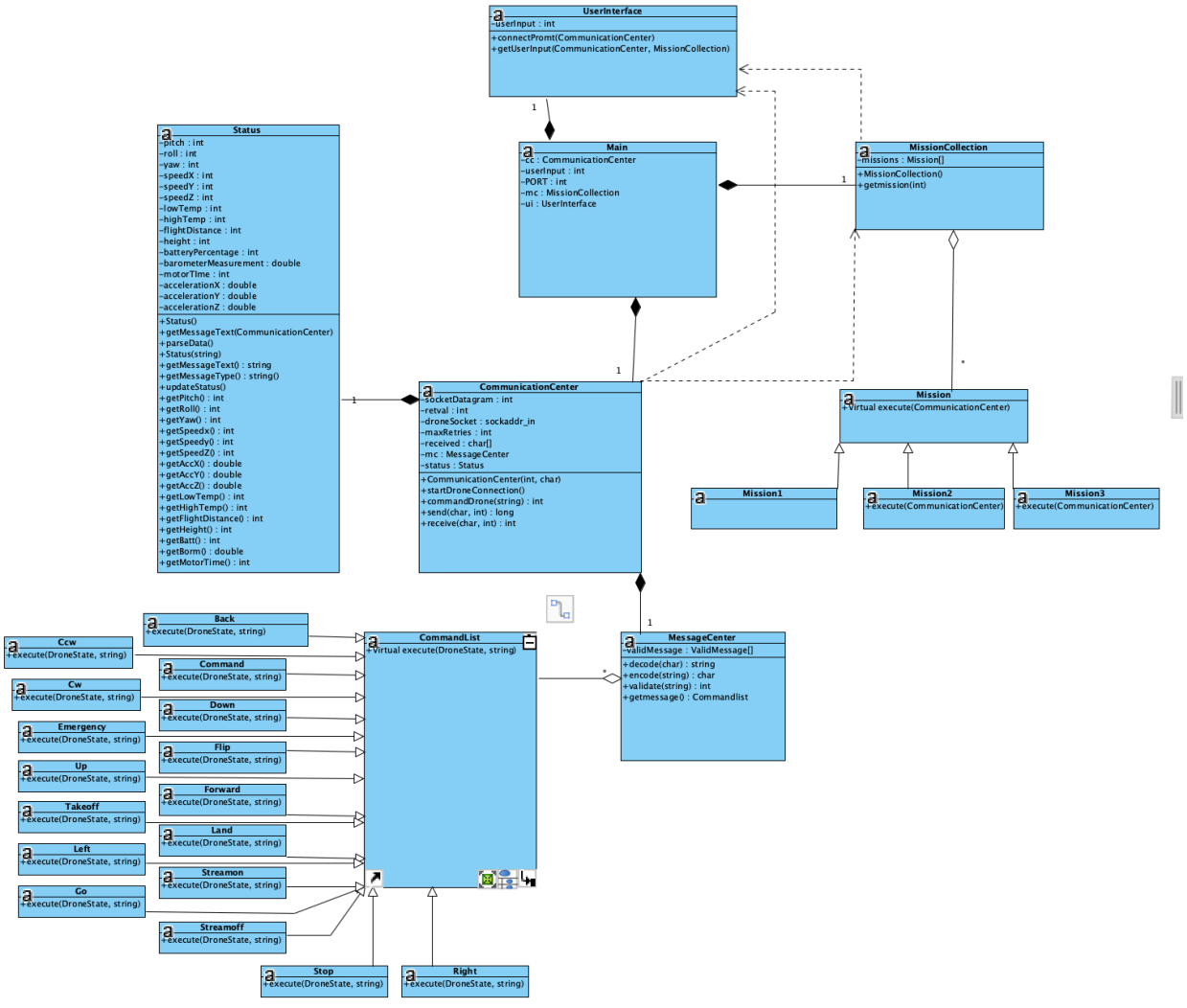
When the code was getting started it we difficult to decide the the best approach of development. At first I wanted to make the UML diagram form around the Drone but quickly determined that it was not such a great idea. I came to the conclusion that the UML needed to be centered around come sort of command center so I made a “Communication Center” and allowed this class to control the drone.

