Telemetry, Control Systems, and Sensors on an Autonomous Aircraft

Maxwell Bradley, Kodiak North, and Zane Bradley

Abstract—This paper describes the subsystems onboard an autonomous aircraft that detects wildfires. The first subsystem is the data transfer system, between the aircraft and the ground. The second is the control system, which navigates the plane between predefined way points, and keeps it on course in the event of wind or thermals. The third system is the fire detection system, which scans incoming images, determines if there is a fire present, and communicates with the data transfer system in the event of a fire.

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. I will focus this section of the paper on the transfer of data between the in-flight aircraft and the ground control station.

2.2 Materials and Material Background

Many different components make up the data transmission path between the plane itself and the ground station. I will now discuss each of those components 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 downward facing IR sensor through a serial communication port (using the I2C protocol). The Pixhawk is where the majority of the data transfer work occurs in the second and third versions of the data transmission system design. It uses an overall system scheduler to decide when to call the MAVLink packet sending function. Within that MAVLink sending function, 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. Its packet structure is described in the following table.

The start frame transmission is the hex character set 0xFE, which tells the ground station which MAVLink protocol version is being used. 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.

MAVLink message definitions exist in the common.xml file, where the message title and its payloads are all declared.

The most important part of this packet is the message id, the 5th byte in the index. This byte signals which particular MAVLink message the incoming data corresponds to. Every message coming in indexes to one of the messages in the

Field Name	Byte Index	Purpose
start of frame	0	Denotes the start of frame transmission
payload length	1	length of the payload, in bytes. Let this value be called 'N'.
packet sequence	2	counter which increments with each message, used to detect packet loss
system ID	3	Identifier for the message sender, so that the receiver can differentiate between multiple senders
component ID	4	Additional identifier for the message sender
Message ID	5	Identifies the message, so that the receiver knows how to parse the payload.
Payload	6 to N+6	The message-dependent data. For example, an attitude message will include pitch, roll, and yaw in its payload
CRC	N+7 to N+8	A 2 byte Cyclic Redundancy Check of the entire packet, to catch transmission errors

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. During my time as the data transmission engineer, I have become very familar with this protocol.

2.2.3 ArduPilot

After looking at all the free and open source autopilots on the market, the team selected ArduPilot. 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. Its 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 substance for the ArduPlane module is contained in the libraries. These libraries contain code which all different vehicle versions that run ArduPilot (ArduPlane, ArduSub, ArduTractor, etc.) use. More importantly, they provide functionality for sending MAVLink packets 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 go to on over this link as well. A picture of this module is below.

Fig. 1. 916MHz telemetry module

2.3 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.

2.3.1 System Set Up

Our data transmission system starts in the Pixhawk scheduler code within 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 out of hundreds for the processor to run. This is what the scheduler looks like in code:

```
36 const AP_Scheduler::Task Plane::scheduler_tasks[] = {
37                         // Units:   Hz      us
38
39     SCHED_TASK(ahrs_update,          400,    400),
40     SCHED_TASK(read_radio,            50,    100),
41     SCHED_TASK(check_short_failsafe,  50,    100),
42     SCHED_TASK(update_speed_height,   50,    200),
43     SCHED_TASK(update_flight_mode,   400,    100),
44     SCHED_TASK(stabilize,            400,    100),
45     SCHED_TASK(set_servos,           400,    100),
46     SCHED_TASK(read_control_switch,    7,    100),
47     SCHED_TASK(update_GPS_50Hz,       50,    300),
48     SCHED_TASK(update_GPS_10Hz,       10,    400),
49     SCHED_TASK(navigate,              10,    150),
50     SCHED_TASK(update_compass,        10,    200),
51     SCHED_TASK(read_airspeed,         10,    100),
52     SCHED_TASK(update_alt,            10,    200),
53     SCHED_TASK(adjust_altitude_target, 10,   200),
54     SCHED_TASK(afs_fs_check,          10,    100),
55     SCHED_TASK_CLASS(GCS,           (GCS*)&plane._gcs,       update_receive,   300,  500),
56     SCHED_TASK_CLASS(GCS,           (GCS*)&plane._gcs,       update_send,      300,  500),
```

Fig. 2. Ardupilot scheduler

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:

```
454 bool GCS_MAVLINK_Plane::try_send_message(enum ap_message id)
455
456     switch (id) {
457
458     case MSG_SYS_STATUS:
459         CHECK_PAYLOAD_SIZE(SYS_STATUS);
460         plane.send_sys_status(chan);
461         break;
462
463     case MSG_NAV_CONTROLLER_OUTPUT:
464         if (plane.control_mode != MANUAL) {
465             CHECK_PAYLOAD_SIZE(NAV_CONTROLLER_OUTPUT);
466             plane.send_nav_controller_output(chan);
467         }
468         break;
469
470     case MSG_SERVO_OUT:
471 #if HIL_SUPPORT
472         if (plane.g.hil_mode == 1) {
473             CHECK_PAYLOAD_SIZE(RC_CHANNELS_SCALED);
474             plane.send_servo_out(chan);
475         }
476 #endif
477         break;
```

Fig. 3. Vehicle specific MAVLink sending function

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 the overall project structuring, but made figuring out the actual implementation of MAVLink transmission very challenging.

