Flying Under the Radar

Linsey Szabo, Ayesha Gupta, and Angie Bu

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**We propose a universal drone attachment that uses mmWave radar and machine learning (ML) to accurately detect humans in fire and fog search and rescue (SAR) conditions. Our web application interfaces with the device to allow the user to drop a pin to mark a victim's location for subsequent rescue. Our device automates human detection to increase SAR mission efficiency at 50% of the cost of existing market solutions, assuming the average cost of a drone is $1000.

***Index Terms*—**Chassis, drone attachment, GPS, mmWave radar, safety, SAR, sensor, 3D-CNN, web application

I. Introduction

RECENTLY, SAR applications have expanded by using drones [1]. Drones provide many advantages; they limit the exposure of first responders to extreme weather conditions and dangerous terrain, cover large swaths of area efficiently, and are compatible with high-definition cameras and sensors that enhance SAR missions. For these missions, the standard maximum flight altitude is 10 meters, and the duration is a maximum of 30 minutes [2], [3].

However, there are a few drawbacks to the current technology. Because they utilize a high-definition camera and infrared sensing, SAR drones are very expensive. Also, they do not perform well in high temperature conditions and are subject to visible occlusions like fog and smoke. Lastly, they require manual identification of victims.

Our universal drone attachment overcomes these barriers to aid first responders. By using a millimeter-wave (mmWave) radar, we provide a more cost-effective solution, making our technology more accessible for public agencies like fire departments and the National Park Service. Our mmWave radar is also more robust to adverse weather conditions and fire rescue situations, broadening the utility of our SAR drone application. Specifically, we will focus on fire and fog situations. The radar functions by measuring the range (distance to target), Doppler (relative velocity of the target to the drone), and azimuth (angle to the target) data. The Doppler data in particular helps us locate moving targets, since they will create a different velocity relative to the drone compared to stationary targets. Therefore, we will be using this to our advantage. Using machine learning, we automate the manual identification of victims, increasing the efficiency of the mission where rescuing victims is extremely urgent.

To attach our device to the drone, we will encase it with a plastic chassis that will go around the drone's rails. The first responder will fly the drone remotely and hover in certain areas that they choose to examine. The attached mmWave will transmit radar data to the machine learning architecture that is embedded in our web application. By viewing our web application interface, the first responder can see if a human is detected, and if so, they are sent the exact location of detection and can save that location for subsequent rescue.

II. Use-Case Requirements

Since we are building a SAR drone attachment, drone compatibility, increasing the efficiency of missions, safety of first responders, and cost are the most important factors of our application.

Our device must work well with a drone. Its size and weight cannot impede the drone, and it must easily attach. With our material we estimate our device will weigh less than 0.5 kg and have an area of 11 square inches. By encapsulating our device in a plastic chassis, we can attach our device to any drone via its rails. While using plastic in high temperature situations may invoke environmental questions, we have taken this into consideration and will implement a high temperature warning system for the user, so that neither the plastic nor our device is ever damaged. Lastly, our device must work for the duration of a drone flight and work at the standard flight altitude. These last for roughly 30 minutes, so our device must maintain that functionality for at least that long; our device must work within 10 m above the ground.

Moving onto increasing the efficiency of missions, SAR conditions indicate people in crisis situations–it's paramount that these victims are rescued as quickly as possible. Our device facilitates this by automating the human detection process, pinning locations of victims, and having overall real-time functionality. Instead of first responders manually scanning for victims and not only wasting time but also having a more substantial environmental impact, we can point out where victims are using the mmWave radar and machine learning; pinning locations of victims allows first responders to mark spots of rescue, so that they can then systematically dispatch rescuers. Finally, our system must accomplish all of this in a timely manner under the pressure of a crisis.

The safety of first responders is also very important. By making a drone attachment, we help limit their exposure to extreme weather and dangerous terrain conditions.

Finally, since our application would be used by fire departments, it needs to be affordable to enable accessibility. Currently, drones are very expensive and employ a high-definition camera and thermal imaging to find victims. Our mmWave application overcomes this cost barrier, because mmWave radars are substantially cheaper. We can make it easier for fire departments to use our product and improve the overall process of rescuing people from wilderness fires.

