F.R.A.T: Fire Rapid Analysis Tracker

John Lomax, Katrina Sharonin, Francesco Crivelli
jclomax25@berkeley.edu, ksharonin@berkeley.edu, francrivelli@berkeley.edu

ABSTRACT

F.R.A.T. is a drone payload designed to enable fire crews to (1) identify and monitor live fire detections (2) classify the potential surface fire Rate of Spread (ROS) in observed areas should a fire ignite in the areas. Using live fire detections along with ROS rate potential, crews can quickly recognize where to target suppression efforts.

For this payload, real-time thermal hotspot generation was achieved with a 32x24 pixel IR camera. These images were scaled and colored according to their relative thermal spectrum. The imagery was processed with an OpenCV algorithm written to identify thermal hotspots over a given threshold size. These “contours” were identified, characterized for their temperature and size, and then projected into the GPS frame of reference to create a hotspot map for the payload and a coordinate list for the next position GPS points to measure at.

The payload samples weather measurements on a real-time basis (relative humidity, temperature, wind), applies geo-positioning to combine the measurements with existing static datasets pre-downloaded in memory (fuel models, slope, aspect), and output real-time predictions of fire rate of spread (chain/hour) along with danger classifications according to the National Wildfire Coordinating Group (NWCG) standard [1]: “Low”, “Moderate to High”, and “Very High to Extreme”)..

1 Applied Course Topics

The following course topics were used in the F.R.A.T. project:

1. **I/O Peripherals:** We are connecting to a temperature and relative humidity sensor, along with a GPS onboard. The temperature/humidity sensor relied on the I2C bus for communication with each peripheral's memory and real time data. The GPS required the UART protocol to enable a serial communication connection to read and parse its NMEA messages.

2. **Networking:** We programmed an MQTT network solution to publish data across any network FRAT was connected to. This enabled any phone or network device to view and access the data and classification results FRAT published.

3. **Sensors & Actuators:** We rely on multiple sensors (temperature, relative humidity, wind speed, thermal camera) for measurements which require correction for accuracy and precision. Specifically, we designed a wind speed sensor designed from the back emf of a 5V DC fan when air was forced through it. This system required us to set up a 16-bit ADC and properly characterize the equation to describe our wind speed sensor for accurate measurements. Additionally, we used specific scaling formulas to map our thermal cameras temperature values to colored images, and to amortize the readings within a classified contour area. Furthermore, satellite measurements stored as static datasets required geographic datum transformations and functions to transition from latitude and longitude to pixel positions, enabling accurate sampling of the datasets.

2 Abstract Design

F.R.A.T. is composed of 6 major abstractions that interact with each other:

1. End Users: F.R.A.T. outputs readings to end users in both an MQTT server stream (see below) and output images+text files that are stored onboard the hardware (Raspberry Pi + SD card) memory
2. MQTT Server: a lightweight protocol to publish and stream information to user devices on the network while the drone is airborne.
3. Datasets: static files queried by the onboard software for rate of spread modeling.

4. Compute Hardware: Raspberry Pi 4B and microSD card, the critical components for executing code and memory storage.
5. Peripherals: all sensors necessary for thermal hot spot detection and rate of spread prediction, including: humidity and temperature sensor, thermal camera, DIY anemometer, and GPS.
6. Software: code hosted on the Compute hardware which reads peripherals, processes data, runs rate of spread classification and thermal hotspot classification, and publishes to the MQTT server as a means of communicating with end users.

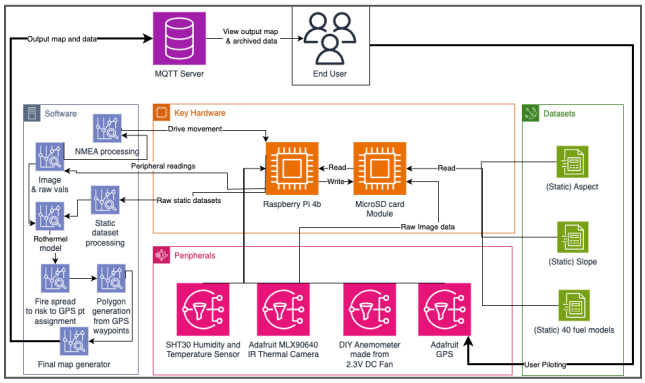


Figure 1: **Abstract design diagram with the major abstractions interacting with each other; software (left, blue) key hardware (middle, orange), peripherals (bottom, pink), datasets (right, green), MQTT server (top, purple), and end-user (top, gray)**

3 Hardware and Software Materials

The following hardware was used:

1. Raspberry Pi 4B+64GB MicroSD Card
2. SHT30 Humidity & Temperature Sensor,
3. MLX90640 Thermal Camera
4. Adafruit Ultimate GPS
5. DIY Anemometer; 5V computer fan
6. HiLetgo ADS1115 16 Bit ADC

The following software was used:

1. OpenCV Python Library
2. gpsd Python Library
3. SHT30 Python library
4. Modified pithermalcam Python Library
5. ADS1115 Python Library
6. Rasterio Python Library
7. Osgeo Python Library
8. PyProj Python Library
9. OpenCV Polygon Creation Libraries