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

```
528
529     default:
530         return GCS_MAVLINK::try_send_message(id);
531     }
532     return true;
533 }
```

Fig. 4. Sending function common to all Ardupilot vehicles

By editing code in either the library try_send_message function or the ArduPlane specific try_send_message function, I as able to make changes to the MAVLink messages that were being sent to the ground station. I changed 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). I started by selected messages to get overwrite in the common.xml file.

```
2869     <message id="0" name="HEARTBEAT">
2870       <description>The heartbeat message shows that a system or component is present and responding.
     The type and autopilot fields (along with the message component id), allow the receiving system to
     treat further messages from this system appropriately (e.g. by laying out the user interface based o
     n the autopilot).</description>
2871       <field type="uint8_t" name="type" enum="MAV_TYPE">Type of the system (quadrotor, helicopter, e
     tc.). Components use the same type as their associated system.</field>
2872       <field type="uint8_t" name="autopilot" enum="MAV_AUTOPILOT">Autopilot type / class.</field>
2873       <field type="uint8_t" name="base_mode" enum="MAV_MODE_FLAG" display="bitmask">System mode bitm
     ap.</field>
2874       <field type="uint32_t" name="custom_mode">A bitfield for use for autopilot-specific flags</fie
     ld>
2875       <field type="uint8_t" name="system_status" enum="MAV_STATE">System status flag.</field>
2876       <field type="uint8_t_mavlink_version" name="mavlink_version">MAVLink version, not writable by
     user, gets added by protocol because of magic data type: uint8_t_mavlink_version</field>
2877     </message>
```

Fig. 5. Sending function common to all Ardupilot vehicles

2.3.2 *Design Process*

I began the design process by trying to prevent the flight controller from having to handle any of the data transmission at all. Instead, my 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 I did 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:

However, this original set up became less than optimal when the incredible wealth of

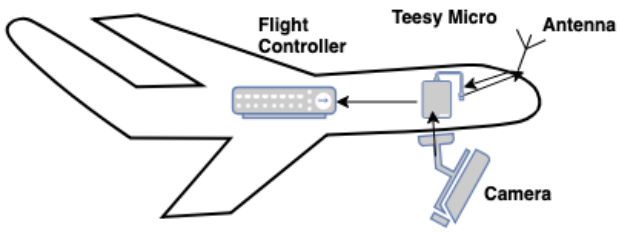

Fig. 6. Diagram of dual microcontroller setup

the already existing ground control software became apparent. I discovered that the ground control software, Mission Planner, could be used to send GPS waypoints to the plane which the plane would then track to without any extra code. The interface between the flight controller and the ground control station would only work if it could receive bytes from the ground station. This required the flight controller to have a direct link to the ground station which it was not configured to do in the first design.

It didn't make sense to spend a lot of time designing a system where the Teensy acted as a go-between for the flight controller and ground station. 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, letting the flight controller bear a greater computational load. A diagram of the evolved design is below.

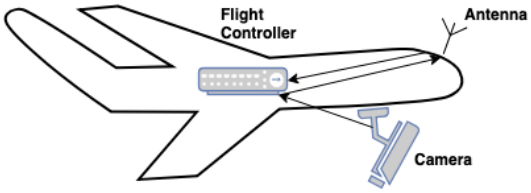

Fig. 7. Updated design without Teensy microcontroller

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

Due to a parameter problem that I will cover in-depth in the problems part of this discussion, I found it necessary to overwrite one of the existing MAVLink definitions. The HIL_STATE message with ID 90 was selected because it had been replaced by another message and was no longer in use. This message was overwritten with a custom packet that contained fire detection data. This packet would only be sent when the sensing apparatus detected a fire. The simple control flow of this setup is as follows.

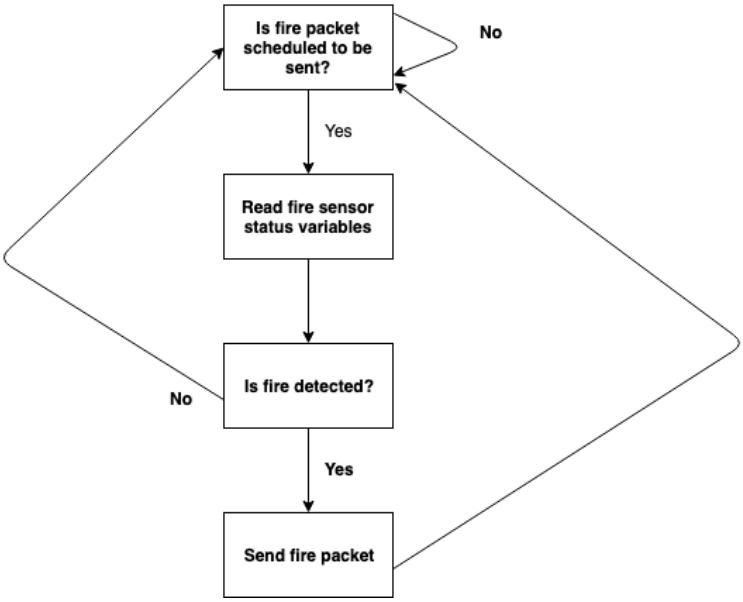

Fig. 8. Control flow of fire packet transmission algorithm

I wrote a Python script on the ground station side that used the Python package PyMavlink. This script read the serial port incoming messages would enter on and translate the incoming data into a human readable form. The associated data was then used to make plots, such as the following that plots latitude and longitude from the incoming GPS data packets.

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.

