Data transfer, Control Systems, and Sensors CMPE 185 Final Project

Maxwell Bradley, Kodiak North, and Zane Bradley

Abstract—The purpose of this document is to teach someone the basics of EDIT THIS BRO. Readers will be able to insert tables, figures, mathematical formulas, bibliographies, and references by the end of the tutorial.

Index Terms—Aircraft, Airplane, RC Airplane, UAV, Autonomous, Wildfire, Wildfire Detection, Search and Rescue

CONTENTS

1 PROJECT INTRODUCTION

FOREVER Flight is a persistent aerial monitoring system to detect wildfires in fire-prone areas. It will consist of a plane with a mounted IR camera to detect fires below and a flight controller capable of autopilot, guidable with GPS waypoints sent from a laptop computer on the ground. This project was named

'Forever Flight' for its goal of never having to recharge its batteries, staying aloft and performing fire detection while the sun is up, charging back up from solar energy and our power regenerative mechanisms.

This project was inspired by the recent California wildfires like the Camp Fire, a 2018 fire in Northern California that caused $16.5 billion in damage and claimed 86 lives [1]. The reason that these fires were so deadly is because there was no early warning system that detected the fire before it started ripping through towns like Paradise, California. The team hopes that Forever Flight will prevent fires like this.

2 DATA TRANSFER

2.1 Introduction

An important part of this project is the data transfer from the plane to the ground station. This transfer contains a huge amount of different kinds of data, such as fire detection stats, fire detection processing, GPS coordinates, and more. Getting this kind of information out of the plane can be challenging because of the flight controller's running overhead and varying transmission protocol versions. This section of the paper will talk about the communication of data between the in-flight aircraft and the ground control station read by our product user.

2.2 Materials and Material Background

A multitude of different components make up the data transmission path between the plane itself and the ground station. Each of those parts will now be discussed in detail.

2.2.1 Pixhawk Flight Controller

The Pixhawk flight controller is the main brain of the entire plane. It receives IR information from the IR sensor mounted downwards pointing at the ground through a serial communication port (using the I2C protocol). The Pixhawk is where the majority of the data transfer work occurs in the second version of the data transmission design. It uses an overall system scheduler to decide when to send MAVLink packets to the ground station. Within the function that is scheduled in the overall scheduler, a smaller MAVLink scheduler exists. Within the MAVLink scheduler, one of a large group of packets is selected and sent over whatever telemetry port MAVLink is configured to use.

2.2.2 MAVLink

MAVLink is a protocol commonly used between drones and ground stations. Most autopilots use MAVLink to both send and receive packets to and from the ground station, respectively. Its packet structure is described in the following picture.

Field name	Index (Bytes)	Purpose
Start-of-frame	0	Denotes the start of frame transmission (v1.0: 0xFE)
Payload-length	1	length of payload (n)
Packet sequence	2	Each component counts up their send sequence. Allows for detection of packet loss.
System ID	3	Identification of the SENDING system. Allows to differentiate different systems on the same network.
Component ID	4	Identification of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
Message ID	5	Identification of the message - the id defines what the payload "means" and how it should be correctly decoded.
Payload	6 to (n+6)	The data into the message, depends on the message id.
CRC	(n+7) to (n+8)	Check-sum of the entire packet, excluding the packet start sign (LSB to MSB)

Fig. 1. Max B1

The start frame transmission is the hex character set 0xFE, which tells the ground station that the packet is incoming. The next byte tells the receiver how many bytes in the payload to expect. The next set of bytes are for identifying the system sending the message, the component sending the message, and the packet sequence.

The most important part of this packet is the message id, the 5th byte in the index. This byte signals which MAVLink message the incoming data corresponds to. MAVLink messages are defined in a file called common.xml. Every message coming in indexes to one of the messages in the common.xml file. For instance, several times a second a 'heartbeat' message is sent from the plane to the ground station. This message is declared in the common.xml file as having ID 0. When the ground station receives this message, it reads that xml file, looking for a message definition that matches the incoming ID. When it finds it, it interprets the payload attached to the MAVLink message as the attributes associated with the message definition in the xml file. The autopilot that was chosen for this project uses the MAVLink protocol to communicate with the ground station.

As a result, the team has become very familiar with its function.

2.2.3 ArduPilot

After looking at all the free and open source autopilots out there on the market, Ardupilot, an open source autopilot that runs on the Pixhawk4, was selected. This firmware can be run on a variety of different platforms: there is an ArduSub, an ArduCopter, an ArduTractor, and obviously an ArduPlane. The flight controller for this project uses the ArduPlane version.

This autopilot is written in C++, and has a relatively small code base for the ArduPlane itself. It's most important parts consist of a main plane scheduler, a MAVLink sending module, and a huge header file that includes all the important modules from the libraries and from the plane folder itself.