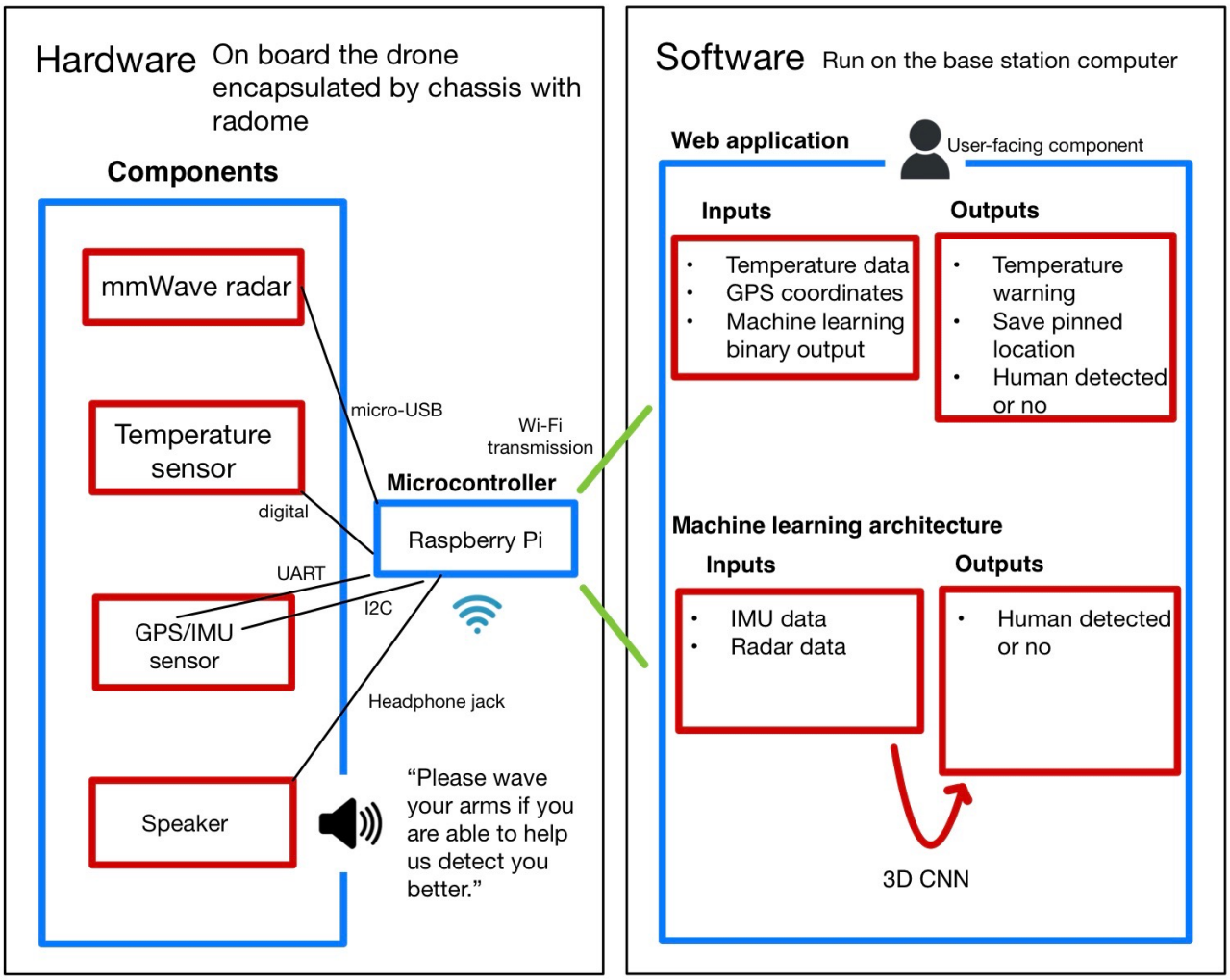

Fig 1: System block diagram

III. Architecture and/or Principle of Operation

Enclosed in a plastic chassis and attached to the drone, the device collects range-Doppler and range-azimuth coordinates through the mmWave radar and temperature, Global Positioning System (GPS), and inertial measurement unit (IMU) data through their respective sensors. The device also includes a Raspberry Pi (RPi), which is key for data transmission. Lastly, on board the drone, there is a speaker, which is a safety addition.

We implemented an important high temperature feature to ensure the functionality of our device during its deployment in fire and fog missions. To prevent the first responder from flying the drone into temperatures that can cause functionality degradation, the temperature sensor data is captured on board the drone and sent via the RPi to the web application. Once the measured temperature exceeds 100° C, the web application indicates to the first responder that the current environment is dangerous for the drone and device, so the first responder can safely fly the drone out of harm's way.

The speaker also enhances the usability of our system. By playing the message, "Please wave your arms if you are able in order to help us detect you better," repeatedly, it effectively alerts nearby victims to wave their arms. This movement will enhance their ability to be detected by the Doppler shift that we observe with the mmWave radar.

Embedded in the web application in the base station computer, the machine learning model is loaded into the web application and reformatted to perform inference on a single radar frame. The range-Doppler and range-azimuth coordinates are sent via the RPi to the machine learning architecture. In practice, inference would be performed when the IMU data–specifically the drone's velocity and horizontal acceleration–indicates that the drone is upright and stationary. However, due to the scope of this class and the inability to fly the drone indoors, we will perform inference on the radar data when a key is pressed. The 3D CNN (convolutional neural network) architecture then runs and determines whether a human has been detected by outputting a 1 for a human or a 1 for no human. That binary value is then returned to the web application.

The web application alerts the first responder of the result with either "Human detected!" or "No human detected." The web application uses the GPS data that has the same timestamp as the radar data to determine the location of the detected human. The web application will then provide the user with the ability to save this location on the map and drop a marker to track this location, using the HERE Maps API.

We have made some changes from our original design. We had small speakers from 18-100 lab but found that they were too quiet for our use case. After purchasing louder speakers, their connection to the RPi now was a headphone jack. Additionally, we decided not to deploy our web application as an EC2 instance. By only running it locally, we made testing and integration more efficient. Lastly, after examining TensorFlow functionalities for loading model weights, we didn't use an API to connect the web application and machine learning architecture. We simply loaded the model weights and reformatted the machine learning code to perform inference on a single radar frame.

IV. Design Requirements

Table I: Design Requirement Metrics

Requirement	***Goal***
mmWave Radar Image Quality	Capture high resolution range-Doppler and range-azimuth data at 5 Hz to determine a human presence from 5 m
Speaker Volume	Heard from within the 5 m range
Temperature Warning Point	100 ºC
GPS Localization Accuracy	0.5 m
Machine Learning Model F1-Score	0.7
Web Application Latency	100 ms
System Latency (good Wi-Fi, bad Wi-Fi)	1 s, 3 s
Power metric	30 min

A mmWave radar must be able to produce data from 5 m away such that the machine learning architecture can detect humans waving their arms from the radar returns. This range is safely within 10 m, which is the standard flight altitude for a SAR drone. The human may be obscured by obstacles such as fog, fire, and smoke.

The speaker will be playing the usability message instructing victims to wave their arms. Not only will this help our machine learning architecture detect them, but it will also make the victims themselves aware of the drone. This message must be audible from our detection range of 5 m.

A temperature sensor that can function between -20º and 120º C and output whether the temperature exceeds 100ºC is to be included to inform the user that the drone and radar system are operating in high temperatures. This warning point is when the plastic chassis and radar are in danger of losing form and functionality [4], [5].

GPS and IMU sensors are included to determine the location, speed, and orientation of the drone. This requirement is set to increase the efficiency of the SAR mission by aiming for a low search radius for the first responders.

An RPi controller is included to collect data from the sensors and transmit the data to a base station computer, which runs our web application. The assembly is powered by a 5V battery, enabling it to function for the 30-minute flight time. It is encased in a plastic enclosure including a radome to protect the electronic components from damage caused by hostile environments the drone may encounter such as smoke and fire.

The machine learning architecture for detecting humans will have an F1-score of at least .7. This metric is defined with precision and recall and "TP" denoting true positives, "FN" denoting false negatives and so on.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \quad (1)$$

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \quad (2)$$

$$F1\text{-}Score = \frac{2 * \mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \quad (3)$$

In this case, using the F1-score as our metric penalizes false negatives and false positives more than accuracy does [6], which is necessary for our use case. It would be very costly to falsely send out a search team for a human that doesn't exist or to not send out a search team when a human is present.

