Drone Mission Planning Software: Design Document

Taylor Trabun

May 6, 2015

1 Introduction

This document provides the general design of the Drone Mission Planning Software by breaking the entire project down into several components. The current components are the physical drone, the communication system, and the graphical user interface for mission planning.

2 Drone Design

The drone design's major requirement that needed to be met was to achieve stable and reliable flight.

By analyzing the drone, it was determined that its center of gravity was not directly centered on the drone, which resulted in drifting during flight. To remedy this issue, a "part holder" was designed to be mounted on the underside of the drone to hold all the motor controls, which moved the center of gravity to the center of the drone.

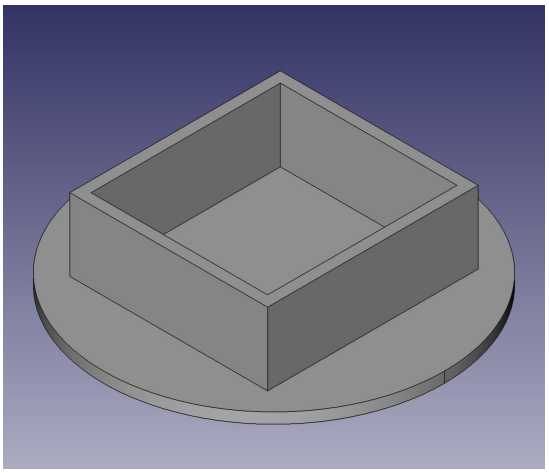

Figure 1: 3D sketch of partbin

3 Communication Design

The communications system needs to follow the following requirements:

- Use of XAPI and XBee hardware

- Define required TUN packets for communication system
  - Manual drone instructions
  - Settings and status
  - ACK
  - Heartbeat
  - Override
  - Mission plan protocol types

The following sections will break the communication system down into its several components and detail their design.

3.1 Communication Overview

Our communications system, as depicted in the graphic below, requires two-way communication between the computer (including the attached Arduino) and the Arduino located on the drone. This system will take an instruction created on the computer, send it over serial to the connected Arduino, pass it to XAPI, XAPI will ship it over XBee to the Arduino on the drone, and a flight control service will execute the instruction.

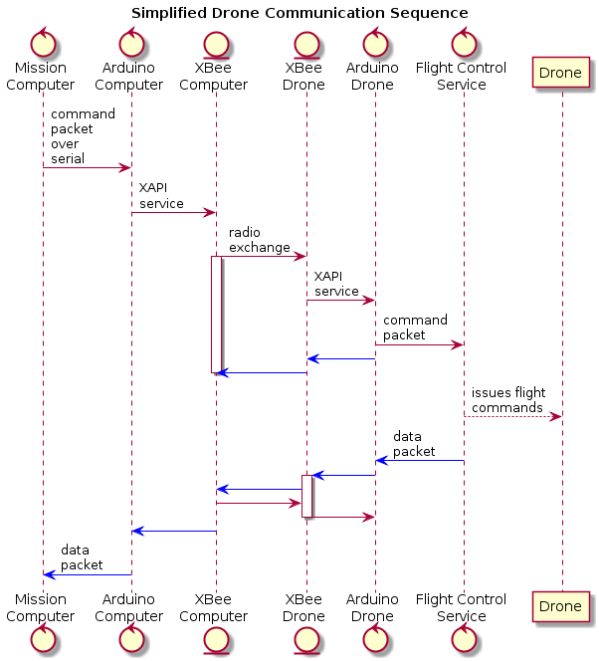

Figure 2: Overview of communications system

3.2 Hardware Components

Our current communications requires the following hardware components:

- Arduino Mega 2560
- XBee modules

- LCD Shields (for development and debugging)
- Serial add-on for Arduino
- A computer running Windows to communicate with base Arduino (Workstation needs to be able to run C# programs)

3.3 XAPI

To satisfy one of our major requirements, we run the XAPI on each Arduino in the communications system.