The majority of the meat for the ArduPlane module is contained in the libraries. The programmers for ArduPilot wisely chose to combine the largesse of the code in libraries which all different vehicle versions that run ArduPilot (ArduPlane, ArduSub, ArduTractor, etc.) use. These libraries contain code for sending MAVLink that can be used between vehicles, hardware abstraction layers, control systems, and plenty more.

2.2.4 Telemetry Module

The team landed on using the 3DR Radio Telemetry Kit. This set of transmitters and receivers use the frequency 916MHz. The flight controller uses this link to send flight data. The ground station sends GPS coordinates for the plane to track to on over this link as well. A picture of this module is below.

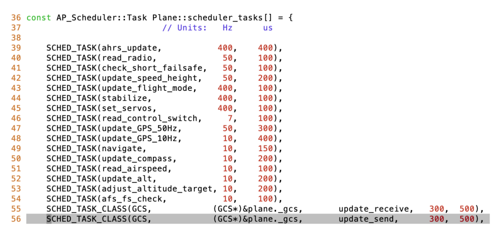
Fig. 2. Max B1

3 Results and Discussion

This section of the paper will be about the overall set up of the system, the design process, and the problems that were encountered on the way.

3.0.1 System Set Up

Our data transmission system starts in the Pixhawk scheduler code in the main file called ArduPlane.cpp. The flight controller is essentially a microcontroller without an operating system, which means that it needs to implement a scheduling system that figures out which process of the hundreds of processes competing for the processor to run. This is what the scheduler looks like in code:



```
const AP_Scheduler::Task Plane::scheduler_tasks[] = {
                            // Units:   Hz      us

    SCHED_TASK(ahrs_update,           400,    400),
    SCHED_TASK(read_radio,             50,    100),
    SCHED_TASK(check_short_failsafe,   50,    100),
    SCHED_TASK(update_speed_height,    50,    200),
    SCHED_TASK(update_flight_mode,    400,    100),
    SCHED_TASK(stabilize,             400,    100),
    SCHED_TASK(set_servos,            400,    100),
    SCHED_TASK(read_control_switch,     7,    100),
    SCHED_TASK(update_GPS_50Hz,        50,    300),
    SCHED_TASK(update_GPS_10Hz,        10,    400),
    SCHED_TASK(navigate,               10,    150),
    SCHED_TASK(update_compass,         10,    200),
    SCHED_TASK(read_airspeed,          10,    100),
    SCHED_TASK(update_alt,             10,    200),
    SCHED_TASK(adjust_altitude_target, 10,    200),
    SCHED_TASK(afs_fs_check,           10,    100),
    SCHED_TASK_CLASS(GCS,            (GCS*)&plane._gcs,       update_receive,   300,  500),
    SCHED_TASK_CLASS(GCS,            (GCS*)&plane._gcs,       update_send,      300,  500),
```

Fig. 3. Max B3

The highlighted line of code schedules the GCS sending module, which calls functions to use the MAVLink link between the ground station and the plane. Diving deeper into the code base in the plane folder, there is the GCS_MAVLink.cpp file that contains the functions to actually go about sending the message. This is where changes started:

As a backup to the above function, ArduPilot also implemented a function that all vehicles under the ArduPilot umbrella use. If none of the IDs passed into the try_send_message function matched, it would fall into the try_send_message that was defined in the libraries of ArduPilot. This was good for overall project structuring, but made finding the ac-

```
bool GCS_MAVLINK_Plane::try_send_message(enum ap_message id)

    switch (id) {

    case MSG_SYS_STATUS:
        CHECK_PAYLOAD_SIZE(SYS_STATUS);
        plane.send_sys_status(chan);
        break;

    case MSG_NAV_CONTROLLER_OUTPUT:
        if (plane.control_mode != MANUAL) {
            CHECK_PAYLOAD_SIZE(NAV_CONTROLLER_OUTPUT);
            plane.send_nav_controller_output(chan);
        }
        break;

    case MSG_SERVO_OUT:
#if HIL_SUPPORT
        if (plane.g.hil_mode == 1) {
            CHECK_PAYLOAD_SIZE(RC_CHANNELS_SCALED);
            plane.send_servo_out(chan);
        }
#endif
        break;
```

Fig. 4. Max B4

tual implementation of MAVLink sending very challenging.

Below is the fallback function at the end of the previous function that links the plane implementation to the common MAVLink packets that all ArduPilot vehicles use. This function is contained in the library files that all ArduPilot vehicles contain.

```

    default:
        return GCS_MAVLINK::try_send_message(id);
    }
    return true;
}
```

Fig. 5. Max B5

By editing code in either the library function or the ArduPlane specific function, it was possible to make changes to the MAVLink messages that were being sent to the ground station. Changes were made to existing MAVLink definitions rather than create new ones because of the difficulty in changing the MAVLink protocol version from version 1.0 to version 2.0 (please view the problems section for more information on this issue). To edit MAVLink message definitions, changes needed to be made in the common.xml or ardupilotmega.xml files.