We arrived at this metric by examining similar architectures that have achieved F1 scores in the range .6-.7 when predicting over multiple classes (i.e., identifying many different objects) [7], so we will beat that metric when predicting over just two classes–there is a human present versus there is not. This ties back to the user requirement of increasing the efficiency of the SAR mission. We need to accurately detect humans in order to successfully automate the victim searching process. It is also important to clarify that this metric will be achieved on our own radar data.

The web application latency refers to the time for the page to load, received sensor data to be displayed, and location pin to be saved and visualized on the map. We expect this time to be 100 ms since this is perceived as instantaneous to users. The purpose of the web application ties back to our use case requirement of keeping first responders safe by being able to track detected humans and knowing their exact location without having to search the entire area first.

The system latency refers to the time for collected sensor and radar data to be sent by the RPi and displayed on the web application with the resulting inference result from that radar frame. Similar machine learning architectures ([7]) take approximately 40 ms for inference time, and we have set a 100 ms latency of the web application. Therefore, to achieve real-time functionality for the system, the time of transmission from the RPi to the web application is the more important component. Given that our use case for SAR missions can occur outdoors where there may be slow service, we are allowing for a maximum latency of 3 seconds end-to-end of our entire system, which would occur with an approximate Internet speed of 15 Mbps. When we are testing on campus where the Internet speed is approximately 215 Mbps, we expect to be well below that threshold with a quicker latency time of 1 s.

Our whole system needs to last at least 30 minutes [2]. The typical drone flight duration for a SAR mission is 30 minutes, so by setting this requirement, we ensure drone compatibility in this way.

V. Design Trade Studies

A. Radar

We underwent several changes while selecting and testing frequency modulated continuous wave (FMCW) radar modules. In the beginning, we had the TI AWR1843BOOST and AWR1642BOOST radars at our disposal. The two radars are similar in specifications and performance–both having the same dimensions, weight and configuration of patch antenna array transmitting from 77 to 81 GHz. We chose the AWR1843 because the AWR1642's antenna was more corroded from extended outdoor use, which would introduce more noise into the collected radar data. Next, we tested the Silicon Radar TRX120 [8] a 120 GHz radar which transmits from 119.3 to 126.8 GHz. Owing to its higher frequency, the TRX120 has higher range resolution which would provide higher quality data for resolving the range-Doppler features of humans versus nonhumans. Its maximum range as listed on the datasheet is 10 m, which is within our design requirement, but likely only corner reflectors can be detected from such a distance with humans only being resolvable within 1 m due to a combination of background noise and the relatively low radar cross section of a human body compared to a corner reflector.

After returning to the AWR1843, we used TI's DCA1000EVM in hopes of collecting raw ADC samples from the AWR1843 through Ethernet to create higher definition range-Doppler maps, as had been done by the Smart Robot dataset which also collected data using an AWR1843. However, TI's MMWAVE SDK and MMWAVE Studio GUI are required to collect and parse the raw data, and with dependencies added, take up nearly 1 GB of space which is scarce on the RPi. Since the time it would take to develop code to collect the raw data without MMWAVE Studio seemed substantial, we decided to sacrifice the ideal high velocity-resolution data. Processing the raw data on the RPi or the laptop is slower than the range-doppler and range-azimuth maps generated onboard the AWR1843, which may cause the requirement for inference within 3 seconds of data collection to fail. Additionally, using the AWR1843 is the cheapest and fastest solution out of the radar configurations we've tried, in keeping with our goal to create a low-cost system. The below table includes some of our quantitative and qualitative considerations for selecting a radar configuration:

Table II: Radar Comparison

			Resolution			
Radar	Price	Low Latency	Range	Doppler	Azimuth	Human Detection Range
AWR1843BOOST	$299	39 ms	0.047 m	0.07 m/s	15 degrees	~10 m
AWR1843BOOST + DCA1000EVM	$898	Slowest	Higher	Higher	Higher	Higher
TRX_120_067 + SiRad Easy R4	$600+	Slower	Highest	Highest	Highest	~1 m

We also made tradeoffs in data collection and the scope of human detection after training the neural network on our own collected data. After training the neural network on an additional 3600 samples of preprocessed images including 1800 samples of humans moving subtly, such as standing still or deep breathing, the F1 score decreased from 0.50 to 0.33, indicating that the radar signature of a breathing human cannot be detected using our data, which has relatively low velocity resolution compared to literature, and performs inference on a single frame instead of a sequence. After taking into account the limitations of the data available to us by our choice of hardware, we limited the scope of human detection to a human who is waving their arms or legs, adding a speaker to audibly prompt nearby people to move their arms in order to be detected.

B. Temperature Sensor

Two temperature sensors were considered and tested. The first one was an Adafruit TMP36 with analog output. Although using an analog output sensor is simpler to read on the RPi, with a logical 1 reading when the temperature exceeded a threshold and 0 when the temperature was below that threshold, a voltage divider connected to a transistor was required to set the temperature threshold's output equal to the threshold between a logical 0 and 1 on the RPi's General-purpose input/output (GPIO) pins. Additionally, the analog output is relatively noisy compared to a digital output and would lead to more incorrect readings about whether the temperature was greater than the threshold. Therefore, we switched to the SparkFun TMP102 sensor with a digital output. This sensor had more connections to the RPi and used more memory, but the output is more accurate and precise and does not require additional circuitry to determine whether the temperature is above the threshold.

C. Computer

We considered using either a Raspberry Pi or Nvidia Jetson Nano to control our system. Eventually, we chose the Raspberry Pi 4. The Raspberry Pi includes a dedicated audio output which allows messages to be played loudly and clearly from the speakers such that people can understand spoken messages five meters away with background noise. The Raspberry Pi at $71.20 is also less than half as expensive as the Nvidia Jetson Nano at $149.00 in accordance with our goal to produce a low-cost system. While the Nvidia Jetson has superior processing capabilities for ML due to its GPU, our system performs inference remotely, on the laptop or a computer which may have even more processing power than the Jetson, instead of onboard the Raspberry Pi already occupied with wirelessly streaming large volumes of data, which is better suited for prototyping of controlling and sending data from multiple peripheral devices such as our radar, GPS, temperature sensor, and speaker. Additionally, the popularity of Raspberry Pi lends itself to a richness in hardware, software, and tutorials specifically centered around Raspberry Pi which speeds up

development, such as our GPS module which is designed to fit onto the Raspberry Pi's pins.