XAPI is, put simply, a micro-controller service manager that communicates, both internally and externally, using TUN packets. When communicating externally, the XAPI ships the TUN packet using the XBee hardware by embedding the TUN packet in a XBee packet.

Each service available with the API use XAPI to communicate, using XAPI as the core that each service "latches" to. For instance, if a chat service wanted to display a message on a attached LCD screen, the following steps would be carried out:

1. Chat service creates a LOCAL_ LCD TUN packet
2. Chat service passes the newly created packet to the XAPI core
3. XAPI places the packet in its internal packet buffer
4. The LCD service latch queries XAPI for any packets designated for the LCD service
5. LCD service grabs LOCAL_ LCD TUN packet
6. LCD service processes and displays the packet

To satisfy our project's requirements, we need to design a Flight Control service that will be able to handle any instruction packets and translate them to instructions that can be given to the drone's flight computer. In addition to this, there is a requirement for a Mission Plan service that will store and execute flight plan's designed by the user and sent to the drone.

3.4 TUN Packet Types

The following is a table of all the TUN packets types that will be required for this design:

TUN Type Key Internal/External	Payload	Use
0x50/0x51	Altitude	Take off and maintain specific altitude
0x52/0x53	N/A	Land
0x54/0x55	X,Y,Altitude	Go to specific coordinate
0x56/0x57	MoveDirection,Amount,Metric	Carry out specific move (Forward/Back/Left/Right/Rotate)
0x58/0x59	N/A	Start mission plan upload
0x60/0x61	X,Y,Altitude	Mission instruction - go to specific coordinate
0x62/0x63	MoveDirection,Amount,Metric	Mission instruction - carry out specific move (Forward/Back/Left/Right/Rotate)
0x6A/0x6B		Instruction - Land
0x6C/0x6D	Altitude	Instruction - Take off and maintain specific altitude
0x64/0x65	ChecksumValue	End install and check checksum to validate success
0x66/0x67	N/A	Stop current mission plan and activate manual control (maintain altitude)
0x68/0x69	N/A	Stop current mission plan and activate manual control (no longer autonomous)
0x70/0x71	N/A	Acknowledgment (ACK)
0x72/0x73	N/A	Acknowledgment of issue (NACK)
0x74/0x75	InstrumentData	Heartbeat that contains the drone's current state
0x76/0x77	Altitude	Set altitude
0x78/0x79	Speed	Set the never exceed speed
0x80/0x81	Speed	Set never exceed fall speed
0x82/0x83	Throttle	Set throttle
0x90/0x91	On/Off	Set drone armed status
0x92/0x93	Heading	Set drone heading for auto pilot
0x94/0x95	HoldAltitude	Set drone to hold altitude (true/false)
0x96/0x97	HoldHeading	Set drone to hold heading (latitude/longitud

3.5 Mission Plan Upload Protocol

In order for the drone to autonomously carry out a mission plan, the plan must be uploaded from the source computer to the drone's micro-controller. The following is depicts the Mission Plan upload protocol:

Simply put, an installation packet is used to signal the start of a mission plan upload, each instruction is then sent one at a time, and finally an end installation packet with the checksum of the final mission plan file is sent to the drone. After each of these packets are received by the drone, the drone will send an acknowledgement (ACK) packet. It should be noted that the final ACK packet can be a NACK packet if the checksum received is not the same as the one calculated.