The following static datasets were downloaded and applied:

1. LANDFIRE 40 Scott and Burgan Fire Behavior Fuel Models (FBFM40) (2022) [3]
   a. The FBFM40 categorizes various fuel components (live and dead), size classes, and fuel types into distinct distributions, offering a more extensive range of fuel models across the grass, shrub, timber, and slash categories compared to Anderson's set of 13. It includes more models representing higher dead fuel moisture content and introduces dynamic models for herbaceous components, where loads shift between live and dead to simulate natural curing processes instead of staying constant.
2. LANDFIRE Slope [4]
   a. Slope represents the change of elevation over a specific area. The product is generated from 1 arc-second Digital Elevation Model (DEM) tiles (approximately 30 meters in resolution).
3. LANDFIRE Aspect [5]
   a. Aspect defines the downslope direction in degrees and represents the azimuth of the sloped surfaces across a landscape. The product is generated from 1 arc-second Digital Elevation Models (DEM) tiles (approximately 30 meters in resolution).

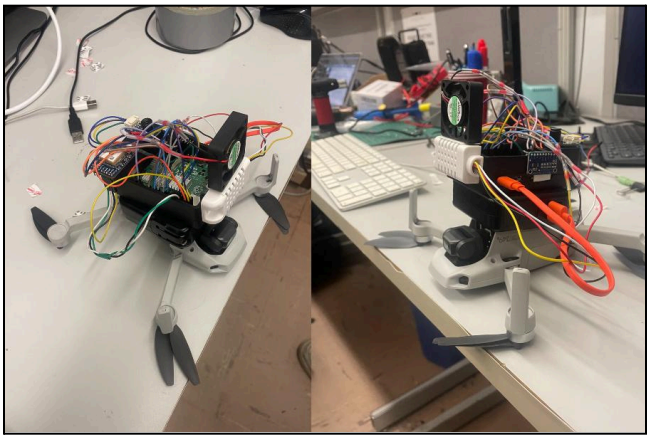


Figure 2: **Image of final assembled FRAT payload mounted to a DJI Mini 2 SE Drone for flight**

4 Implementation and Process

4.1 Github

Our project is located on GitHub for source control; please see the following link for our code: https://github.com/jclomax25/eecs149projectfa23/tree/main/code

4.2 Thermal Image Processing

Processing live thermal hotspots was an important characteristic of this system for usage during live fire events. To meet this requirement, we processed imagery from the MLX90640 over a fast (400kHz) I2C bus. We modified the pithermalcam Python package to suit our project. Each frame of data was scaled relative to the minimum and maximum temperatures in the frame as seen in Equation 1. Once scaled, the image was resized to 800x600 pixels and colorized according to the 'jet' thermal color map using the OpenCV library in Python.

$$\frac{(Temp_{pixel} - Temp_{min}) \times 255}{(Temp_{max} - Temp_{min})}$$

Equation 1: **Pixel temperature to integer scaling formula**

Adding to the pithermalcam source code, we proceeded with the filter_thermalobj() function to enable us to detect and highlight hot signatures of interest. Our algorithm first converted the colorized thermal image to HSV and then set a threshold for the colorized temperatures we wanted to detect contours of. After setting these thresholds, we used OpenCV to find the contours that the thermal image mask revealed.

These contours were then individually processed to two ends: (1) temperature bounding and (2) GPS localization. To bound the temperature of each contour, a rectangular bounding box was defined around each contour, then the pixel coordinates were sampled, and in a copy of the uncolored raw image was scaled back down to their original x, y location of the raw camera image. An inverse of the temperature scaling equation was applied and the original temperature value was found. The temperature of the center of each contour was found by the same method.

To localize each contour once the temperature range was identified, we designed our algorithm to take in the altitude from sea level from the GPS and subtract the identified local elevation. Given an approximate relative altitude, our algorithm then utilized the field of view and frame size of the thermal image to derive a pixel height and width in meters of the scaled 800x600 image. Given the low range of our imager (7 meters), we did not include the curvature of the earth in our GPS coordinate calculation. Using the x, y pixel distance of the center of each contour from the center of the image we then calculated the meter distance. Scaling this distance to the radius of the Earth, we calculated an approximate delta in latitude and longitude as seen by Equation 2. Finally, these contours were mapped onto an image and saved to a local text file, creating a GPS-localized hotspot map, and a list of GPS waypoints for further thermal investigation.

$$Lat_{new} = Lat_{old} + \frac{Dist_{y-coord-to-center}}{Radius_{Earth}} \times \frac{180}{\pi}$$
$$Long_{new} = Long_{old} + \frac{Dist_{x-coord-to-center}}{Radius_{Earth}} \times \frac{180}{\pi} \times \frac{1}{\cos\left(Lat_{old} \times \frac{\pi}{180}\right)}$$

Equation 2: **Latitude and Longitude update formulas**

4.3 Rate-of-Spread (ROS) Processing

The ROS processing involved combining live peripheral readings with static data; the two were used as inputs into the Rothermel model surface fire spread model which outputs a value of ft / minute [6].