D. Machine Learning Architecture

For the machine learning architecture, the tradeoffs considered pertained to the preprocessing and the architecture itself. For preprocessing, we weighed the difference between reconstructing the 3D data on the RPi versus in the architecture. However, reconstruction on the RPi requires sending higher dimensional data over Wi-Fi, which would consume bandwidth that may be very sparse in the wilderness. Therefore, we chose to perform this preprocessing in the machine learning architecture.

Next, for the architecture, we needed a design that would be able to learn complex relationships among the 3D data. Traditionally, CNNs are 2D, so the kernel only slides in two dimensions. However, after conducting research, having a kernel that slides in all three dimensions is better able to learn relationships for 3D data. Therefore, we chose a 3D CNN architecture. To support this approach, [7] and [9] both used 3D CNNs to detect targets using radar data. Additionally, [7] provided relevant metrics for F1-scores, leading to the benchmark .6-.7 range that we mentioned.

Lastly, we considered both PyTorch and TensorFlow for architecture implementation. PyTorch has harder to use functions but is better at handling higher dimensional data. TensorFlow has increased functionality but requires conversion from NumPy arrays to create tensors. Because the dataset we used for training ([6]) was so high dimensional, we ran out of memory when using TensorFlow. Therefore, that network was implemented in PyTorch. However, once we were able to collect data from our radar, we realized that our data was lower dimensional and able to be processed in TensorFlow, so we migrated the model to TensorFlow.

E. Web Application

There were a couple different considerations made in our design choices. The main one was deciding which maps API to use. We originally wanted to use the Google Maps API because it has a great satellite view and other Google Earth tools, which is useful for when we need to zoom into a very specific location on a map. This view would provide more information about the coverage area. However, this API cost money and was not attainable due to course policy. We were able to use HERE Maps API, which is free and still has all the necessary functionalities such as marker adding abilities, map display, scroll, and zoom. However, there was no satellite view that provided as much detail as the Google Maps API. Because this was a secondary requirement, we chose the HERE Maps API.

The next tradeoff considered was how to communicate with and receive information from the RPi. We considered options such as WebSockets, but these options were very complicated and unnecessary to accomplish what we needed. Through research, we found that we could use the Python "requests" package to send HTTP requests from the RPi to the web application, which is simpler and made integration much easier.

Additionally, we were going to use the REST API to run our machine learning model within our web application. We then considered fully training our model beforehand and then loading it into our web application by replicating the code. This proved to be significantly easier than using the API, because we were able to easily integrate the algorithm into existing files as opposed to using a new API with different guidelines than the rest of the functionalities. While our model was quite large, we still found that it was small enough to implement directly into our web application as opposed to deploying it somewhere else and having to send the information. This also allowed us to be able to send the images to the web application and directly use them in the model instead of having to send it somewhere else based on where the model was stored.

Lastly, we considered whether to deploy our application on an SDK (software development kit). While this may have helped improve latency, for testing purposes it is significantly easier and more efficient to run the application locally and debug while making improvements. Since we are still able to send requests to our locally run application from any device, we decided to not deploy so that we could focus on thorough testing and smoother integration.

F. Chassis

Originally, we were going to 3D print our chassis to perfectly fit our device. However, we saved this task for the end of our timeline, and TechSpark lost power during one of our print releases and many of the machines were down. Therefore, we decided to build our chassis from existing spare parts we have. Additionally, we acknowledge that the chassis of our device is specific to the drone and weather conditions and is not pivotal to the main functionality of our project.

VI. System Implementation

Our system has both hardware and software components. The software is split into machine learning and frontend.

A. Hardware

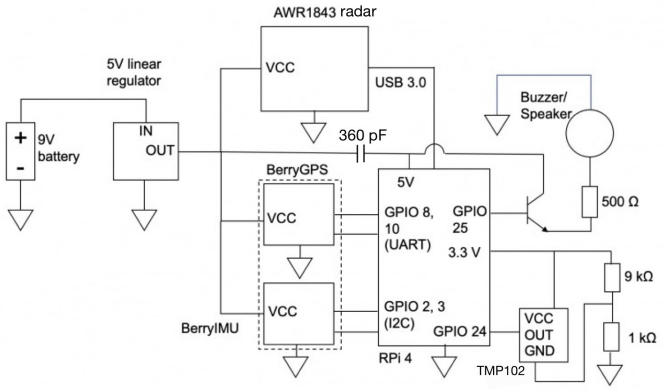

Fig 2: Hardware block diagram

A RPi controls the peripheral devices and reads, preprocesses, and sends the sensor data to a base station computer over WiFi. A rechargeable 5000 MAh battery pack powers the RPi at 5VDC via USB-C, which in turn powers the rest of the peripherals from its 5V and 3.3V pins for over 30 minutes without recharging.

The sensor suite consists of:

- **TI AWR1843BOOST radar** - A 77 GHz automotive radar evaluation module that transmits FMCW chirps from 77 to 81 GHz from its patch antenna array,

configured to transmit using 4 antennas and receive using 2. Further configuration details are listed in section VII.E. The radar board is mounted such that the plane of the antenna array sits parallel to the drone's "up" direction.. The radar transmits range-doppler and range-azimuth data via UART to the RPi , which preprocesses the data and sends the 123 kB of binary data and timestamp to the laptop via WiFi at 5 Hz.

- **Ozzmaker BerryGPS receiver** - This module receives Global Positioning System (GPS) data and outputs NMEA sentences over a serial port. Once the GPS fixes onto satellites, the GNGLL and GNGGA sentences contain the calculated latitude and longitude of the receiver. The coordinates and timestamp are sent to the laptop via WiFi at 1 Hz.
- **Sparkfun TMP102 temperature sensor -** This module measures temperatures between -25◦ and 80◦C at a precision of 0.0625◦ C and outputs the reading to the RPi's GPIO pins. The temperature and timestamp are sent to the laptop via WiFi at 1 Hz, where the user is notified if the temperature exceeds a threshold temperature.
- **Speakers -** These speakers connected the RPi's audio output play sounds to notify nearby people up to and at least 5 meters away of the drone's presence. For example, the system can play a spoken message prompting people to wave their arms, whose radar return is easy to identify as a human by the neural network compared to a still human.