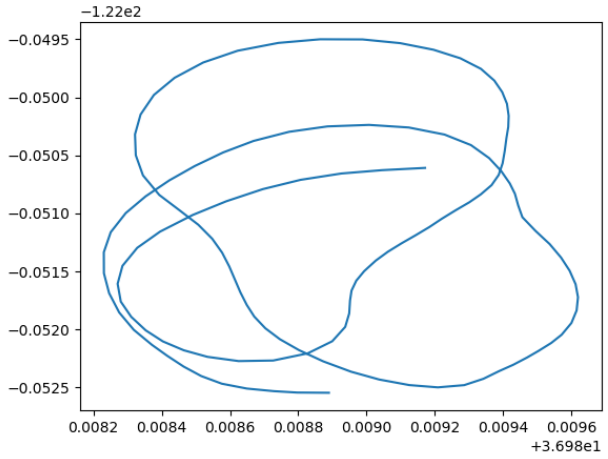

Fig. 9. Plot of the aircraft's maiden flight using GPS coordinates that were logged in real time via the telemetry link.

2.3.3 Design Problems

The biggest and most enveloping design problem I 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 data transmission 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 could not be sent by the current MAVLink protocol.

I realized this when I created a custom packet with an ID value of 11065 and tried to send it. 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 used rather than the updated and more flexible MAVLink version 2. Overwriting the message was a necessary evil to send custom packets.

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 MAVLink), but MAVLink 1 packets continued to be sent. In the interest of time, I decided that simply overwriting the useless packets defined for MAVLink 1 was the best course of action.

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.

2.4 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. Once the difficultly of working with different transmission protocols over several different programming languges is overcome, data transfer's extreme value can be truly realized.

3 Control System

3.1 Introduction

The control systems section will explore the 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 we are just focused on control in general for now.

3.1.1 Treating the Aircraft as a Dubins Vehicle

This section focuses 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 velotic 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

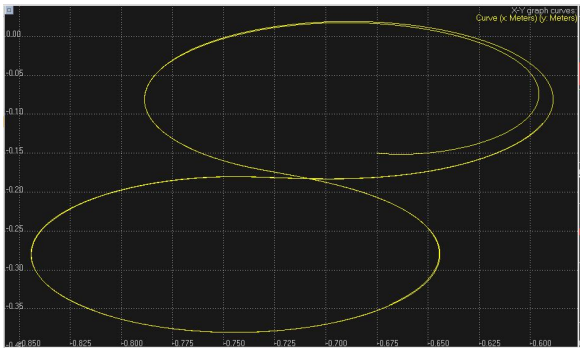

Fig. 10. Dubins vehicle programmed to move in a figure-eight pattern by holding a constant velocity and changing the sign of its turning rate at a specific time interval. An airplane at a constant forward velocity and altitude can be modeled in a similar fashion. Note this picture was taken from a project the author, Kodiak, 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 rotation rate along the vertical 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 turning rate. This equation is similar to the steering wheel on a car - it rolls left and right to make the car turn left and right. In this case, the aircraft itself is the steering wheel rolling left and right to change its heading. 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.

3.1.2 Aircraft Ground Control Stations

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)" ([9], p. 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 working improperly, 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 operators play a crucial role because although the plane can fly mostly, or fully autonomously, "most UAV losses are attributed to operator errors" ([9], p.

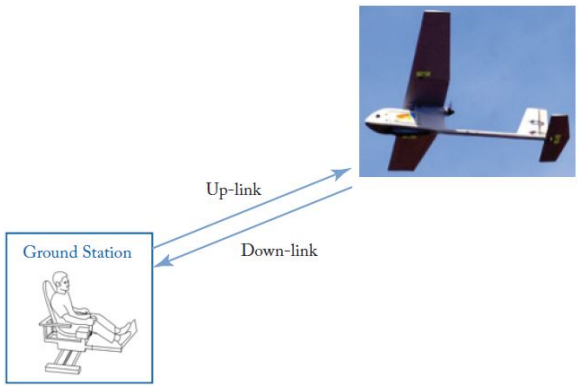

Fig. 11. Diagram displaying and operator of a ground control station to pilot an aircraft. Most stations do not require the operator to sit in a cockpit-like seat, but the exaggeration is useful for a novice UAV enthusiast. [9], p. 142, Figure 8.1.

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.

3.1.3 Glider Soaring via Reinforcement Learning in the Field [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 choice, 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 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.

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 because they are tracking thermals or updrafts in a similar manner. [6]

3.2 Methods

3.2.1 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 planning on 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. The team built a cardboard glider but did not have much luck flying it; the aircraft had unstable pitch oscillations despite the center of gravity being perfect. It ended up crashing, but served as a good tool to learn how to connect all of the electronics. (see Figure **??**). Fortunately around that time, we received

funding and were able to purchase a well designed plane. It has a 2.6m wingspan to generate a ton of lift, and it has an optimal amount of area for solar cell placement. This new plane has no pitch oscillations and flies very well.

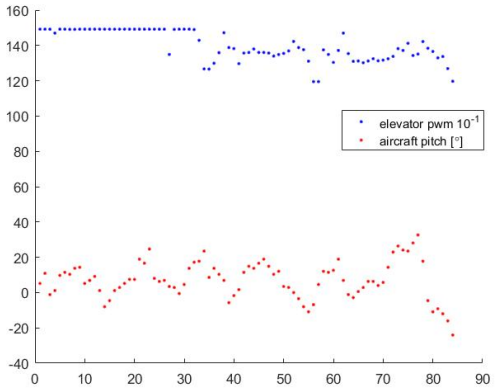

Fig. 13. This plot if of aircraft pitch (red) and elevator deflection (blue) in PWM over the flight duration. A decrease in elevator PWM equates to the control surface deflecting upward, i.e. it causes the aircraft to pitch up. Notice that only pitch up elevator was applied throughout the entire flight, however the plane pitched up and down. This made it clear that the plane was not flying properly. Rather than spending time debugging the issue, we purchased a new plane that was guaranteed to fly well.