4.3.1 Rothermel Model

The Rothermel fire model is built upon fundamental physical principles governing how surface fires behave. These principles formed the basis for physical experiments conducted by Rothermel.

Expressed in a relatively simple equation (Equation 1), the model predicts the rate of fire spread. In essence, the numerator signifies the heat sources, representing the heat produced by burning fuel. Conversely, the denominator accounts for heat sinks—energy lost from the fire system due to processes like convection and diffusion. Fire propagation occurs when the instant heat generated surpasses the heat required to ignite nearby unburned fuel.

$$R = \frac{I_R \xi (1+\phi_W+\phi_S)}{\rho_b \epsilon Q_{ig}}$$

Equation 3: **Rothermel equation; numerator heat sources, denominator heat sinks. Equation taken from source [6]**

Table 1—Definition of components in the final equation for the Rothermel surface fire spread model. $R = \frac{I_R \xi (1+\phi_w+\phi_s)}{\rho_b \varepsilon Q_{ig}}$

Component	Name	Explanation
R	Rate of spread (ft/min)	Flaming front of a surface fire
I_R	Reaction intensity (Btu/ft²/min)	Energy release rate per unit area of fire front
ξ	Propagating flux ratio	Proportion of the reaction intensity that heats adjacent fuel particles to ignition (no wind)
ϕ_w	Wind factor	Dimensionless multiplier that accounts for the effect of wind in increasing the propagating flux ratio
ϕ_s	Slope factor	Dimensionless multiplier that accounts for the effect of slope in increasing the propagating flux ratio
ρ_b	Bulk density (lb/ft³)	Amount of oven-dry fuel per cubic foot of fuel bed
ε	Effective heating number	Proportion of a fuel particle that is heated to ignition temperature at the time flaming combustion starts
Q_{ig}	Heat of preignition (Btu/lb)	Amount of heat required to ignite one pound of fuel
$I_R \xi$	No-wind, no-slope propagating flux (Btu/ft²/min)	Heat release rate from a fire to the fuel ahead of the fire, without wind or slope
$(1+\phi_w+\phi_s)$		Increase to the no-wind, no-slope propagating flux due to wind and slope
$I_R \xi (1+\phi_w+\phi_s)$	Heat source (Btu/ft²/min)	Propagating flux
$\rho_b \varepsilon Q_{ig}$	Heat sink (Btu/ft³)	Heat required to ignite the fuel
$\frac{I_R \xi}{\rho_b \varepsilon Q_{ig}}$	No-wind, no-slope rate of spread (ft/min)	

Table 1: **Rothermel components, taken from [6]**

Equation 3 captures the essence, but implementing the Rothermel model involves complexities beyond this equation. Many of its components necessitate calculations from additional equations, relying on constants or variables for practical use. These crucial constants and relationships were established through a series of meticulously designed experiments conducted during an extensive research initiative.

Table 1 delves deeper into the terms within Equation 3, offering a more comprehensive explanation. Additionally, the F.R.A.T. Python code provided serves as an effective means to comprehend the complete parameterization process. Nevertheless, for the most reliable and comprehensive understanding of the model, referring to the original publication [6] remains the optimal approach.

4.3.2 F.R.A.T. Rothermel Implementation

The final F.R.A.T. The Rothermel model can be expressed as the following function:

```
rate of spread =
get_simple_fire_spread(fuel load, fuel
depth, windspeed, slope, fuel moisture,
fuel sav)
```

Inputs (**bolded** indicated sourced from live peripheral readings, non-bold indicates sourced from static datasets):

fuelload = weight per area of fuel, sourced from FBFM40 (lbs/ft^2)
fuel depth = vertical depth of fuel, sourced from FBFM40 (ft)
wind speed = live measurement (mph)
slope = measured from LF slope (degrees)
fuel moisture = as a proportion of the fuel weight (e.g. 0.05, 5%)*
fuel sav = fuel surface area to volume ratio, sourced from FBFM40 1/ft

Output:
rate of spread = output rate of fire spread (ft/min)

* fuel moisture combines **temperature and relative humidity live readings** with static datasets to produce final value. See 4.3.4 for information

The F.R.A.T. Rothermel model has been simplified to use one fuel type to simulate fire spread, where alternative models are capable of integrating multiple fuel types simultaneously. Our simplification was due to the resolution of the F.R.A.T; each measurement from F.R.A.T. ties wind speed, relative humidity, and temperature to a single latitude longitude point. Multi-fuel models rely on resolutions of 30x20 meters and above. Due to this resolution restriction, we assumed F.R.A.T.'s single point could only represent one fuel at a point in time.

The finalized implementation of the Rothermel can be seen at the Github link:
https://github.com/jclomax25/eecs149projectfa23/blob/kat-dev/code/fire_model.py
Our project relies on the Python Rothermel model implemented by the Prairie Project Knowledge Hub [7].