B. Machine Learning Architecture

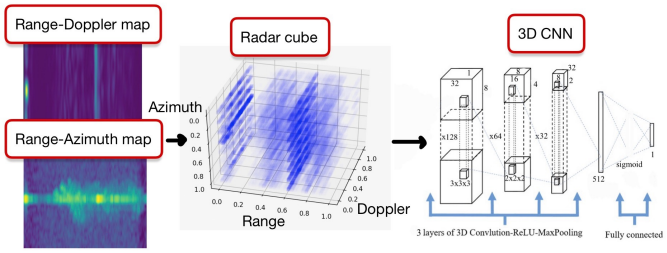

Fig 3: Machine learning architecture block diagram

The machine learning model is loaded into the web application. The radar data–both the range-Doppler and range-azimuth data–is received by the base station computer from the RPi through Wi-Fi. Upon a key press event, the model performs inference on the radar frame.

The collected radar data is preprocessed before being reconstructed into 3D data and fed into the neural network for training and test purposes as well as normal operation as illustrated.

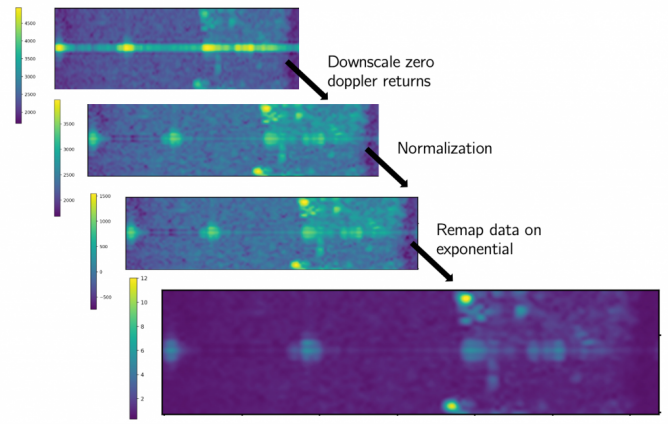

Fig 4: Radar frame preprocessing procedure

1. **Downscale zero-doppler returns**. As most of the indoor scenes in our dataset and use case consist of static returns, the range-Doppler maps at all ranges contain a spike in the three Doppler bins representing the lowest speeds. To reduce the emphasis on these static returns as well as increase emphasis on the moving returns, the three central bins are downscaled by a factor of 0.7.
2. **Normalize range-Doppler map**. The mean of all points in the range-Doppler map is subtracted from the map, but the data is not divided by its standard deviation since this would darken the brightest moving radar returns that we are looking to emphasize.
3. **Exponential weighting of range-Doppler map**. The normalized range-Doppler map is remapped along the exponential function f(x)=40.001x such that the brightest returns, which are more likely to represent macroscopic parts of a moving human, are amplified the most, while the darkest returns, which are more likely to be noise, are suppressed. The weighting function was adjusted such that moderately bright returns, which are likely to represent smaller parts of a moving object, are also sufficiently represented in the data so that moving humans could be distinguished from moving nonhumans.

After this preprocessing, by multiplying the range-Doppler and range-azimuth data along the corresponding axes, we can create a low-fidelity 3D tomographic reconstruction of the scene, resulting in 3D data where the x, y, and z axes represent the azimuth, range, and Doppler values respectively.

This radar cube is fed into the 3D CNN network, which consists of the following in this order: 3 convolution layers each followed by max-pooling, batch normalization, and ReLU (rectified linear unit) activation, and then 1 fully connected layer at the end. The convolution layers work to learn relationships along each of the range, Doppler, and azimuth axes. The fully connected layer does the final reduction in output size step by step by flattening through taking an average along the first axis then passing the output through 512 nodes. At the end, we output a 1 for human presence and a 0 for no human presence. The ReLU activation function prevents the vanishing gradient problem during training and introduces nonlinearities into the network to better learn the potential presence of a human. By employing this 3D CNN architecture,

we will detect micro-Doppler features produced by moving objects in the radar's frame. From these features, we can then deduce which correspond to a moving human.

During training, binary cross entropy will be the loss used to tune the network. This loss is necessary, because we are concerned with binary outputs; this loss will effectively compute the difference between our output and our ground truth label and reduce the model's uncertainty in its predictions. Once detection is complete, a yes or no output is returned to the web application. Then, to deploy the model in the web application, we saved the model weights after independent training and testing and load those weights into the web application code, using TensorFlow functions.

C. Web Application

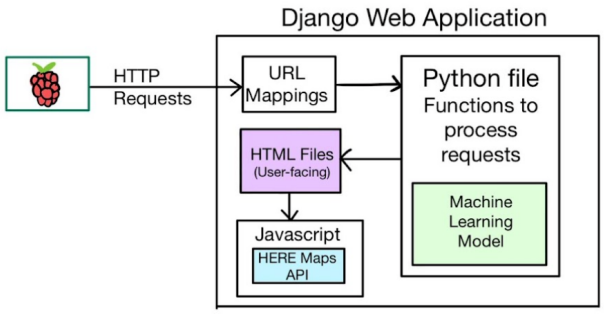

Fig 5: Diagram of web application architecture and interactions.

We used the Django framework in conjunction with AJAX and the HERE Maps API to create our web application. Django allows for easy website creation and maintenance and is implemented in Python, which will integrate well with the machine learning model (also in Python). From the RPi an HTTP Request will be sent to the server, and using Django, we will be able to connect that request to a specific URL, retrieve the associated data, and display the associated page. AJAX will be used in conjunction with Django to send and receive data such that the browser display and behavior is not interrupted, and so that we can update the data quickly and consistently, since we will be sending multiple requests, containing sensor data and radar images, per second. Our web application contains a map display, which was implemented using HERE Maps API, as well as a display of hardware data, which was sent to the web application from the RPi using HTTP requests. Our web application also displayed the output of the machine learning algorithm by receiving the radar images from the RPi and running the machine learning model from within the web application code. For the scope of this class, we will be running the application locally, but we will still be able to accept requests from any host.

D. Integration

In order to connect our hardware and software components, we are using an RPi. This RPi will send the radar data to the machine learning architecture to begin the image processing and run the human detection algorithms. The RPi will also collect the sensor data, as outlined in Figure 2, and send that to our base station computer through Wi-Fi. This base station computer is what runs the web application. This is important because the GPS data needs to be sent from the RPi in order for users to be able to drop pins on the map which is displayed on the web application.