3.0.2 Design Process

The design process was begun by trying to prevent the flight controller from having to handle any of the data transmission at all. Instead, the original plan was to have a Teensy microcontroller serve as the gateway between the ground station on the ground and the plane up in the sky. The first thing that was done was to set up a UART link between the Teensy and the flight controller, sending over information that would eventually be relayed to the ground station. The overall structure looked like this:

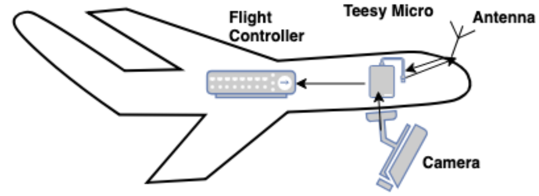


Fig. 6. Max B6

However, this set up became less than optimal when the incredible wealth of the already existing ground control software became apparent. It was discovered that the ground station software, Mission Planner, could be used to send GPS waypoints to the plane which it would then track to. The interface between the flight controller and the ground control station would only work if it could receive bytes from the ground station, and only allowing the Teensy access to the ground station would prevent this huge benefit from being realized. It didn't make sense to spend a lot of time designing a system where the Teensy read in fire detection data forwarded them to the flight controller. In addition, the small and underpowered Teesny with a processor clocked at 48 MHz could not hope to keep up with a flight controller clocked at over 3 times that. The Teensy was then removed from the picture, replaced with a more overloaded flight controller. A diagram of the current set up is below.

After the flight controller became the center of data processing and transmission, the design process began anew. The first important thing to do was to get a good understanding all of the underlying code for data transmission in the ArduPilot framework. After tracking down the control flow through the different files which eventually ended in in the library file that all ArduPilot vehicles used, the next step was to

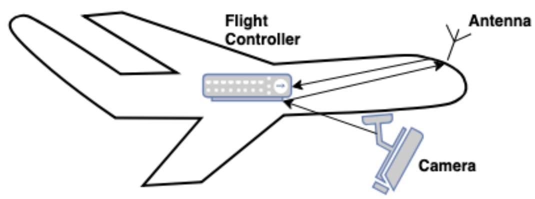


Fig. 7. Max B7

add a particular message to the common.xml file which contained the MAVLink message definitions.

Due to a parameter problem that will be covered in-depth in the problems part of this discussion, it was necessary to overwrite one of the existing MAVLink definitions. The MAV_CMD_NAV_LAND with ID 21 message was selected because the plane should never be allowed to land automatically This message was overwritten with a custom packet that contained fire detection data. This packet would only be sent when the sensing apparatus actually detected a fire, even though the message would be scheduled for sending at least once a second. The simple control flow of this setup is as follows.

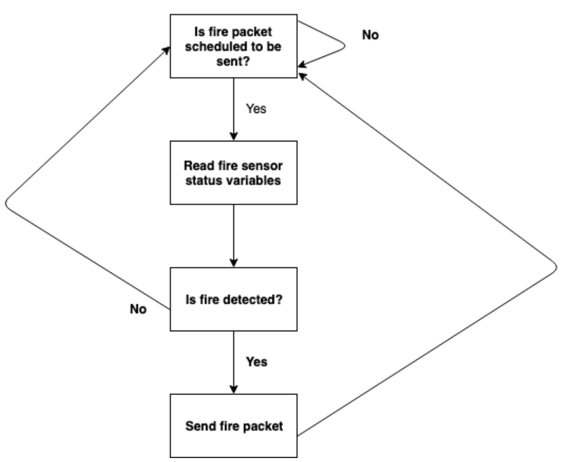


Fig. 8. Max B8

For the ground station that would receive messages, a Python script that used the Python package PyMavlink's submodule mavuti was used. This allowed a reading of the serial port and a parsing/translation of the incoming MAVLink message into human readable data. The associated data was then used to make plots, such as the following that plots latitude and longitude from the incoming GPS data packets.

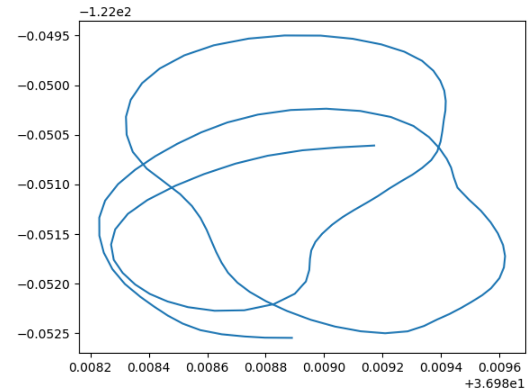


Fig. 9. Plot of the aircraft's maiden flight using GPS coordinates that were logged in real time via the telemetry link.