One of the biggest challenges that I had was the integration of the unit tests. It took quite a while to decide what unit test platform I wanted to use and even longer to figure out how it was installed and properly integrated into Cmake. There is still that I need to learn about “Catch 2” but now that everything is integrated it is much easier to start running forward.

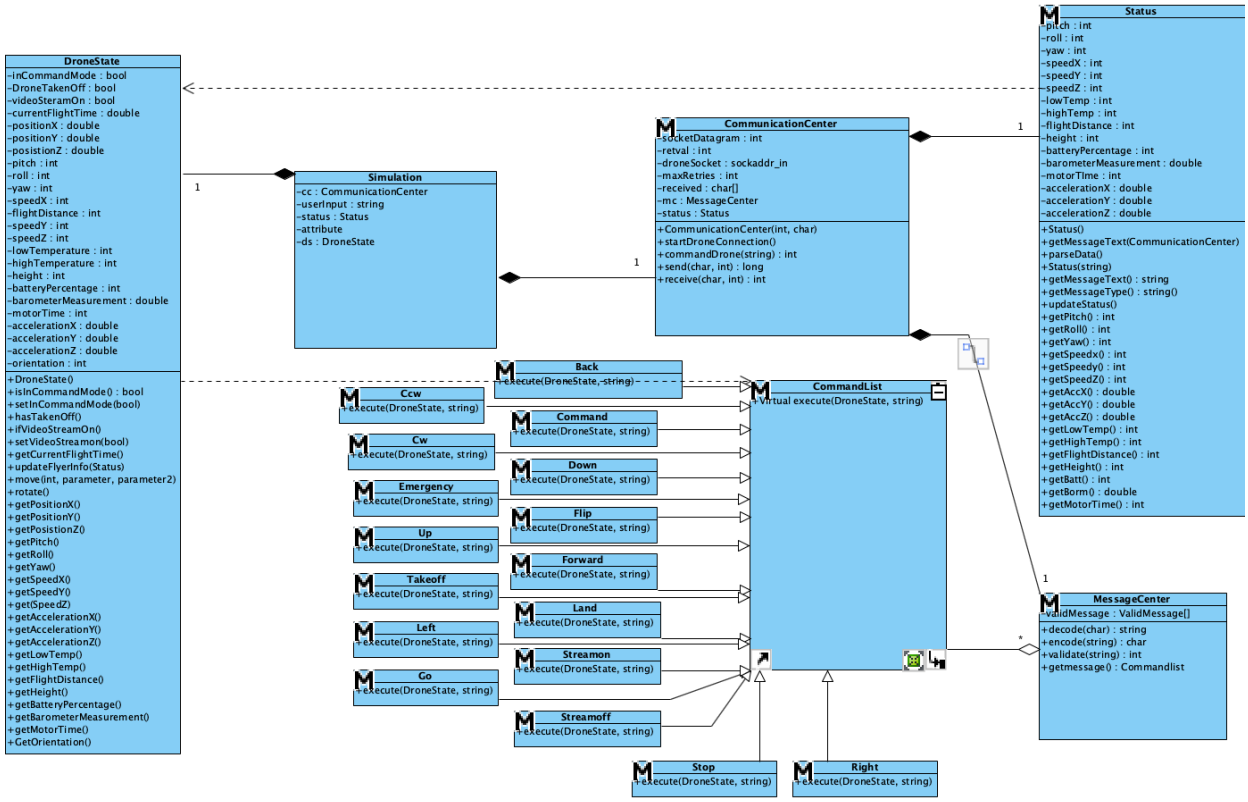
After sprint 1 I continued to run into difficulties with understanding the toolchain of the mac OS. The first two week was spend trying to figure out how to make threads work on a mac. The solution was as simple adding two lines to the make file but took a bit to figure it out because Clang would let me create threads with static functions but this was not what I needed to make the program run correctly. Although is was frustrating at the time, It was a great learning experience. After gaining more experience with using the Cmake I am sure that Catch 2 can be included as a outside library and plan to go back and fix this issue so that when the program is built the user will not have to download this on their own.