The machine learning model is run from a Python file, which allows it to be loaded into within the web application. Since our web application uses Django, we can use the Django REST Framework, which is free. This framework is easily installable within our application.

VII. Test, Verification and Validation

Our goal for testing all our components is to achieve the metrics outlined in our design requirements summary table (see Table I).

A. Results for mmWave Radar

To test the radar functionality, data of different scenes was captured from a stationary location: scenes with moving humans waving their arms, no humans, and humans obstructed at different distances from 0 to 5 m. While the purpose of the radar is to generate data with characteristics sufficient for the neural network to distinguish humans from moving and stationary nonhumans, tests independent from the neural network were conducted as well, with qualitative results:

- The moving radar return of a human waving their arms from 0 to 5 meters away is noticeable (by human eyes) from the same scene without humans
- The moving radar returns of humans obstructed behind cardboard, glass, wood, and thin concrete are also noticeable, indicating the penetration of mm waves through visually opaque materials, as expected.
- The moving radar returns of stationary, breathing humans are not noticeable. With more information in the radar trade study, the neural network could not discern a human solely by the Doppler return of a human's breath or heartbeat, leading to a change in scope of what comprises a human moving in place.

B. Results for Speaker

To test the functionality of the speaker, we used the ALSA-utils package on the RPi to play audio. We copied an audio file of Linsey saying, "Please wave your arms if you are able to help us detect you," to the RPi and then used the ALSA-utils package. Our speakers also came with an amplifier, which we also attached. To test if our message was audible from 5 m, we measured a spot 5 m away from the speaker and played the message at maximum volume. We were able to hear the message over the background noise in TechSpark.

C. Results for Temperature Sensor

To test the functionality of the temperature sensor, we connected it to the RPi and used the serial monitor to visualize all the current temperature readings. We also purchased an ambient temperature thermometer. By comparing the two readings, we verified that our digital temperature sensor was taking accurate readings. We increased the ambient temperature using a hair dryer and set the temperature alert to be 84°F,

because it would be unsafe to test in temperatures that could degrade the functionality of our system. We again compared the two readings and observed that the accuracy of our temperature sensor was within 1°C. Also, when the temperature reached 84°F, a temperature warning was issued. Since we set this threshold in the RPi code, we can easily change it to 100°C to protect our device in its use case.

D. Results for GPS/IMU Sensor

Several quantitative tests were run to determine the functionality of the GPS module. First, the time to first fix was measured. This metric represents how long it takes for the module to calculate a location.

A *cold start* is when the GPS does not know the information of the satellites and downloads the almanac containing information about the satellites, which is repeatedly broadcast over 12.5 minutes and is valid for 180 days. Ideally, the time to first fix from a cold start would take only 12.5 minutes, but poor reception can lengthen this time. The GPS module was unable to obtain first fix using its internal antenna after continuously running for 8 hours indoors. After placing the module outside a window for 1 hour, the GPS obtained a spotty fix, which was broken once the GPS was taken indoors. One way to improve the GPS reception is to use an external antenna. However, our use case assumes that the drone is flying outdoors, where reception of GPS signals is superior.

A *warm start* is when the GPS has the almanac saved and waits for reception of signals from satellites to obtain the first fix. The BerryGPS can keep the almanac saved for at least 4 hours using the power from a capacitor. After rebooting the RPi and BerryGPS, the GPS is able to obtain a first fix from a warm start in 30 seconds.

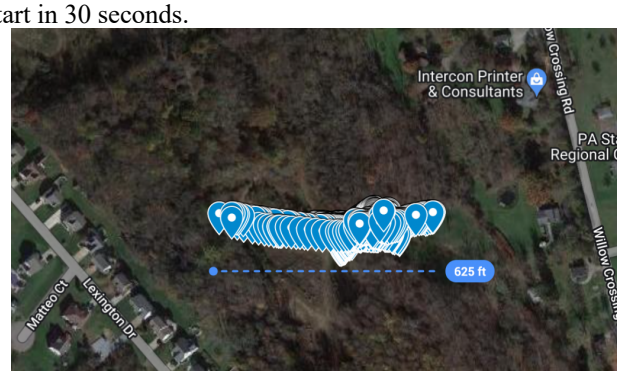

Figure 6: Results of geolocation test with coordinates plotted on map

4927 coordinate pairs were logged from a stationary receiver to determine the accuracy and precision of geolocation. The result shows that the posted coordinates are highly inaccurate but moderately precise. On average, the coordinates are 32 km away from the actual location, and the standard deviation of the coordinates from the average location is 1.5 m.

Overall, tests showed that our GPS module does not meet the design requirement of geolocating itself to 0.5m accuracy, and an alternative way to possibly meet the requirement would be to use the onboard IMU to determine location, using the GPS to mark the starting point.

E. Results for Machine Learning Architecture

To train the neural network, a total of 7200 samples were collected, consisting of 3600 samples containing humans and 3600 samples without humans. Each sample consists of a 128x32 range-Doppler map and a 128x8 range-azimuth map. The range resolution is 0.047 m, giving a total range of 6.0 m in the maps with an unambiguous range of 5.29m. The velocity (Doppler) resolution is 0.07, giving a total velocity space of ±1.12 m/s and a total unambiguous velocity space of 1.00 m/s. The azimuth resolution is 15 degrees, leading to a total field of view (FOV) of 120 degrees around the axis where the beam is steered. This leaves the data with 92% unambiguous points in the range-Doppler maps and 98% unambiguous points in the range-azimuth maps. The sample rate is 5 Hz. Each scene used in the training, validation, and test data was taken indoors.

The data used to train humans contains a human waving their arms and/or legs at several different ranges, azimuths, and elevations. While parts of the humans are moving, the human as a whole is stationary and is not continuously walking or moving away. The data containing humans may also capture moving non-human objects such as cardboard and metal chairs. The human may be situated behind a barrier such as a cardboard wall, a wooden door, and partial obstruction by metal chairs and walls. The data without humans contains a variety of scenes: static images of a room, scenes containing moving nonhumans such as cardboard, chairs, and wasps, and images taken from a moving radar. The scenes were taken from a variety of elevations, but mostly from high resolution looking down at the human to simulate a flying drone.