This allows for the collection of real time flight data which can be very different from the data read from a wind tunnel in the lab.

3.0.3 *Design Problems*

The biggest and most enveloping design problem that was encountered during this project was changing the MAVLink protocol version. Please note the MAVLink byte structure referred to in the next few sentences is the first diagram of this section of the report. The 5th byte of the packet designates the ID number that corresponds to the MAVLink message. This limits the total number of incoming messages to 256. The common.xml file contains message definitions up to 256. Due to the limited number of bits (8) that correspond to the incoming ID, there are only 256 possible messages that can be sent. This is why overwriting the already defined messages in the common.xml file is needed. Creating new messages with an ID value greater than 255 would not be able to be sent by the current MAVLink protocol. Originally, a custom packet with an ID value of 11065 was created and the sending function for that specific packet was called. However, nothing was sent, not even garbage. This along with the packet header of 0xFE made it very clear that the first version of the MAVLink protocol was being used rather than the updated and more flexible MAVLink version 2. Overwriting the message was a necessary evil to send custom packets unfortunately.

The changing of this protocol is badly documented on the ArduPilot website. Eventually, a manual change using the MAVProxy python library was attempted (specifically changing the protocol version parameter of the MAVLink version), but MAVLink 1 packets continued to be sent. In the interest of time, it was decided that simply overwriting the useless packets defined for MAVLink 1 was the best course of action.

3.1 Data Transfer Conclusion

Data transmission software is essential to pulling real time data that closely reflects and explains the events that occured during flight. Even though the open source autopilot is incredibly difficult to parse through and overall not very well written by industry computer science standards, it can be manipulated to add custom MAVLink functionality.

An added problem of working with this flight controller is the time and processing constraints that adding a custom bit of functionality to an already extremely overtasked system come with. However, by keeping subroutines short, it was possible to add a good amount of data collecting and transmission software to an already built system.

4 Control System

4.1 Introduction

The control systems section will explore the mysterious art of controlling planes without a pilot on-board. The overall plan is to control the aircraft in an optimal way to detect wildfires, but for now, we are just focused on control in general.

4.1.1 *Treating the Aircraft as a Dubins Vehicle*

All information in this section was found in [5], chapters 9 and 10. These chapters focus mainly on creating a model for guidance and path following. Using coordinate frame matrices to resemble flight, we can model aircraft dynamics along with outside forces with the following equation (eq 9.9 in [5]):

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{h} \end{pmatrix} = V_a \begin{pmatrix} cos(\psi) \\ sin(\psi) \\ 0 \end{pmatrix} + \begin{pmatrix} w_n \\ w_e \\ 0 \end{pmatrix}$$

It essentially says that the angular roll rate $\dot{p}$, and the rate of change in altitude $\dot{h}$ are equivalent to the airspeed V_a times the cosine and sine of the yaw angle ψ plus the wind direction w. Note that aerodynamic forces acting on the aircraft body are removed from this equation to greatly simplify it. The equation assumes the aircraft maintains a steady altitude to model it like a dubins vehicle with a forward velocity and a turning rate. The airplane is then controlled by increasing or decreasing the velocity and changing the turning rate. Below in Figure 10 is a plot of a dubins vehicle moving in a figure-eight pattern: This idea is particularly

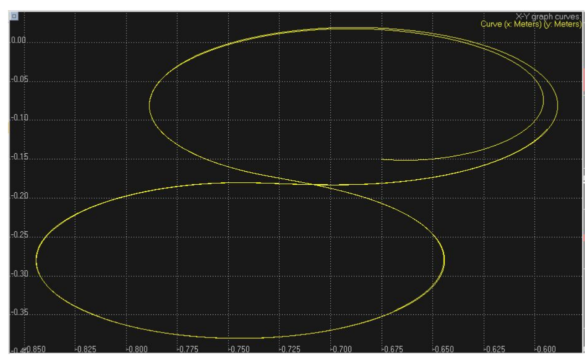


Fig. 10. Dubins vehicle programmed to move in a figure-eight pattern by holding a constant velocity and changing the sign of its turning rate at a specific time interval. An airplane at a constant forward velocity and altitude can be modeled in a similar fashion. Note this picture was taken from a project the author completed for CMPE 141.

useful for Unmanned Aerial Vehicles (UAVs), which do not perform aerobatic maneuvers and usually maintain a steady altitude to scan an area. Engineers can basically pretend their UAV is a dubins vehicle once it reaches its cruising altitude! We can then relate the turning rate along the yaw axis $\dot{\psi}$ to gravity, velocity, and the roll angle ϕ with the following equation (eq 9.14 in [5]):

$$\dot{\psi} = \frac{g}{V_a} tan(\phi)$$

It is important because it gives us a direct relation from roll angle to yaw rate. This equation is similar to the steering wheel on a car - it is rolled left and right to make the car yaw left and right. In this case, the aircraft itself is the steering wheel rolling left and right to yaw back and forth. We now have everything we need to control a UAV like a dubins vehicle using the roll angle to modify turning rate, and the motor to adjust its speed.