3.2.2 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.

3.2.3 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 a typical aircraft. The motor connects to the ESC via three wires with bullet connectors (see Figure 14).

TABLE 1
FrSky X8R Channels to Aircraft Control Surfaces

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

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 pow-

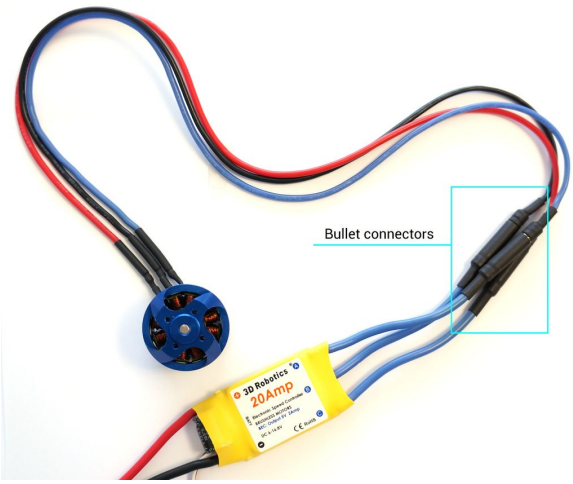

Fig. 14. Brushless motor connected to ESC via bullet connectors found it [8]

ered 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.

3.2.4 Connect Motors, Servos, and Receiver to Pixhawk

Refer to Figure 15 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!

3.2.5 Flash Taranis X9D with FlightDeck Software

The FlighDeck software from Craft and Theory can serve as a backup ground station directly 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 at https://www.open-tx.org.
    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: https://www.open-tx.org/2019/01/06/opentx-2.2.3.
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 at https://downloads.open-tx.org/2.2/sdcard/opentx-x9d%2B/.
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.

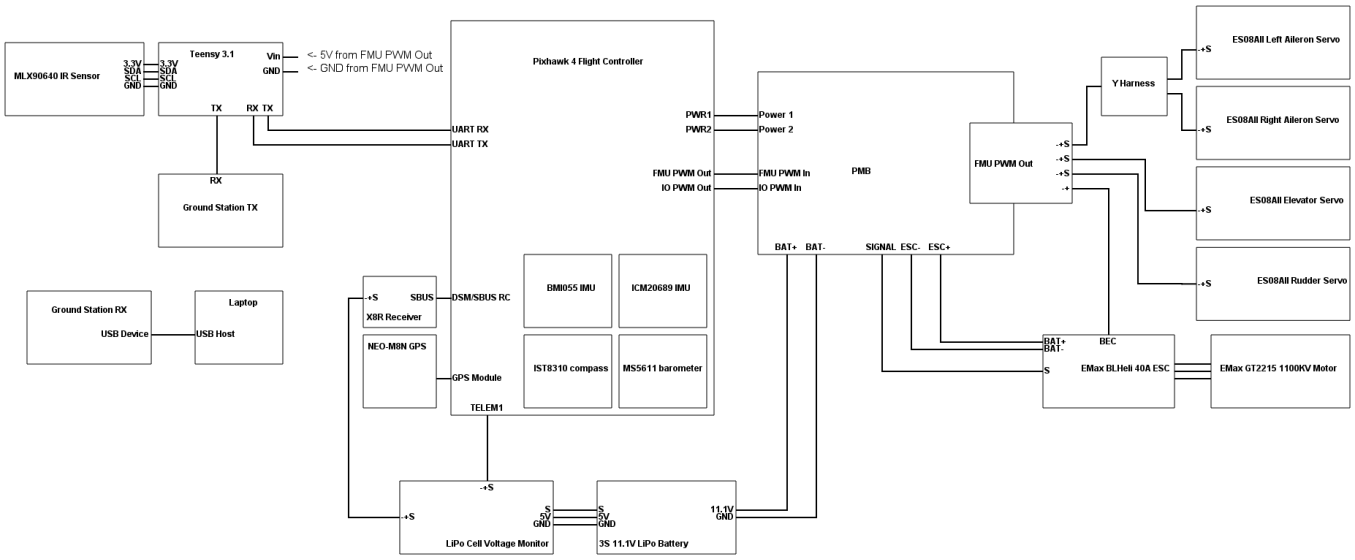

Fig. 15. Diagram of all electronics in the aircraft.

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.

3.3 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 1/4 of the distance of the chord 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 Fire Detection System

4.1 Introduction

The fire detection system onboard the aircraft scans a stream of images for signs of fire. These images are captured by an Infrared camera. A background on the inner workings of this camera will be given below. Infrared imaging is only one of many feasible techniques. An

overview of other techniques explored by other researchers will also be given. Finally, this section will detail the fire detection system implementation, the advantages of our implementation, and the drawbacks of our implementation.