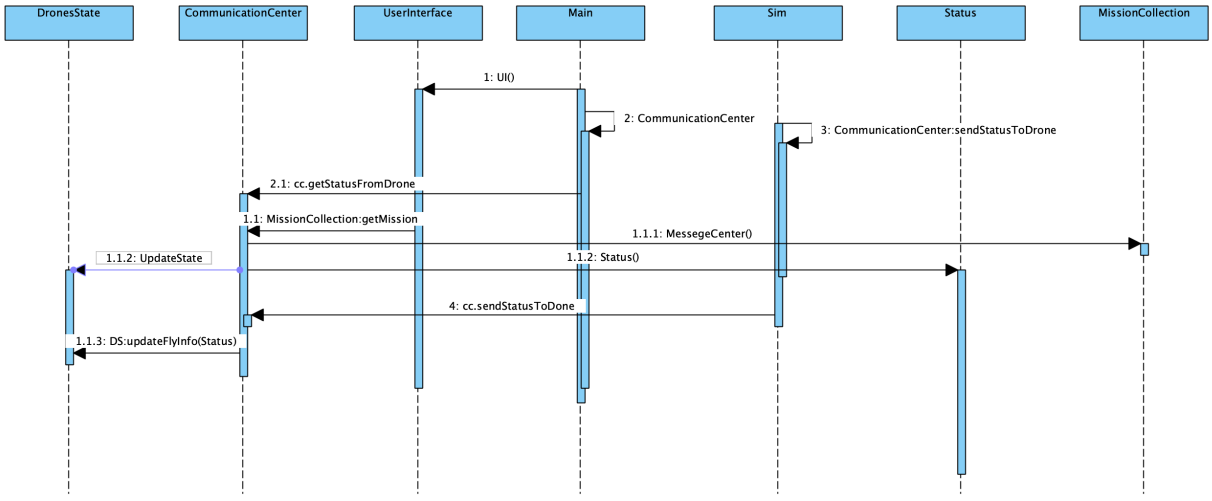
6. Diagrams

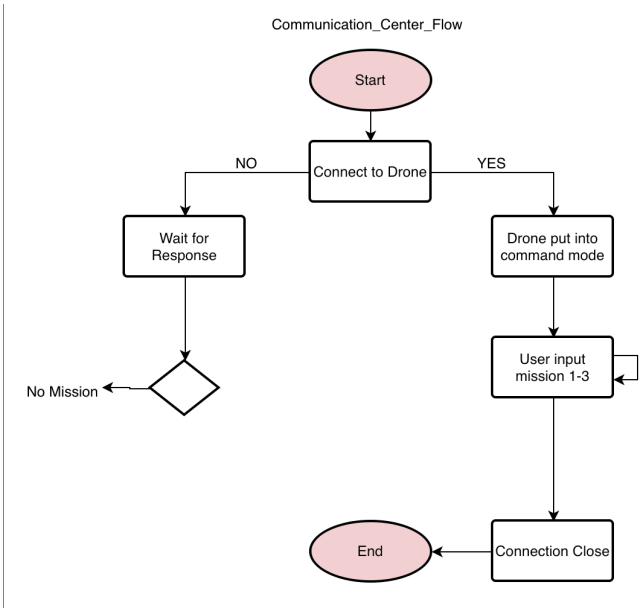
MAIN

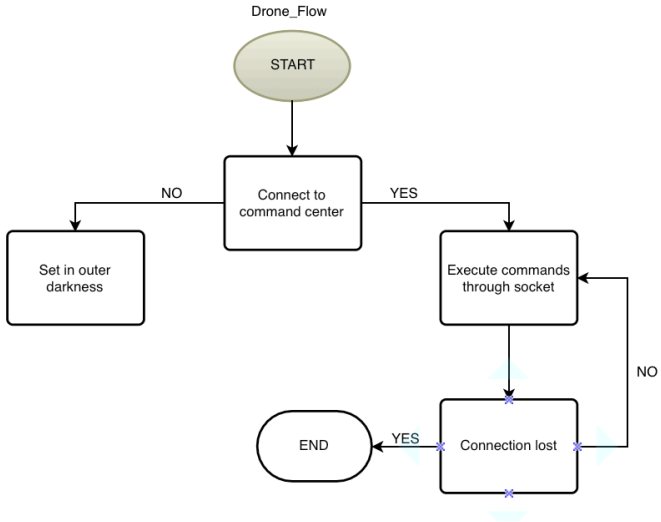


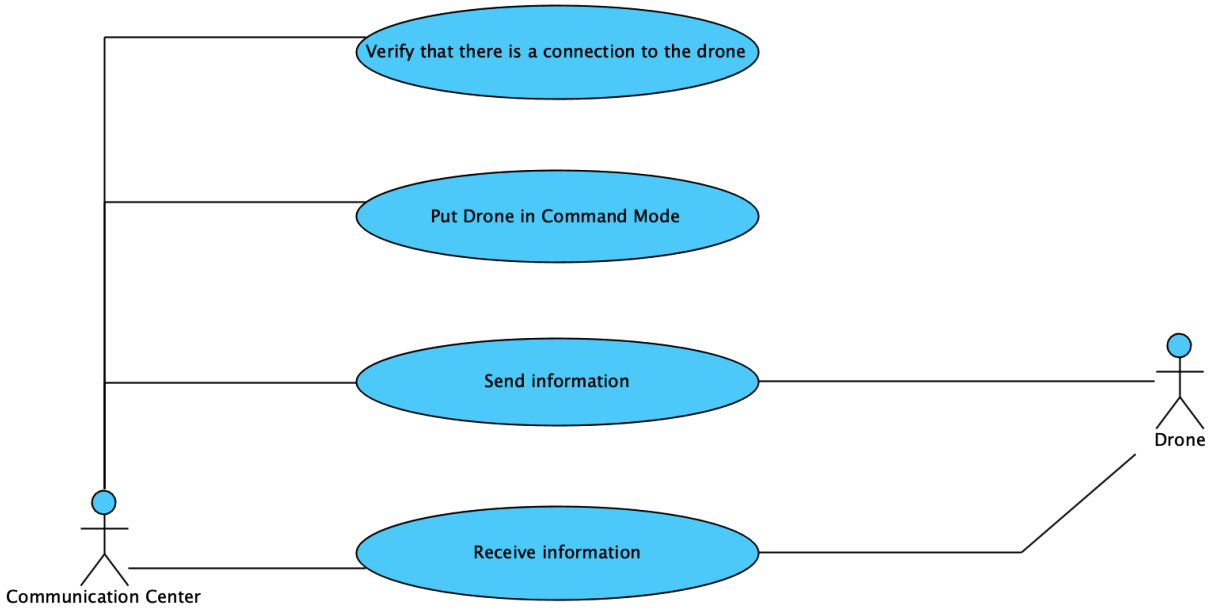
SIM


DroneState
-inCommandMode : bool
-DroneTakenOff : bool
-videoSteramOn : bool
-currentFlightTime : double
-positionX : double
-positionY : double
-posistionZ : double
-pitch : int
-roll : int
-yaw : int
-speedX : int
-flightDistance : int
-speedY : int
-speedZ : int
-lowTemperature : int
-highTemperature : int
-height : int
-batteryPercentage : int
-barometerMeasurement : double
-motorTime : int
-accelerationX : double
-accelerationY : double
-accelerationZ : double
-orientation : int
+DroneState()
+isInCommandMode() : bool
+setInCommandMode(bool)
+hasTakenOff()
+ifVideoStreamOn()
+setVideoStreamon(bool)
+getCurrentFlightTime()
+updateFlyerInfo(Status)
+move(int, parameter, parameter2)
+rotate()
+getPositionX()
+getPositionY()
+getPosistionZ()
+getPitch()
+getRoll()
+getYaw()
+getSpeedX()
+getSpeedY()
+get(SpeedZ)
+getAccelerationX()
+getAccelerationY()
+getAccelerationZ()
+getLowTemp()
+getHighTemp()
+getFlightDistance()
+getHeight()
+getBatteryPercentage()
+getBarometerMeasurement()
+getMotorTime()
+GetOrientation()
Simulation
-cc : CommunicationCenter
-userInput : string
-status : Status
-attribute