4.3.3 **Static Dataset Sampling**

After receiving live readings from peripheral devices, F.R.A.T. proceeds to query the on-board static data sets to complete the following inputs: fuelload, fuel depth, fuel moisture, slope, fuel sav

Static datasets are stored as TIFF files, see visual examples in Figure 3. TIFF files, like regular images, can be expressed as arrays; TIFFs are distinguished by their additional geographic metadata. Each pixel holds a numeric value which can be interpreted as is or mapped to a numeric/literal definition.

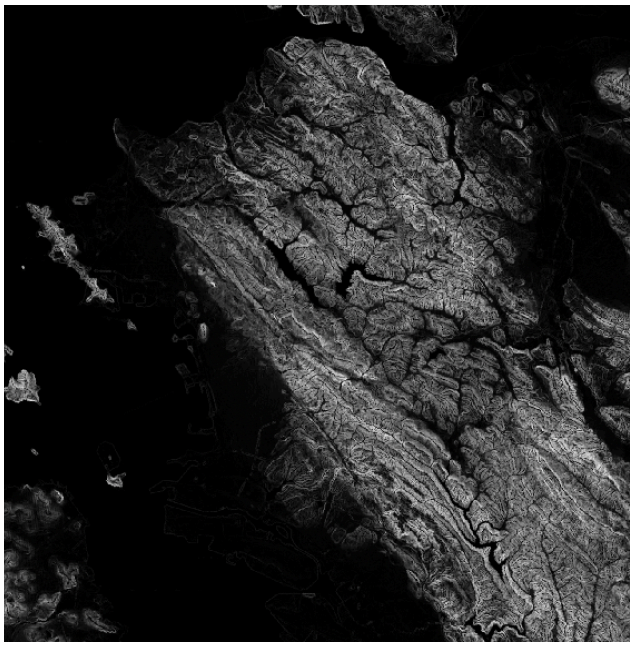


Figure 3: **Example TIFF file for slope (in degrees) for the East Bay Area, loaded in QGIS. Values range from 0 (black) to 90 (white). The gradient represents the varying slope values across the image.**

F.R.A.T. takes a latitude longitude point, and passes it to the code which will proceed to query each dataset for the necessary data. The latitude longitude point must undergo geotransformation, seen in figure below. This transformation uses metadata to transform the coordinate point into the TIFF's array representation (pixel x, pixel y, which range from 0 to the size of the TIFF in pixels; contrast with longitude latitude which rely on a scale from -90 to 90+ and -120 to 120+ respectively).

```python
def image_latlon_pxpy(latitude, longitude, path, crs_form):
    """ apply lat lon to proper pixel x and y form, use known projections
        NOTE needs casting which may impact accuracy
    """
    dataset = rio.open(path, crs=crs_form)
    coords = transform(Proj(init='epsg:4326'), Proj(init=crs_form), longitude, latitude)
    px, py = coords[0], coords[1]
    # print(px,py)
    px_pc = (px - dataset.bounds.left) / (dataset.bounds.right - dataset.bounds.left)
    py_pc = (dataset.bounds.top - py) / (dataset.bounds.top - dataset.bounds.bottom)

    # return (px_pc*dataset.width, py_pc*dataset.height)
    # print(px_pc*dataset.width)
    # print(py_pc*dataset.height)

    return (int(px_pc*dataset.width), int(py_pc*dataset.height))
```

Figure 4: **image_latlon_pxpy function; responsible for transforming latitude and longitude into pixel coordinates.**

After undergoing point transformation, the program generates a 30 meter by 30 meter bounding box for the point and queries each of the TIFF files (slope, aspect, 40 fuel models) for necessary data.

Below describes how each of `fuel load`, `fuel depth`, `fuel moisture`, `slope`, `fuel sav` are derived for the Rothermel input

4.3.4 `fuel moisture`

Fuel moisture requires an extensive combination of live and static values to complete the calculations.
The National Wildfire Coordinating Group (NWCG) defines a set standard for formulating dead fuel moisture content [2]. F.R.A.T. operation is day-time based, so the project uses the daytime derivation method.

1. Using live peripheral readings of relative humidity and temperature (otherwise known as dry bulb temperature), identify the intersection in table A; record as initial percentage (CSV query for F.R.A.T.)
2. Use table D for 1-hr fuel corrections (F.R.A.T. development has occurred November-December)
    a. Query time in PDT
    b. Query Slope TIFF for value
    c. Query Aspect TIFF for value
    d. Use “L” column (the fire is Level, 1000+ or - from viewpoint)
    e. Look for intersection; save value as additional correction
    f. No heavy shading in testing areas, thus 50% shading table indices were ignored
3. Combine initial value with additional correction to receive final fuel moisture estimation

Table A. Reference Fuel Moisture

Dry Bulb Temp (°F)	Relative Humidity (%)																				
	0 to 4	5 to 9	10 to 14	15 to 19	20 to 24	25 to 29	30 to 34	35 to 39	40 to 44	45 to 49	50 to 54	55 to 59	60 to 64	65 to 69	70 to 74	75 to 79	80 to 84	85 to 89	90 to 94	95 to 99	100
10-29	1	2	2	3	4	5	5	6	7	8	8	8	9	9	10	11	12	12	13	13	14
30-49	1	2	2	3	4	5	5	6	7	7	7	8	9	9	10	10	11	12	13	13	13
50-69	1	2	2	3	4	5	5	6	6	7	7	8	8	9	9	10	11	12	12	12	13
70-89	1	1	2	2	3	4	5	5	6	7	7	8	8	8	9	10	10	11	12	12	13
90-109	1	1	2	2	3	4	4	5	6	7	7	8	8	8	9	10	10	11	12	12	13
109+	1	1	2	2	3	4	4	5	6	7	7	8	8	8	9	10	10	11	12	12	12