4.2 Comparison of Techniques

4.2.1 Human Observation

There are many tried and true methods for detecting wildfires. The oldest, and most simple method is to station human beings in watch towers in high elevation positions, such as the ridge lines of mountains. This method is not prone to false positives, as human beings are very good at identifying a fire once they see it. Human beings can detect fire through many senses, such as visible smoke, visible flames, burning smells, and changes in temperature or humidity. However, human vision has a limited range. A large number of watch towers and human watchers are necessary to monitor a small area. Also, the logistics of watch towers are complex. New watch towers take time to construct, the system is not mobile. The system also places human beings in danger. A fire could easily spread after detection but before being extinguished, and burn down the watch tower.

4.2.2 Manned Aerial Observation

A more recent technique uses human observers inside large manned aircraft. Unlike the watch tower method, this system keeps human beings out of harms way. This system also has the advantage of being mobile, as a plane can be deployed anywhere. The range of this method is also less limited, as a plane cover an area much larger than a human's vision in a single flight. Large enough planes can even be equipped to combat fires when they are detected.

The downsides of manned aircraft are cost and complexity. Flying a plane is an expensive activity. It also takes a highly trained pilot, and a team of people to maintain the vehicle. The fire department of a small rural town will lack the resources to use this method. Instead, they must rely on the state to provide the aircraft, pilot, and maintenance crew. Unfortunately, small rural towns are most affected by wildfires.

4.2.3 Satellite Imagery

Satellite imagery is another technique employed to detect wildfires. Satellites have an enormous upfront cost, but thankfully many space agencies allow small organizations to access their imagery. Many spaces agencies also allow organizations to place custom modules onboard their satellites, in the form of "Cube Satellites". However, the major drawback of using satellites to detect fires lies in the technical details. Satellite imagery has a very slow frame rate, on the order of days [12]. This delay is simply not suitable. In the span of a day, a fire can grow to an enormous size. With the right combination of weather and fuel, a wildfire can grow at a rate of 300 feet per second [17].

Satellite imagery has another crucial downside: low resolution. Satellites orbit at very high altitudes, on the order of hundreds and thousands of miles above earth's surface [12]. Advanced camera equipment would not be able to detect small fires. Satellite imagery would likely only capture large scale artifacts of fire, like smoke trails. However, these only develop after a fire has grown to a considerable size. If the goal of fire detection is to catch fires early, then satellites are not the right tool for the job.

4.2.4 Optical Camera Imagery

Many real world fire detection systems employ optical cameras. These operate in much the same way as a human observer, and are positioned in high elevation positions. They look horizontally over a region, and the resulting images can be digitally processed, or manually monitored for signs of smoke and fire. A major downside of this approach is the horizontal camera orientation. In densely forested areas, a fire may be obscured by trees while it is small. Optical imagery also cannot be used at night, as smoke is nearly impossible to distinguish in the dark.

The figure below demonstrates an optical image processing algorithm produced by Cruz et al. [11]. For each pixel of an input image, the algorithm computes the probability that the pixel is part of a fire.

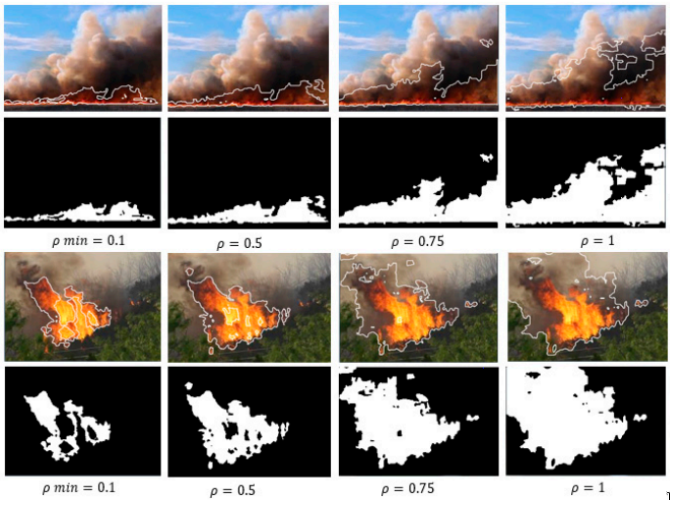

Fig. 16. The Forest Fire Detection Index algorithm with ρ values of 0.1, 0.5, 0.75, 1, and 1.25. Note that the region considered as being "on fire" grows in size with larger values of ρ. For values above 0.75, the algorithm mistakenly classifies smoke as part of the fire.

4.3 Infrared Sensor Background

An infrared sensor is a specific type of spectrometer, which is a device capable of converting electromagnetic radiation into an electrical signal. Spectrometers can be tuned to be sensitive to different wavelengths of radiation. An infrared sensor is tuned to produce the maximum response signal when it receives radiation in the Infrared Region.

According to Planck's Law, all matter radiates energy in proportion to its temperature. The hotter an object is, the more energy it radiates. Plank's Law also tells us that the wavelength at which the most energy is produced decreases with temperature. In other words, hotter objects radiate higher frequency energy, and more total energy than colder objects. The figure below shows various Planck curves, which relate energy, temperature, and wavelength.

There are several wavelength bands within the Infrared Region. The Thermal Infrared Region occupies wavelengths of 8-15 μm. The Medium Wave Infrared Region occupies wavelengths of 3-5μm. A peak in the Thermal Infrared Region range indicates an object that is roughly 248K. The Thermal Infrared Region is not appropriate for our application, because wood burns at 573-973K [15]. Instead, we will use The Medium Wave Infrared Region. A peak in this region indicates an object that is between 600K and 900K.