The test dataset, consisting of 300 human and nonhuman samples each, is taken indoors in a different room than where the training dataset was taken. The human test data similarly contains a human moving arms and legs including behind a cardboard barrier, while the test data without humans contains imagery of a room from both a static and moving radar, as well as imagery of moving nonhuman objects. In compiling the samples into a dataset, a small number (< 100) of garbled samples were removed, and the rest of the samples were preprocessed to reduce noise.

We initially trained the model on the unprocessed data and achieved an F1-score of .33 on the testing data. After preprocessing the training data, the F1-score increased to .5. At this point, we were still considering breathing, non-moving humans as a human presence. However, we decided that this was too challenging for our radar to detect. Therefore, all samples classified as human presence contained humans waving at least their arms. After this relabeling, the model's F1-score increased to .99 on our testing data, achieving our requirement.

F. Results for Web Application

For testing the frontend, we used scripts containing "dummy data" in order to test that we could send information to the web application using python requests and the IP address of the base station computer. From there, we then ran those same scripts from the RPi to make sure that we could create the connection and send data over it securely. Lastly, we gathered the data from the sensors and then sent that over the RPi. Through using these steps, we were able to isolate any issues and quickly set up the pipeline from hardware to front end.

G. Results for Integration

For integrating the machine learning architecture with the web application, we added the fully trained model to our web

application and retested it on our test files to ensure that we saw the same results and metrics. The rest of the results for this relies on the integration of the radar and the ability to send accurate images to the web application.

For integrating the hardware circuit with our web application, we measured a latency of 27 ms when sending GPS and temperature data from the sensors to the web application. We did this by recording the time of the request and the time of when it was received and using these two values to calculate the time it takes to receive the data.

The most intricate component of our system is the radar, specifically the pipeline of sending images from the radar to the web application. For the radar, we recorded the time when the radar recorded the images and sent that information to our web application. We then subtracted that from the time of the base station computer when that data was received. This allowed us to record our latency per request, which ended up being around 39.3 ms on average. Finally, we tested our radar data and machine learning algorithm by recording live images with the radar and sending that data to the frontend and running the machine learning algorithm on it immediately. This is to ensure that radar information is not getting lost when sending.

VIII. Project Management

A. Schedule

We all worked on our individual subsystems in parallel for the two thirds of the semester. At the beginning of April, we began testing the individual subsystems, and by mid-April, we began integrating subsystems. We had some delays from our original schedule, which were due to design changes, namely switching the radar, switching the maps API, and deciding to collect our own data which delayed training our model. See Figure 7 on the next page for the schedule, shown in a Gantt chart. Linsey is blue, Ayesha is brown, Angie is green, and the remaining colors involve multiple people, if not all of us.

Our major tasks included the following:

1. Acquire radar.
2. Set up web application.
3. Capture radar images.
4. Train ML architecture.
5. Validate ML architecture.
6. Test ML on unseen radar images.
7. Send images to web application.
8. Test sensors and speaker.
9. Integrate HERE Maps API.
10. Add marker functionality.
11. Send sensor data to web application.
12. Test ML output and temperature warning display on web application.
13. Test entire system latency and functionality.

B. Team Member Responsibilities

We have both hardware and software components in this project, but we have split it up into three specific concentrations—hardware, machine learning, and web application.

Angie has a lot of experience with hardware and signal processing, and she had a specific interest in using the radar we procured. Angie worked on capturing the images with the radar, connecting the GPS/IMU to the system, and using an RPi to store and send the images to our software system.

Linsey is minoring in machine learning, so she worked on the image processing portion of our project.

Ayesha has experience with building web applications, so she created the frontend portion of our project. Ayesha's secondary responsibility was to help Linsey with the machine learning architecture as needed.

Linsey and Ayesha worked on connecting and testing the speaker and temperature sensor to the system, as well as building the full circuit to connect all of the hardware components to the RPi and the 5 V battery.

All three members tested their individual portions on their own. They altogether collected data for training, and they also all worked on integration. Specifically, Ayesha and Angie worked on sending data from the hardware to the web application, and Angie and Linsey worked on processing the data for machine learning.

C. Bill of Materials and Budget

Our total budget for this project is $83.44. This is because we were able to borrow our most expensive items from labs such as CyLab. If we were to have purchased each item, the total price would have been $549.49. Our bill of materials is located on page 8 (see Table III).

D. Risk Management

As mentioned before, we did experience delays from our original schedule due to malfunctions and design changes. When we experienced issues with our dataset, we pivoted to collecting our own data and made sure it was small enough to be run faster than the dataset we were dealing with so that it would not delay us significantly.

In addition, when we did not meet our F1-score for the machine learning architecture, we pivoted by adjusting our dataset so that we were training on images of either moving humans or no humans, as opposed to having images of breathing/non-moving humans also. Having this data dropped our F1-score quite significantly, but removing it helped us surpass our goal for our F1-score. This helped us remove a great risk of poor detection scores.

For our web application, we had allowed for a bit of a buffer in our timeline, so switching our map API did not hugely affect our schedule.

When any hardware was not meeting the metrics we had set, we quickly ordered new parts. Specifically, our speakers were not loud enough so we ordered louder ones. Our temperature sensor worked well, so we were able to comfortably work with that. We did attempt to switch our radar to improve the resolution, however we had connection issues with this, so we ended up going back to our original radar. This did add a bit of time, but we immediately began testing our radar once we finally switched back to it and began capturing data immediately to avoid any further delays.

IX. Ethical Issues

The main concern with our device is privacy. Its intended use is for SAR wilderness missions. However, because mmWave radar can overcome visible occlusions, we worry about people's

concern about being able to detect them without their consent. To combat this, we encourage strict regulation to make sure our device is only available to public departments for SAR missions. Additionally, we save locations of victims using our GPS/IMU sensor data. If this data was stolen, it could raise privacy concerns. Therefore, in its use case, we recommend encrypting this data before storing it to prevent any attacks.

Because our device can be considered as drone technology, we make minimal environmental impact during our device's actual use. This is key to the wilderness environment in which it will operate.

Safety is also a paramount ethical issue for our product. We aim to keep first responders safe by limiting their exposure to harsh SAR conditions through our automated detection of humans deployed on a drone.

Lastly, we increase accessibility to helpful SAR technology by beating the price of current technology. SAR drones with an HD camera and thermal sensor retail for $3300 [2], while our device is compatible with any drone and uses cheaper technology, mainly the mmWave radar.