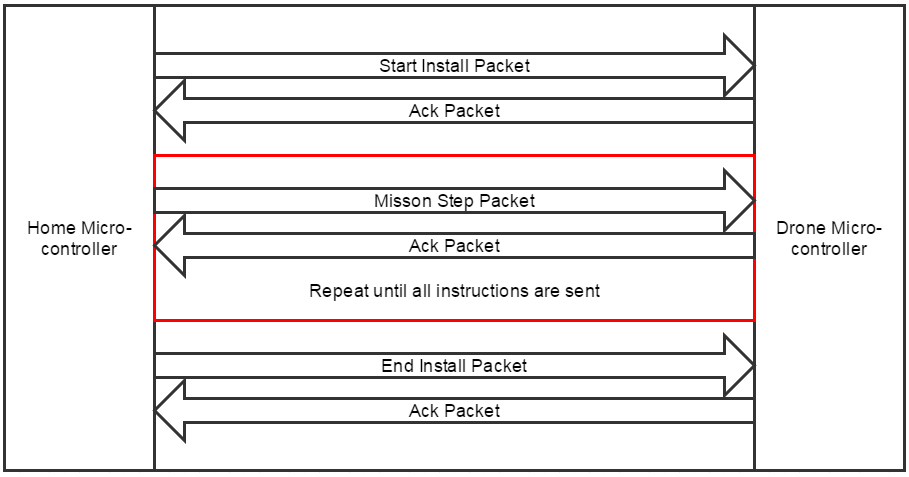

Figure 3: Mission plan upload process

4 Graphical User Interface Design

The graphical user interface must satisfy several requirements:

1. Must be user-friendly
2. Allow 3-dimensional mission planning
3. Allow upload of flight plan to drone
4. Allow manual override

Given these requirements we were able to design a general design for the graphical user interface (GUI), as shown below. This GUI is split into several sections that convey different information, that in some cases can be adjusted.

We have a flight control section (light-blue) that shows the status of the drone components and allows the user to ”zero out” each component or take manual control. The status information section (yellow) shows different readings from the drone’s on-board instruments. The flight planning system (light-red) will be where the user can develop a flight plan to be uploaded to the drone (note that this functionality is still under design and may end up being a separate window that needs to be opened up). The final section is the communications terminal (light-green) that displays all packets sent and received on the Arduino attached to the source computer. This communications terminal will allow the user to see that the drone is still connected and will allow for easy communications debugging.

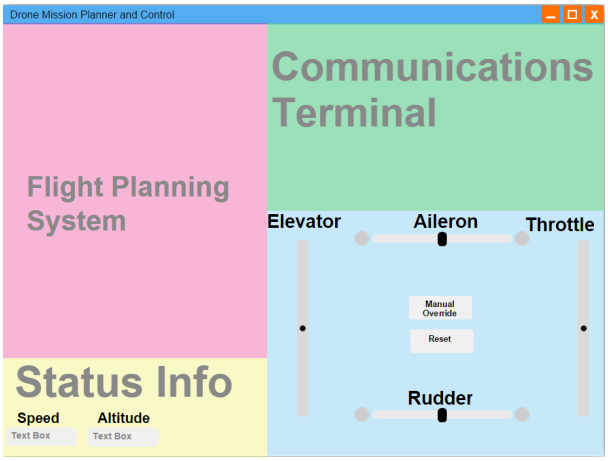

Figure 4: Graphical user interface mock-up design

From this mock-up we began designing the actual GUI using Windows Forms in VisualStudio C. In this version, we decided to get rid of the communication terminal section in favor of a larger area for the map and altitude display. We still need a communication terminal, so we instead used the Serial Terminal that already exists. From the Serial Terminal, a GUI window can be opened when you successfully open a port to the arduino system.

In the upper-left section of our GUI window we have the flight planning panel. Here you can add, subtract, edit and submit several instructions at a time. We have also included two emergency buttons, one to stop the drone in place and hover and another to force the drone to land. These are important in case the drone is about to harm itself or other people.

The bottom left section shows our mission status for current instructions and the mission plan as a whole. Currently this has no function but in the future it will update as the drone transmits data back to the system. Similarly, the map and altitude sections will also update as information is received from the drone.