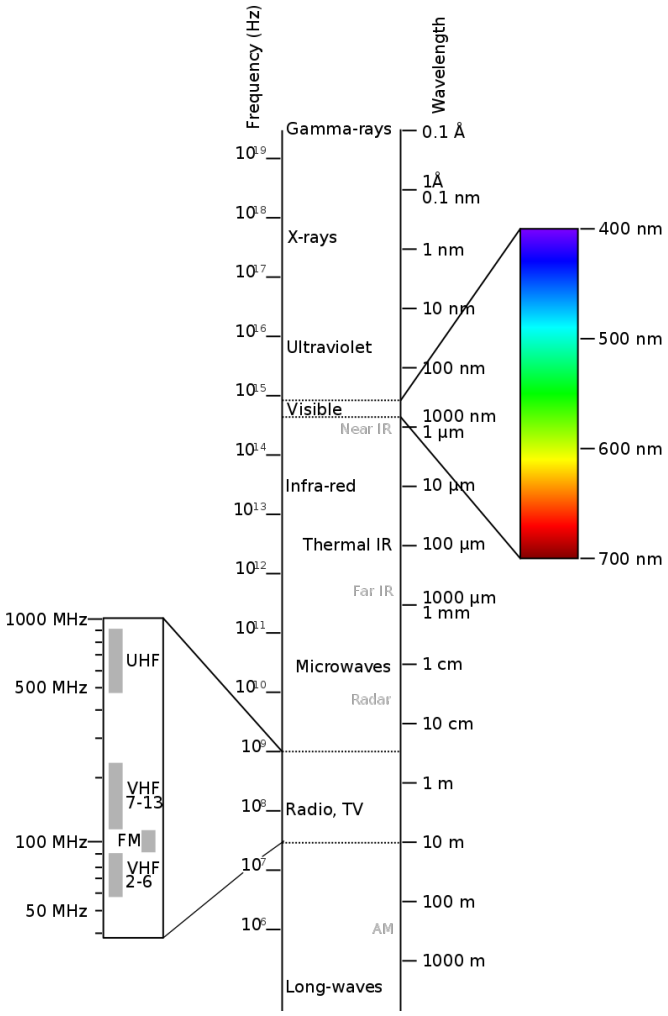

Fig. 17. The electromagnetic spectrum. Radiation between wavelengths of 1μm and 1000μm is classified as Infrared Radation

4.3.1 *Infrared Imagery*

Infrared imagery uses a grid of infrared sensors to create an image. Each sensor corresponds to a single pixel, and varies its electrical resistance in response to infrared radiation that collides with the sensor's surface. Infrared imagery has a key advantage over optical imagery, because IR radiation can be detected in night or day, and can detect objects that are merely hot, and not actually on fire. IR radiation is also not obstructed by smoke or fog, which could obscure the sight line of an optical imagery fire detection system.

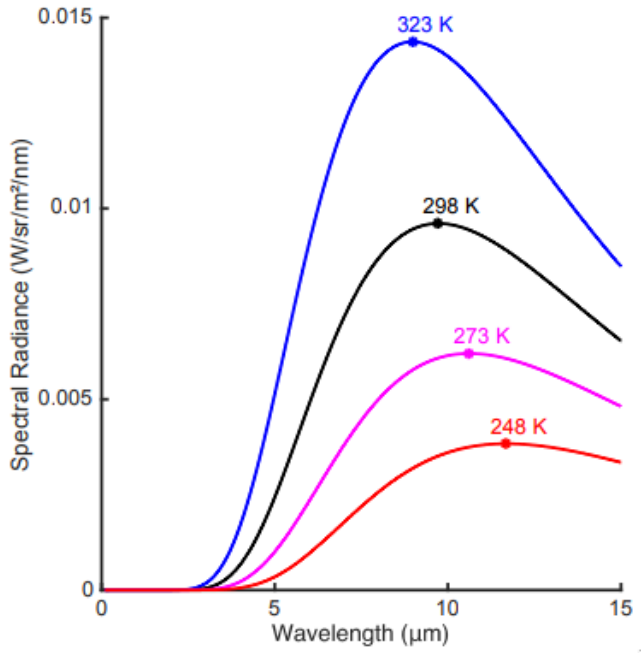

Fig. 18. Planck curves for temperatures 323K, 298K, 273K, and 248K. Notice that as temperature decreases, the peak wavelength increases and the total area under the curve decreases. This means that the temperature of an object can be detected, if we know the peak wavelength of its electromagnetic radiation.