Table A: RH and Dry Bulb fuel moisture correlation table. Source: NWCG [2]

Table D. 1-hr Fuel Moisture Corrections-Nov-Dec-Jan

Aspect	Slope	Unshaded – Less than 50% shading of surface fuels																	
		0800-0959			1000-1159			1200-1359			1400-1559			1600-1759			1800-1959		
		B	L	A	B	L	A	B	L	A	B	L	A	B	L	A	B	L	A
N	0-30	4	5	6	3	4	5	2	3	4	2	3	4	3	4	5	4	5	6
	31%	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
E	0-30	4	5	6	3	4	4	2	3	3	2	3	3	3	4	5	4	5	6
	31%	4	5	6	2	3	4	2	2	3	3	4	4	4	5	6	4	5	6
S	0-30	4	5	6	3	4	5	2	3	3	2	2	3	3	4	4	4	5	6
	31%	4	5	6	2	3	3	1	1	2	1	1	2	2	3	3	4	5	6
W	0-30	4	5	6	3	4	5	2	3	3	2	3	3	3	4	4	4	5	6
	31%	4	5	6	4	5	6	3	4	4	2	2	3	2	3	4	4	5	6
Shaded – 50 % or more shading of surface fuels due to canopy and/or cloud cover																			
N	All	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
E	All	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
S	All	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
W	All	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6

Table D: Slope, Aspect, Level, Unshading, and Time fuel moisture correction table. Source: NWCG [2]

4.3.4 `fuel load`

Sample from the LANDFIRE 40 Scott and Burgan Fire Behavior Fuel Models (FBFM40). Get fuel type value (e.g. GR2 for grass type). Use metadata from CSV to map fuel type to fuel load.

4.3.4 `fuel depth`

Sample from the LANDFIRE 40 Scott and Burgan Fire Behavior Fuel Models (FBFM40). Get fuel type value (e.g. GR2 for grass type). Use metadata from CSV to map fuel type to fuel depth.

4.3.4 `fuel sav`

Sample from the LANDFIRE 40 Scott and Burgan Fire Behavior Fuel Models (FBFM40). Get fuel type value (e.g. GR2 for grass type). Use metadata from CSV to map fuel type to fuel surface-area-to-volume ratio.

4.3.4 `slope`

Sample from the LANDFIRE slope TIFF.

4.3.4 **Rothermel Considerations**

The F.R.A.T. Rothermel fire model operates on the premise of forecasting fire behavior under an equilibrium condition. The concept asserts that if all inputs—such as fuel characteristics, weather conditions, and topography—remain constant, a fire will eventually reach a state where its behaviors stabilize and remain consistent. This theory underwent rigorous testing during the model's research and development phase at USDA/USFS, thus F.R.A.T. adapts this theory.

However, in practical scenarios, it's crucial to note that fuel loads and environmental conditions are subject to constant variation across both time and space. Real-world conditions rarely maintain a static equilibrium, as fuel characteristics evolve, weather patterns fluctuate, and terrain changes. These dynamic factors introduce variability, influencing fire behavior differently over time and across different locations.

Therefore, while the Rothermel model used by F.R.A.T. offers valuable insights under controlled and consistent conditions, its real-world application requires consideration of the inherent variability in fuel and environmental factors.

4.4 **MQTT Networking Server**

Constant communication and review of F.R.A.T's outputs was a necessary requirement for this project. MQ Telemetry Transport (MQTT) was a lightweight, publisher-subscriber protocol for transmission of data over any network. We configured our onboard Raspberry Pi to be a broker to host the MQTT server. Any device, knowing the IP address of the Raspberry Pi, could connect to its MQTT server and see a live stream of published data as picture below:

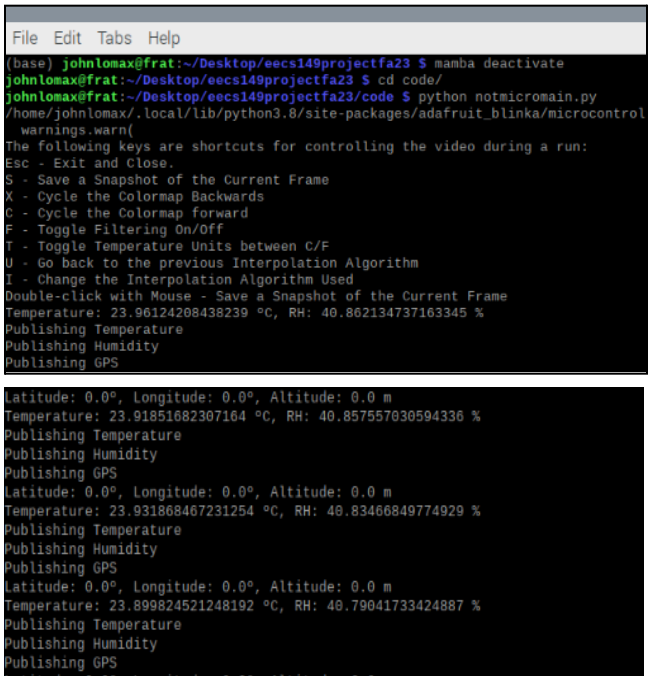
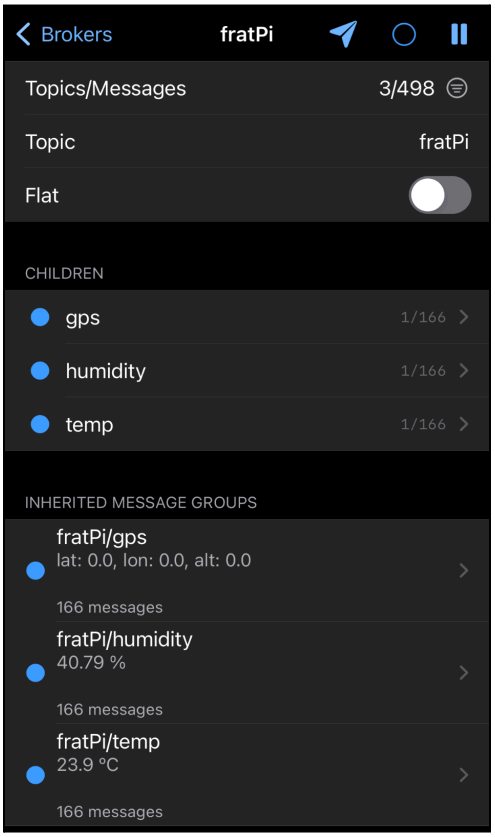


Figure 5: **(Above) The terminal of F.R.A.T publishing its data to the MQTT Server onboard (Lower) An iphone on the same network receiving the published data**



Security of this network was ensured in two ways. First, anyone who wished to access this data required the IP address of the server/Raspberry Pi. This limited any random party from accessing the server. Second, FRAT only publishes to the server, it carries no subscribers, so any foreign party who began publishing on the server could not publish any data that FRAT would subscribe to or interact with. This protects FRAT from malicious attacks over the MQTT server.

Finally, the other key aspect of the MQTT network was it enabled us to run FRAT's main embedded I/O and sensing in parallel with the onboard fire classification. As pictured in Figure 8, separate Python environments run in parallel on FRAT, one running the sensor polling, I/O, and the server publisher. The other ran the rate of spread classification algorithms described in Section 4.3. The rate of spread code ran an MQTT subscriber for the required topics, wind speed, temperature, humidity, GPS location, and ran a classification only when a new set of data was published to the MQTT network. This system enabled all fire and rate of spread modeling and classification to be run onboard FRAT and published once the classification was complete.

5 Results

From our testing we were able to show that we achieved all the functionality we desired from F.R.A.T. For our thermal hotspot imaging, we are able to detect hotspots up to 6 meters away. Our thermal temperature classification had an accuracy of ±3°C. In testing our static temperature and humidity data, we found a temperature accuracy of ±1°C and a humidity accuracy of ±3%. Because of the low delta between GPS coordinates of the drone and thermal hotspots in the frame, we were not able to confirm the precision of our GPS project however intuition would say that it is relatively accurate from the two examples in Figures 6 and 7.

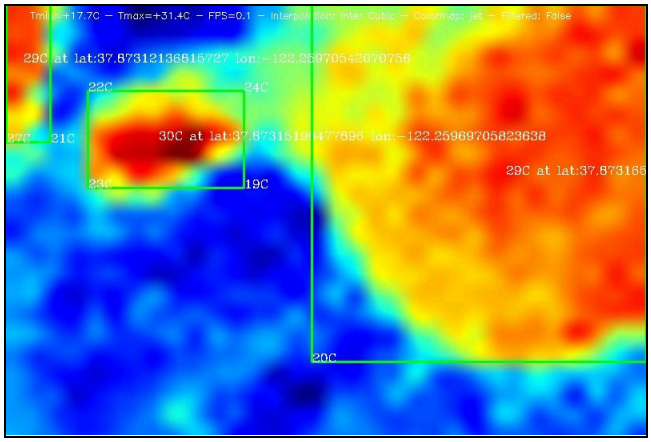
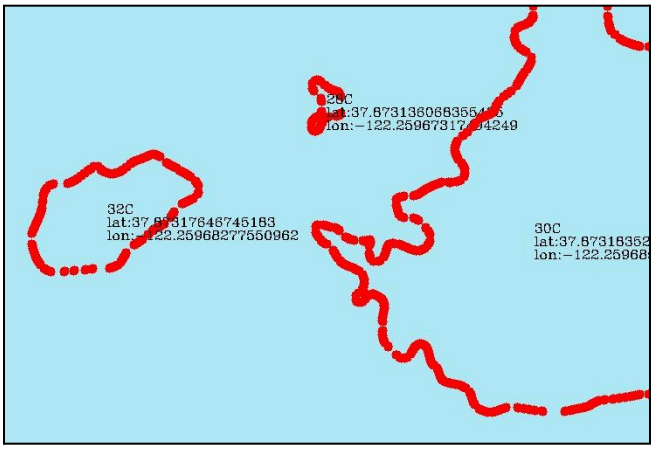