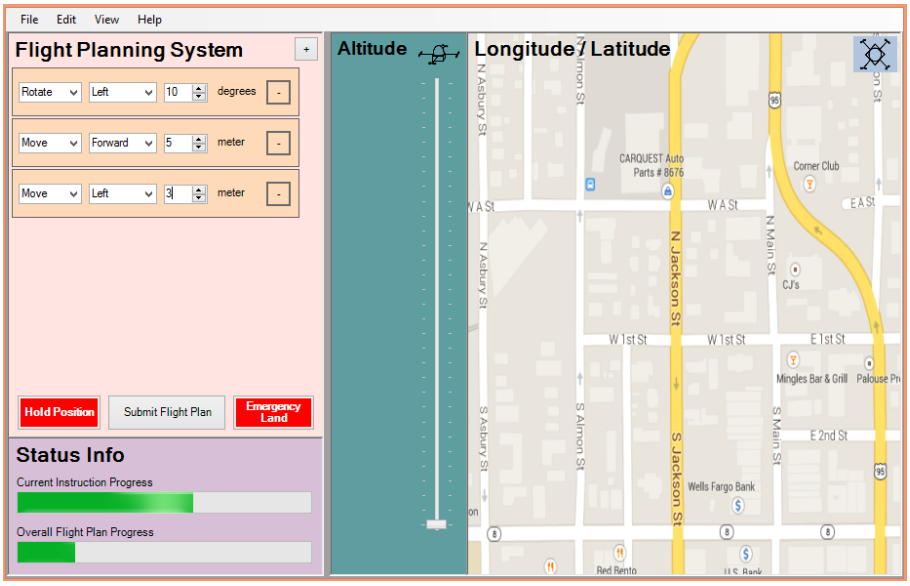

Figure 5: Implemented graphical user interface window

Along the top we have a menu bar. Under "View" we have several options to open other GUI panels. The "Instrument Panel" option will pull up a window that displays status information received from the drone similar to that of an airplane dashboard. The "Manual Controls" option will open a window that allows the user to send manual controls to the drone in real time.

In our manual control panel we have eight arrow buttons. These buttons can be activated via mouse-click or by using the corresponding keyboard shortcut shown on each button. We also have an active indicator in the bottom left. This indicator tells us whether or not the buttons are active, which occurs when the drone is armed. If they are not active then the buttons will do nothing. However, if they are active then keys will be highlighted as they are utilized and send instructions to the drone. Currently when using the keyboard, instructions will be sent to the drone repeatedly as fast as the computer allows. This needs to be updated to only send instructions every so often so we do not flood our connection to the drone.

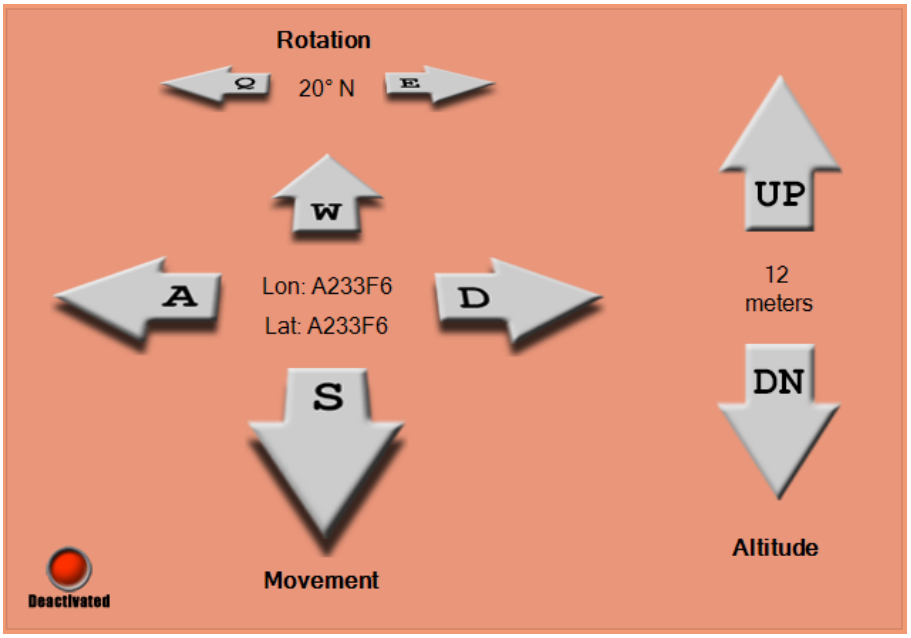

Figure 6: Manual control window for GUI