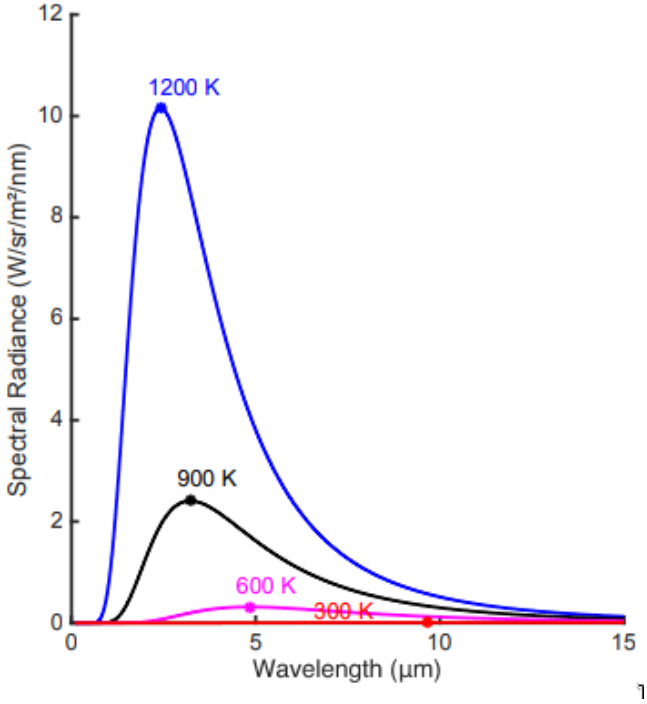

Fig. 19. Planck curves for temperatures 1200K, 900K, 600K, and 300K

4.4 Fire Detection System Implementation

Our fire detection system makes use of an MLX90640, which is a low cost infrared sensor array. The MLX90640 produces images with a resolution of 32 horizontal pixels and 24 vertical pixels. The lens on the camera gives it a 55 degree horizontal field of view, and a 35 degree vertical field of view. The figure below shows the pyramid that encloses the sensor's field of view.

Each pixel represents a 13x10 square foot region, which is the smallest detectable fire size. Any fire smaller than this will not register on the camera, and therefore cannot be detected by the system.

The following image describes the algorithm that backs the fire detection system.

Upon receiving a fire size message, the main microcontroller transmits the current GPS location, current time, and fire size value to the ground station.

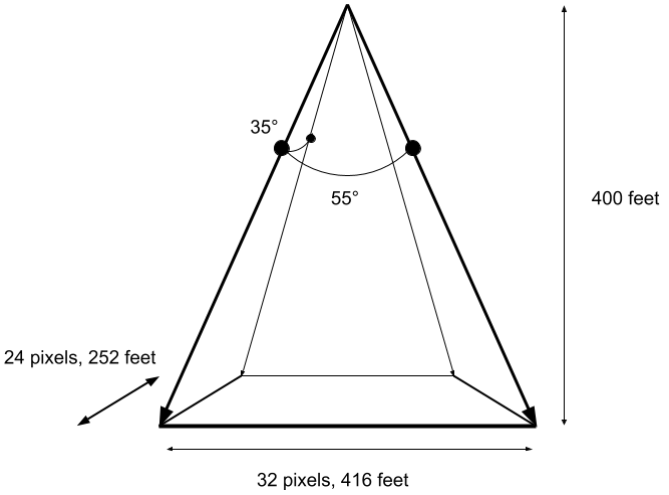

Fig. 20. From a height of 400 feet, images captured by the MLX90640 have a physical width of 416 feet and height of 252 feet. Each pixel represents a 13x10 square foot region.

4.5 Limitations and Problems

The minimum detectable fire size is a crucial metric of our system. The smaller it is, the smaller the fire will be when first responders arrive. The smaller it is, the easier it is to suppress the fire. This metric depends on the cruising altitude of the aircraft, and the resolution of the camera. This system was only tested at heights

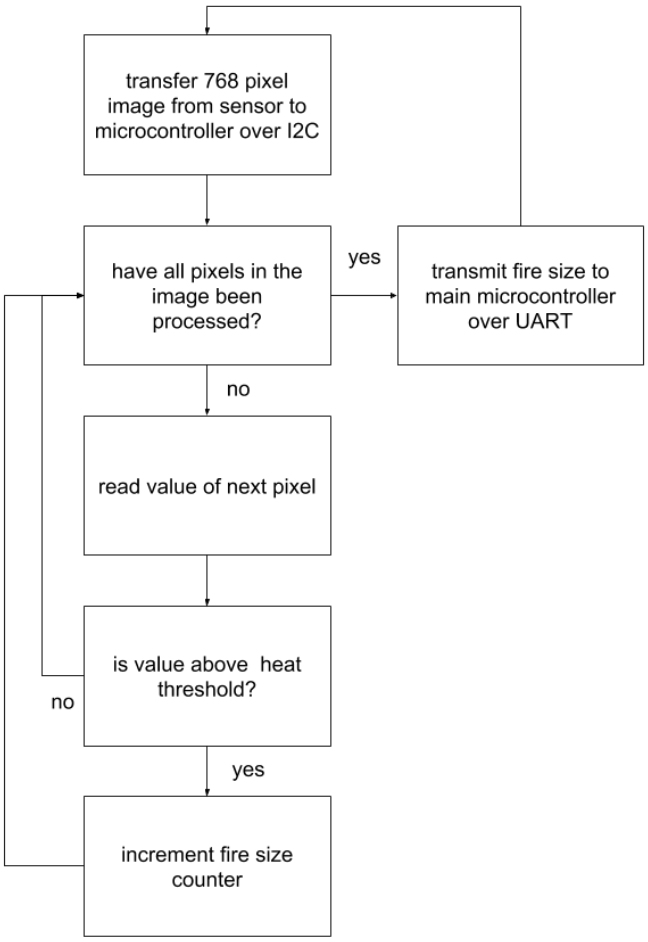

Fig. 21. Fire detection algorithm

below 400 feet, due to FAA regulations. In practice, the cruising altitude should be higher, to mitigate collision with trees or terrain. This negatively affects the minimum detectable fire size. This can be compensated for with a higher resolution camera, however thermal cameras rapidly increase in price with increasing resolution. In short, using higher resolution thermal cameras is not a scalable solution. Using regular optical cameras is one solution, but these have many drawbacks, as discussed in section 4.2. Our fire detection system produces relatively simple information. It records its current GPS location and the current time whenever it flies over a fire. However, there are many more characteristics about a fire that are relevant to fire fighters.