-ds : DroneState
CommunicationCenter
-socketDatagram : int
-retval : int
-droneSocket : sockaddr_in
-maxRetries : int
-received : char[]
-mc : MessageCenter
-status : Status
+CommunicationCenter(int, char)
+startDroneConnection()
+commandDrone(string) : int
+send(char, int) : long
+receive(char, int) : int
Status
-pitch : int
-roll : int
-yaw : int
-speedX : int
-speedY : int
-speedZ : int
-lowTemp : int
-highTemp : int
-flightDistance : int
-height : int
-batteryPercentage : int
-barometerMeasurement : double
-motorTIme : int
-accelerationX : double
-accelerationY : double
-accelerationZ : double
+Status()
+getMessageText(CommunicationCenter)
+parseData()
+Status(string)
+getMessageText() : string
+getMessageType() : string()
+updateStatus()
+getPitch() : int
+getRoll() : int
+getYaw() : int
+getSpeedx() : int
+getSpeedy() : int
+getSpeedZ() : int
+getAccX() : double
+getAccY() : double
+getAccZ() : double
+getLowTemp() : int
+getHighTemp() : int
+getFlightDistance() : int
+getHeight() : int
+getBatt() : int
+getBorm() : double
+getMotorTime() : int
CommandList
+virtual execute(DroneState, string)
MessageCenter
-validMessage : ValidMessage[]
+decode(char) : string
+encode(string) : char
+validate(string) : int
+getmessage() : Commandlist
Back
+execute(DroneState, string)
Ccw
+execute(DroneState, string)