4.1.2 Aircraft Ground Control Stations

All information in this section was found in [9], chapter 8. A ground control station is something that can apply the dubins vehicle model as the UAV is in the air. As an operator plans the flight, the ground control software will calculate the turning rate (i.e. roll angle) required to follow the path. If the turning rate is unachievable, the software must notify the operator, or perform longer, wider maneuvers to correct for the error and follow the path as close as possible. This is one of many reasons why ground station are important tools for a UAVs. Their other purposes are to allow an operator to track/plan the mission, observe flight status, and pilot the aircraft. "Depending upon range and type of mission (complexity of the UAV system), smaller UAVs are controlled via visual contact (manual real-time control), while the larger ones are equipped with a communication system (engage stored on-board flight plans)" (Sadraey, page 141). Some UAVs are not fully autonomous, so an operator must fly it using a laptop with ground control software. UAVs also have various sensors on them like a weapon position sensor, altimeter, airspeed sensor, intertial measurement unit (IMU), and many more. If the ground control station operator notices anything is acting funny, he can take full control and act accordingly. Normally, there are two operators to a ground control station; the vehicle flight operator and the mission payload operator. The flight operator is in charge of keeping the machine in the air, while the payload operator interfaces with other aircraft systems. This could be aiming and firing guns, taking weather data, or gathering recon information for teams on the ground. These

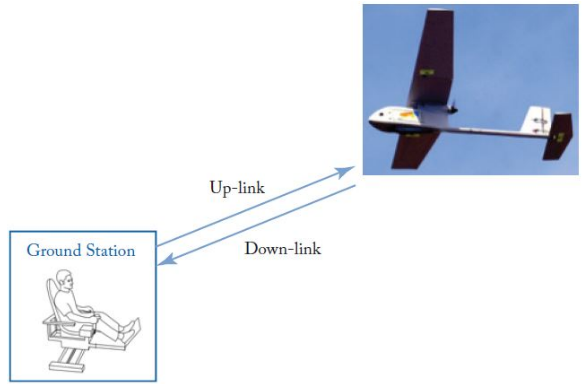


Fig. 11. Diagram displaying and operator of a ground control station to pilot an aircraft. Most stations do not require the operator to sit in a cockpit-like seat, but the exaggeration is useful for a novice UAV enthusiast. This image was found in [9], page 142, Figure 8.1.

operators play a crucial role because although the plane can fly mostly, or fully autonomously, "most UAV losses are attributed to operator errors" (Sadraey, page 143). The operators must be comfortable when working with these expensive devices or else their state of mind will fatigue, and the chances of making a mistake increase. It is important to keep comfort in mind when desigining a ground control station.

4.1.3 Glider Soaring via Reinforcement Learning in the Field

All information in this section was found in [7]. One way to extend the flight time of a small, unmanned aircraft is to utilize thermals and updrafts similar to the way birds do. Physicists at UC San Diego figured out how to do exactly that by using reinforcement learning - a type of machine learning. Reinforcement learning is essentially a way for a system to learn based on trial and error; if it preforms an action and things go well, it remembers that as a good situation, while on the other hand, poor situations from bad maneuvers are kept in the "don't do again" list. The physicists started by simulating gliders in turbulent wind flow to find important parameters that improve navigation through a thermal. They then moved on to testing a real plane with a 2 meter wingspan. The plane was equipped with a flight controller running modified ArduPlane code to pilot the aircraft through updrafts. Researchers soon discovered

Bird and glider in tandem flight. Photo montage courtesy of Phil Richardson, © Woods Hole Oceanographic Institution

Fig. 12. Here is one of the gliders used in the testing process flying in tandem with a seagull. The two both have the same attitude to achieve the most lift out of the thermal or updraft they are flying through. This image was found in [6].

that they needed to write algorithms to estimate “environmental cues” to maneuver the glider in such a way to gain the most lift out of the environment. For example, when a pocket of greater lift is at a diagonal to the aircraft’s forward heading, banking in a proper manner will generate more lift. This project intends to apply these properties to the Forever Flight aircraft to achieve a longer flight time when searching for wildfires.

4.2 Methods

4.2.1 Choice of Aircraft Frame

For our UAV, the team decided to go with a glider style aircraft over a delta-wing style. We originally wanted to use a delta-wing because they are quick to build and durable. However, we learned that these characteristics should not be our main focus because they define what someone would want if they were going to crash a lot! Since we were creating a plane intended to fly forever, we realized that we should choose an efficient-flying plane that can use less power to remain in the air. We wanted a plane that can utilize updrafts or thermals to recharge its battery when running low, or to simply gain altitude free of charge. Therefore, a glider was an obvious choice.