Figure 6: **Initial take-off observations; Francesco in field of view (Above) Raw thermal with contour classified and processed with temperature bounding box (Lower) Classified open contours and geolocated with GPS position**



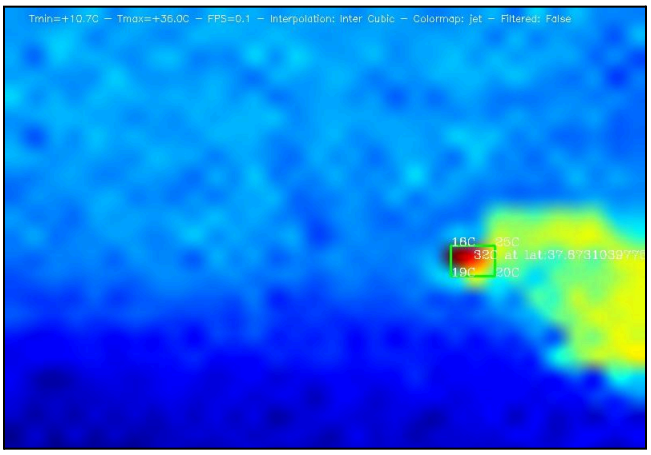
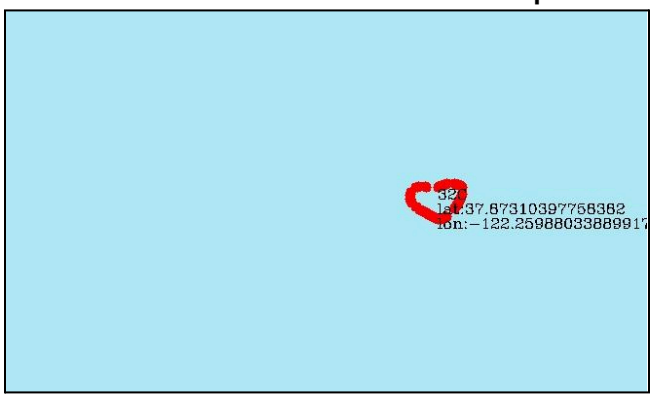


Figure 7: **5+ seconds after Francesco exits frame; sole torch observed which filters out Francesco's arm (Above) Raw thermal with contour classified and processed with temperature bounding box (Lower) Classified open contours and geolocated with GPS position**



For our unique DC fan design, we found that while a forced airstream, such as a breath, would activate a back emf and properly convert to a value the fire model could use. This led us to define an affine model for our fan, Equation 4. However, during testing, we found that wind generally did not provide a strong enough current to rotate the fan while the drone was in flight and as such was not a viable substitute for a proper wind speed reading from an anemometer.

$$14.28 \times (Voltage_{ADC} - 0.45)$$

Equation 4: **Fan Characterization equation converting ADC voltage to wind speed in miles per hour**

F.R.A.T was light enough to be carried by a drone, totaling out to a mass of 173g. The power pack to power F.R.A.T during flight added an additional 140g to the overall mass. This was a success and enabled us to test F.R.A.T in the air during flights.

The MQTT server successfully enabled a connection and live stream of data and images from F.R.A.T over any network, even Berkeley-Visitor. The latency of streamed data to a subscriber over the Berkeley-Visitor network was 1-2 seconds after the publish print statement. The latency between the local data publish and beginning of fire-model classification on the parallel terminal was near-zero.

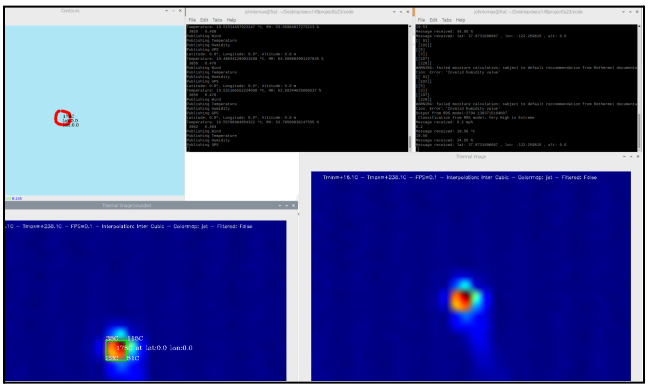


Figure 8: **Screenshot of fully integrated system all thermal hotspot classifications and data published to MQTT, while fire model scripts run in parallel, classifying rate of spread using data from MQTT server**

Once the fire-model MQTT subscriber terminal received each round of static data for rate of spread classification, each contour would become associated with (1) an ROS value (chains/hour) and (2) associated fire behavior classification (“Low”, “Moderate to High”, and “Very High to Extreme”).