Command
+execute(DroneState, string)
Cw
+execute(DroneState, string)
Down
+execute(DroneState, string)
Emergency
+execute(DroneState, string)
Flip
+execute(DroneState, string)
Up
+execute(DroneState, string)
Forward
+execute(DroneState, string)
Takeoff
+execute(DroneState, string)
Land
+execute(DroneState, string)
Left
+execute(DroneState, string)
Streamon
+execute(DroneState, string)
Go
+execute(DroneState, string)
Streamoff
+execute(DroneState, string)
Stop
+execute(DroneState, string)
Right
+execute(DroneState, string)
1
1
1
1
1
*


DronesState
CommunicationCenter
UserInterface
Main
Sim
Status
MissionCollection
1: UI()
2: CommunicationCenter
3: CommunicationCenter:sendStatusToDrone
2.1: cc.getStatusFromDrone
1.1: MissionCollection:getMission
1.1.1: MessegeCenter()
1.1.2: UpdateState
1.1.2: Status()
4: cc.sendStatusToDone
1.1.3: DS:updateFlyInfo(Status)


Communication_Center_Flow
Start
Connect to Drone
NO
YES
Wait for Response
Drone put into command mode
No Mission
User input mission 1-3
End
Connection Close


Drone_Flow
START
Connect to command center
NO
YES
Set in outer darkness
Execute commands through socket
NO
END
YES
Connection lost


Verify that there is a connection to the drone
Put Drone in Command Mode
Send information
Receive information
Drone
Communication Center