4.2.2 Tasks to Connect the Electronics

The first step in connecting the control system is to verify that all electronics are working properly before inserting them into the plane. This involves testing the servos and motor in a simple RC configuration without any autopilot, then connecting them to the autopilot and ensuring manual control can be taken with the flip of a switch, and finally integrating the wireless data transmission system.

4.2.3 Simple RC Configuration

Before motors and servos can be tested, the Taranis X9D RC transmitter must be bound to the FrSky X8R receiver so the two can communicate.

1) With the X9D off, hold down the F/S button on the receiver and power it up. Then let go of the F/S button. The light on the receiver should be solid RED indicating it is in bind mode.
2) Turn the X9D on and select the model you want to bind the receiver to.
3) Short press the menu button, and then press the page button to navigate to page 2 of the settings.
4) Scroll down with the ‘+’ or ‘-’ buttons until you see “Channel Range” and “Receiver No.” settings.
5) Under “Receiver No.”, highlight [bind] and press “ENT” to confirm. Leave the “Channel Range” as 1-8 with telemetry ON. Press “ENT” again.
6) The receiver’s LED will flash GREEN and RED, and the transmitter will beep, indicating that they are binding.
7) After a couple beeps, press “EXIT” on the transmitter and power off the receiver.
8) Power the receiver back on. The LED should turn solid GREEN after a few seconds indicating that it is connected to the transmitter. The two are now paired together under the selected model forever unless the receiver is rebound to a different model.

With that done, servos and motors can be controlled by plugging them into the outputs on the X8R. Power everything off and plug the ESC and four servos into the receiver (see Table 1). Note that the two aileron servos are connected to only one channel using a servo Y-splitter. This causes the servos to behave opposite of one another as they would on any aircraft. The motor connects to the ESC via three wires with bullet connectors (see Figure 13).

TABLE 1
FrSky X8R Channels to Aircraft Control Surfaces

X8R Channel	Aircraft Control Surface
1	Ailerons
2	Elevator
3	Motor
4	Rudder

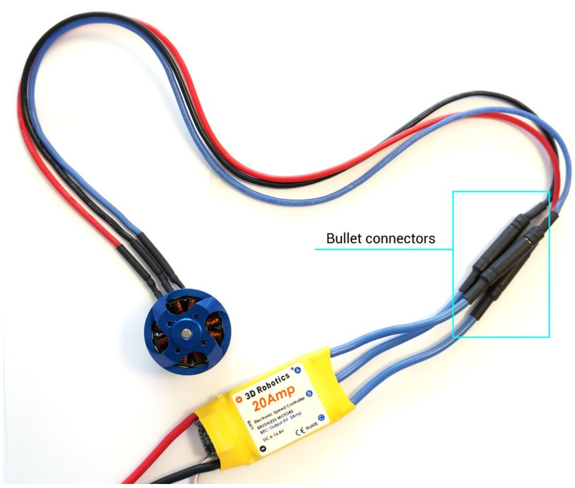


Fig. 13. Brushless motor connected to ESC via bullet connectors found it [8]

Now that everything is connected, follow these steps to center servos and test servo/motor functionality.

1) Ensure the propellers is OFF of the motor shaft, and the transmitter is powered off.
2) Turn on the transmitter.
    a) Ensure all switches are in default position and throttle position is at its minimum (default settings on the X9D will cause it to yell at you if these constraints are not met).
3) Plug the battery into the ESC.
    a) The servos will center right when they receive power for the first time. Make sure that the trim on all channels is centered.
4) Wait for the motor startup sound sequence to play.
5) Move sticks on the transmitter and observe control behavior.
    a) Note that servo direction can be flipped in the transmitter settings, so their directions are not too important at this point.
    b) If the motor spins in the wrong direction when the throttle is applied, switch any two of the three wires that connect it to the ESC.

In this stage, it is also wise to manually calibrate the ESC so it recognizes this specific transmitter's maximum and minimum PWM throttle outputs. Follow the steps below.

1) Ensure the propellers is OFF of the motor shaft, and the transmitter is powered off.
2) Place the throttle stick to its maximum position.
3) Plug the battery into the ESC.
    a) It will play a special sequence to indicate that maximum throttle has been detected.
4) Place the throttle stick to its minimum position within 2 seconds.
    a) The ESC will beep and play its normal startup sequence.
    b) If the normal startup sequence does not play and instead the ESC beeps abnormally, repeat this process and be sure to move the throttle stick to the minimum position within 2 seconds!
5) Calibration is now complete.

With these steps complete, we can move to connecting the motors, servos, and receiver to the flight controller.