Due to time constraints, F.R.A.T. does not auto display the ROS results. Instead, users can use the resulting contour images along with ROS values/classifications to export into a geospatial program, such as QGIS or ArcGIS Pro. Since contours are stored as images, the user must extract the counter via raster classification.

An example export is shown in Figure 9 for a test flight in Cesar Chavez Park, CA. Above is the federal defined standard for rate of spread in chains/hour. Green represents slower rates of spread, while orange to red represent higher values.

Below is the processed F.R.A.T. contours in QGIS; each contour is placed according to its coordinate, projection, and colored to its corresponding ROS classification.

By visual inspection, F.R.A.T. successfully recreates the gradient of ROS between varying regions of the park. The gradient is significant because individual ROS values can vary greatly due to seasonality, one-off weather, etc; data should reflect how certain fuels consistently have higher ROS or lower ROS due to their physical properties and location (southern aspect, inland, etc).

Figure 9: **LANDFIRE Rate of Spread (chains/hr) Classification of Cesar Chavez Park in Berkeley, CA via satellite data (Above) FRAT Rate of Spread (chains/hr) Classification of Cesar Chavez Park in Berkeley, CA (Below)**

6 Challenges and Implications

We faced several challenges throughout our project which impacted our overall results.

A main challenge is the coverage of the drone; due to limited equipment and peripheral range, we could only classify areas of size 24 ft by 21 ft at a time. This impacted our ability to define a ROS gradient that could be directly contrasted with satellite derived estimated (LANDFIRE), which functions on a 30 meter by 30 meter resolution (approximately 98 ft by 98 ft).

In addition, our ROS and classification values were likely impacted by local weather conditions. December is a humid and cold season for the Bay Area, which impacts ROS predictions. Federal standards such as LANDFIRE use historical data that averages years worth of data. However, with recent climate change, it is now difficult to assert which dataset is more accurate and precise for our modern environment.

One other major challenge we faced was calibrating our anemometer design. This was a unique design challenge we wanted to take on. As discussed in Results, the less directed force of wind was not able to get accurate wind speed readings except at very high wind speeds beyond what we could test. This led to a very large coefficient of about 14.28 on our affine function on the ADC which also caused any error in the voltage to be significantly amplified.

A final challenge we faced in this project was signal blocking. Due to the noisy radiation environment of the Berkeley campus and Berkeley in general. Ensuring strong, stable connections for the GPS and MQTT network on FRAT during testing was difficult. Fortunately, testing at Cesar Chavez park gave us separation of this interference and traffic. However, the GPS requirement also prohibited us from doing any indoor testing which also proved a challenge for getting plenty of flight time.

Overall a future continuation of this project would be very valuable to the firefighting industry. Active and static fire surveying are a major challenge and task for the industry. Currently, most of the work is done by personnel, which takes manpower away from other important aspects of wildfire fighting. Our project demonstrates that inexpensive, lightweight systems can produce on-demand and location-based to the expensive systems built by government agencies.

Finally, this project's work with MQTT networking could be expanded further to enable more mesh-based fire surveillance across multiple F.R.A.T.s. This protocol has a lot of potential for expanding this project with capabilities such as controlling a F.R.A.T. over the MQTT server, or creating more secure encrypted MQTT transmissions from F.R.A.T.

ACKNOWLEDGMENTS

We would like to thank the EECS 149 Course Staff for their strong support throughout this course on various topics and project needs.

REFERENCES

[1] Fire Behavior Subcommittee (NWCG). Interpreting Expected Surface Fire Behavior. Fire Behavior Field Reference Guide, PMS 437. https://www.nwcg.gov/publications/pms437/surface-fire/interpreting-expected-behavior

[2] Fire Behavior Subcommittee (NWCG). Dead Fuel Moisture Content. Fire Behavior Field Reference Guide, PMS 437. https://www.nwcg.gov/publications/pms437/fuel-moisture/dead-fuel-moisture-content

[3] LANDFIRE: 40 Scott and Burgan Fire Behavior Fuel Models. [Homepage of the LANDFIRE Project, U.S. Department of Agriculture, Forest Service; U.S. Department of Interior], [Online]. Available: https://www.landfire.gov/fbfm40.php

[4] LANDFIRE: LF2020 Slope Model. [Homepage of the LANDFIRE Project, U.S. Department of Agriculture, Forest Service; U.S. Department of Interior], [Online]. Available: https://www.landfire.gov/slope.php

[5] LANDFIRE: LF2020 Aspect Model. [Homepage of the LANDFIRE Project, U.S. Department of Agriculture, Forest Service; U.S. Department of Interior], [Online]. Available: https://www.landfire.gov/aspect.php

[6] Andrews, Patricia L. 2018. The Rothermel surface fire spread model and associated developments: A comprehensive explanation. Gen. Tech. Rep. RMRS-GTR-371. Fort Collins, CO: U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station. 121 p. https://www.fs.usda.gov/research/treesearch/55928

[7] Open Source. Anonymous Author https://www.prairieprojectknowledgehub.org/books/fire/page/rothermels-simple-fire-model