Fire fighters are interested in modeling fires. The location of the starting point of a fire is nice to know, but it is equally important to know what the fire will do after it has started. Fire models attempt to describe the intensity, forward rate of spread, perimeter growth rate, area of perimeter, and shape of perimeter.

Intensity is defined as energy released per unit time, per unit length of the fires perimeter. For example, a fire burning in a forest will typically have a higher intensity than a fire in a grassland. This is because the available fuel is much denser in a forest than in grasslands.

The forward rate of spread of a fire is the speed at which a particular point on the perimeter of the fire advances. This metric has units of distance per unit time. For example, the recent Paradise fire reached a forward rate of spread of 100 yards per second. The forward rate of spread is large when a fire moves uphill, and is positively affected by wind speeds.

The rate of spread, or rate of perimeter growth is self descriptive. It also has units of distance per unit time. An extremely large fire could have a small forward rate of spread, but a large rate of perimeter growth.

These models depend on information about the fire, such as location, the moisture content of the fuel, the wind speed, the wind direction, and the slope of the surrounding terrain. The figure below, produced by F. Albini [15], shows different shapes of fire, depending on wind speed.

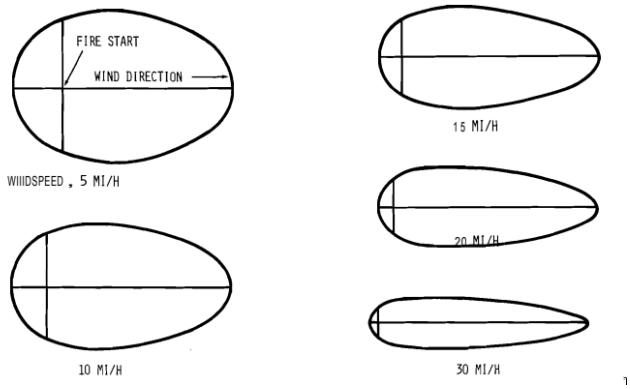

Fig. 22. Approximate fire shapes (not sizes, the scales are arbitrary) for windspeeds of 5, 10, 15, 20, and 30 mi/h [15]

It is important to keep in mind that models are only an approximation of true fire behavior,

and are inherently inaccurate. Fires are also difficult and dangerous to reproduce in a laboratory setting. A large, realistic wildfire would never be purposely started for the sake of measurement. This means that data on wildfires is hard to come by, and of low quality. A model may also be inaccurate if certain assumptions are broken. Many models assume that the region on fire is composed of a single fuel source, which is uniformly dense.

4.6 Future Work

Given these additional fire characteristics, it is only natural that our project's scope should expand and provide additional information, beyond the location of a fire's starting point. Image recognition techniques could be used to measure the area of land covered by a fire. Similarly, image recognition could be used to describe the shape of a fire. This information is useful to fire fighters, because it allows them to predict the rate of spread, and the direction of travel of a fire. It will likely take first responders a non-trivial amount of time to arrive at the scene of a fire, and this information gives them some idea of what to expect. A fire will grow in the meantime between detection and fire fighter response.

5 Conclusion

We now have a control system implemented 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.

Appendix

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.

Max would like to thank Precision Hawk for giving us around two thousand dollars of electronic equipment for free. We would also like to thank the people on forums online that provided help describing ArduPilot functionality.

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&la [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://www.instructables.com/id/Beginners-Guide-to-Connecting-Your-RC-Plane-Electr/ [Accessed: 07-March-2018].

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

[10] Electromagnetic Spectrum, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Electromagnetic_spectrum

[11] Henry Cruz, Martina Eckert, Juan Meneses, Jose-Fernan Martinez, Efficient Forest Fire Detection Index for Application in Unmanned Aerial Systems, 16 June 2016

[12] Ahmad Alkhatib, A Review of Fire Detection Techniques, The University of South Wales, UK, 5 March 2014

[13] Chiachung Chen, Determining the Leaf Emissivity of Three Crops by Infrared Thermometry, MDPI, 15 May 2015

[14] Allison, Johnston, Craig, Jennings, Airborne Optical and Thermal Remote Sensing for Wildfire Detection and Monitoring, MDPI, 18 August 2016
[15] Albini, F. (1976). Estimating wildfire behavior and effects. USDA Forest Service, Intermountain Forest and Range Experiment Station, General Technical Report INT-30, 92 pp.
[16] David M. Doolin and Nicholas Sitar "Wireless sensors for wildfire monitoring", Proc. SPIE 5765, Smart Structures and Materials 2005: Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems, (17 May 2005)
[17] A. L. Westerling, A. Gershunov, T. J. Brown, D. R. Cayan, M. D. Dettinger, Climate and Wildfire in the Western United States, American Meteorological Society, May 2003
[18] I. Bosch, A. Serrano, and L. Vergara, Multisensor Network System for Wildfire Detection Using Infrared Image Processing, The Scientific World Journal, vol. 2013, Article ID 402196, 10 pages, 2013.

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.

Maxwell Bradley is a senior at UC Santa Cruz. He is a computer engineering major speciailizing in computer system, which dabbles in both hardware and software. He will be continuing into the computer engineering MS program here at UCSC after he completes his bachelors, and hopes to work for himself after he graduates. When he isn't laboring over some piece of code, he enjoys distance running, playing the guitar, and checking out breweries.

Zane Bradley is a 4th year Computer Engineering student at the University of California, Santa Cruz. He loves to hiking, traveling, listening to music, and programming in his free time.