4.2.4 Connect Motors, Servos, and Receiver to Pixhawk

Refer to Figure 14 for the wiring diagram of all the electronics connected to the flight controller. Note that a special cable is required to connect the signal port of the LiPo Cell Voltage Monitor to the TELEM1 port on the Pixhawk. This can be purchased from Craft and Theory, or one can be made using an RS232 converter. We decided to buy the $15 connector. Also observe that although not intuitive, the Pixhawk's IO PWM Out is connected to the PMB's FMU PWM In. This is to map the RC outputs to the FMU header pins rather than to the M1-8 solder pads. This is not described very well in the Pixhawk's quickstart guide!

With all components connected, the first step is to run through Mission Planner's setup wizard. This involves calibrating sensors on the Pixhawk, flashing it with the latest firmware, choosing the battery sensor module, and configuring the Taranis X9D so Mission Planner recognizes its PWM outputs. If incorrect PWM values are observed, or they do not change with stick movements, navigate to "Initial Setup" → "Servo Output". Then change each channel function from throttle, ailerons, elevator, and rudder to their respective RCINx channels as defined in Table 1.

The next step is to configure the flight modes in mission planner so switches on the X9D can cycle through different modes. If the preconfigured Flight Deck model is used (described in the next section), RC channel 5 will be mapped to switches SD and SC. Play with these to set the preferred flight mode selections. With everything powered on, when one of those switches is flicked, one can see the flight mode change from gray to green in Mission Planner. Now when in MANUAL or STABILIZE mode, and the flight controller armed (1 second press on the GPS safety switch), the X9D should have full control of the motor and servos!

4.2.5 Flash Taranis X9D with FlightDeck Software

The FlighDeck software from Craft and Theory can serve as a backup ground station directly

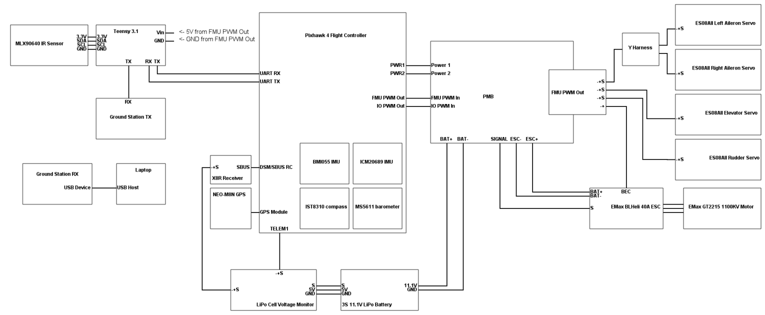


Fig. 14. Diagram of all electronics in the aircraft.

on the transmitter. It provides everything necessary to fly the aircraft such as heading, altitude, battery voltage, and attitude. Follow the steps below to flash FlighDeck onto the transmitter after purchasing it from Craft and Theory. Note that a 40% discount is given on FlightDeck when it is purchased together with the TELEM cable described in the section above.

1) Download OpenTx Companion here.
   a) Run the companion executable.
   b) Select radio type 'FrSky Taranis X9D+' (or a different radio that is being used).
   c) Check the 'lua' box and ensure the 'sql5font' box is unchecked.
2) Download latest firmware version.
   a) We used this one here.
3) Put Taranis in bootloader mode by holding both horizontal trims inward and powering on.
4) Connect Taranis to computer with mini-USB cable.
5) Flash firmware to radio using OpenTx Companion's handy user interface.
6) Navigate to SD card drive on the computer that the Taranis is plugged into. It is drive E:\on my laptop. It contains folder such ad LOGS, MODELS, and SOUNDS.
7) Copy all of these files and back them up in a safe location.
8) Download the latest SD card version here.

9) Extract the contents of the .zip folder into the root directory of the SD card (in my case, drive E:\).
10) Download FlightDeck.zip from the email confirmation when it was bought or from the Craft and Theory account created for the purchase.
11) Extract the contents of the "SDcard" folder directly into the root directory of the Taranis SD card (in my case, drive E:\).
    a) Make sure to merge the contents of these folders and replace/overwrite any file already on the SD card.
12) Manually backup all models currently on the X9D to its SD card before proceeding to avoid any potential frustration.
    a) Long press 'Enter' over each model and select 'Backup'.
13) Add the preconfigured FlightDeck model to the transmitter using OpenTx Companion.
    a) Open the X9D+.otx file that came with FlightDeck in OpenTx Companion.
    b) Press the "Write Models and Settings to Radio" button.
14) Unplug the USB cable from the Taranis, select 'Exit', and confirm. The radio will now boot up normally.
15) Discover new sensors!
    a) In Mission Planner, navigate to the full parameter list and set "SERIAL1" = 10 for FrSky SPORT Passthrough, OpenTx.
    b) Disconnect from Mission Planner and unplug the micro USB cable.
    c) Power the electronics with a battery and wait for the Pixhawk startup sequence to play.
    d) Navigate to the Telemetry screen on the X9D and select the "discover new sensors" button.
    e) Notice that GPS and CELS (battery lowest cell voltage), amongst others are now available.

4.3 Mounting the Electronics in the Plane

Now with the electronics tried and tested, it is time for the moment we have all been waiting for - to stuff everything inside of the plane. When doing this, it is important to mount the flight controller and GPS as close to the aircraft's center of gravity as possible. It is also important to keep other data transmission antennas (like the GPS and Telemetry link antennas) far away from antoher to reduce noise generated between the two. Keep the plane's center of gravity in mind when finding a nice place for heavier equipment. It must balance on two points directly behind the leading edge of the wing, and preferrably be slightly nose heavy. Failure to do this will cause the plane to fly improperly and will likely ensue a crash.

After mounting everything, power up the transmitter, connect the battery to the flight controller and test that all control surfaces move in the correct direction based on manual stick input. Then verify that the plane balances properly over its center of gravity. If it does not, shift something heavy (the battery works well) to correct this. Once all of these checks pass, the plane is ready for manual flight!

4.4 Conclusion

We now have a control system impelemented and ready to be simulated. We do not want to immediately jump into autonomous mode becuase if it does not work as we expect, the plane might crash, or fly far away and never return. For this reason, we are designing a Software in the Loop (SITL) simulation to detect any bugs in the autonomous code. One has been found already where if the aircraft is launched in the opposite direction from its first GPS waypoint, it will never turn around! The ArduPlane code tries to fly the plane around the entire world to hit the waypoint, but since our plane does not have enough battery life to make the journey, it would crash. Future work includes more simulation, and testing the autopilot in real time once we feel we have found all the bugs and we know how to avoid them.

5 PROJECT CONCLUSION

LaTeX is a very useful tool for generating professional PDF documents. Its equation editor makes it simple to add lengthy mathematical equations in a short amount of time, and allows for images to be inserted and BRU NEEDS AN EDIT. There is a learning curve to the language, but hopefully by now readers are familiar with the basic commands and terminology. Thanks to the World Wide Web, tutorials and forums are available in seconds to help learn any advanced LaTeX commands. Thanks for reading!

APPENDIX

A large WHAT EVEN IS AN APPENDIX list of LaTeXmath symbols can be found in [**?**].

ACKNOWLEDGEMENTS

Kodiak would like to thank the authors of Small Unmanned Aircraft, and Unmanned Aircraft Design for providing such detailed descriptions on UAV design and control. He would also like to thank all researchers that are apart of Glider soaring via reinforcement learning in the field ([7]) for conducting such excellent work that will contribute to his capstone project.

REFERENCES

[1] "ArduPlane Home," ArduPilot. [Online]. Available: http://ardupilot.org/plane/index.html. [Accessed: 06-Mar-2019].

[2] "MAVLink," Wikipedia, 08-Nov-2018. [Online]. Available: https://en.wikipedia.org/wiki/MAVLink#Packet_Structure. [Accessed: 05-Mar-2019].

[3] "Pixhawk 4," Basic Concepts PX4 User Guide. [Online]. Available: https://docs.px4.io/en/flight_controller/pixhawk4.html. [Accessed: 06-Mar-2019].

[4] "Teensy v3.1 - 32 bit arduino-compatible microcontroller board," Velleman Spotlight. [Online]. Available: https://www.velleman.eu/products/view?id=420178&country=be&lang=en. [Accessed: 06-Mar-2019].

[5] Beard, R. and McLain, T. (2012). Small unmanned aircraft. Princeton, N.J: Princeton University Press.

[6] C. Dillon, "Physicists Train Robotic Gliders to Soar like Birds", Ucsdnews.ucsd.edu, 2018. [Online]. Available: https://ucsdnews.ucsd.edu/pressrelease/physicists-train-robotic-gliders-to-soar-like-birds. [Accessed: 20-Feb-2019].

[7] G. Reddy, J. Wong-Ng, A. Celani, T. Sejnowski and M. Vergassola, "Glider soaring via reinforcement learning in the field", Nature, vol. 562, no. 7726, 2018. Available: 10.1038/s41586-018-0533-0.

[8] "Motor to ESC," Instructables. [Online]. Available: https://cdn.instructables.com/FDE/6G33/HLPSAYKP/FDE6G33HLP [Accessed: 07-March-2018].

[9] Sadraey, M. (n.d.). Unmanned aircraft design.

Kodiak North is currently a senior at the University of California - Santa Cruz. He is majoring in Robotics Engineering, and looking for a career in the new drone industry once he graduates. On his free time, Kodiak enjoys surfing, repairing his truck, and flying his RC race quadcopter.