X. Related Work

To obtain our dataset, we examined this study [10] that collected FMCW data with a mmWave radar mounted on a stationary drone. While this study collects data in several scenarios, we focused on the one where a corner reflector, an aluminum foil pyramidal reflector, is placed at the center of the open space and the drone hovers in front of it; this scenario

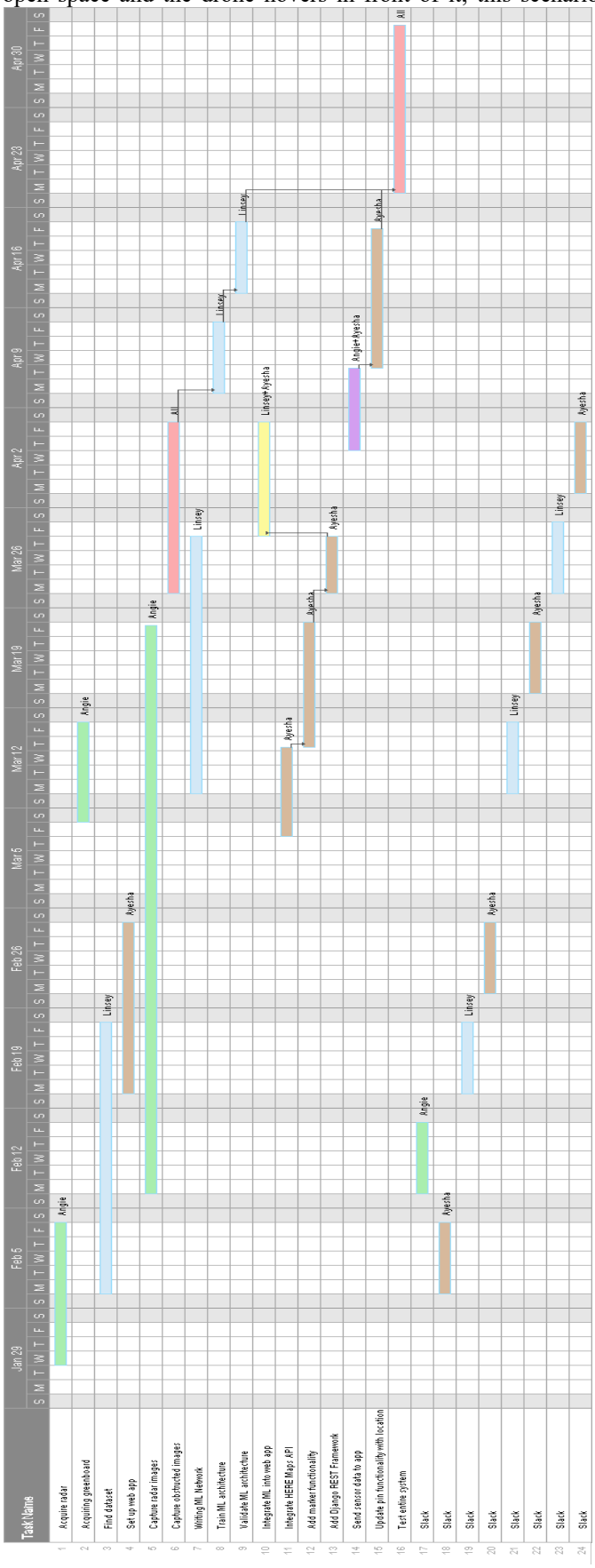

Figure 7: Updated Gantt chart with schedule changes.

provides us with 2869 training examples. The corner reflector is our model human as it ensures strong returns for the radar signal. We acknowledge that this corner reflector isn't moving like our intended human target would; it will only mimic a stationary human with much smaller Doppler shifts like e.g., breathing. Therefore, it will only be used for initial training of the machine learning architecture.

For the system implementation, we examined two studies [7] and [9] that used radar to classify targets on the road. Both construct 3D cubelets to accurately represent the range, Doppler, and azimuth data. They also use 3D CNNs to classify their targets. Therefore, for our architecture, we reconstruct the 3D representation using the range-Doppler and range-azimuth data and employ a 3D CNN architecture. Although these papers classify over multiple classes (they focus on cars, bikes, and pedestrians), we adopted their methods for our two-class prediction problem.

For the architecture code, we followed the construction of the network in [12]. It details a 3D CNN network for classifying CT scans. Therefore, [7] and [12] were key to building the machine learning architecture.

XI. Summary

Overall, our system was able to meet the design specifications we set out for. We achieved almost every requirement, such as the machine learning F1-score, the system latency, and the temperature warning point. One metric we did not meet was our GPS localization metric of accurately pinpointing the location of our device within a 0.5 m radius. We would want to work to either use a new GPS module or improve the fixing abilities of the current one to detect this range more accurately. However, we are aware that our module works better outside which makes more sense for our use case, but we were unable to test this.

A. Future Work

There is a lot of room to expand upon this project, but it is unfortunately outside the scope of this class. We would like to test our device on an actual drone. This would allow us to test our product outside and in more realistic conditions that apply to our use case. It would also allow us to test the functionality of our chassis and how it keeps our entire device together and safe. This would also help us measure if our device was too heavy or too large to be attached to a drone.

In addition, we were only able to gather data with moving humans and other moving objects such as chairs. However, in wildlife, we would see more animals and other moving species that are not humans. Therefore, this could affect our detection algorithm and cause it to produce false positives. We also were not able to test the IMU sensor to understand when we should perform inference based on the horizontal acceleration of the drone. This comes with using a drone for testing as well.

Lastly, we would like to further test our device with poor network signals that more accurately simulate our use case of an outdoor fire catastrophe. Since our presentations did not rely on poor Wi-Fi, we wanted to ensure the functionality of each component and really prioritize integration, and so we were unable to test this extensively. However, this would be an important future test for us to meet the user requirement of speed with low strength signals. In practice, this could require us to change our communication strategy if the drone was too far away from the base station computer or if the signal was too weak.

B. Lessons Learned

We learned a lot throughout this project about how to deal with making complex systems that have multiple different components. As we began testing throughout the semester, we learned the importance of meticulously testing each layer of each subsystem. However, one lesson we still learned was about the difficulty of integration. While we knew this would be a challenge and did allot time for the components to be integrated, there were still problems when trying to integrate no matter how much we tested each individual subsystem.

Glossary of Acronyms

EC2 – Elastic Cloud Compute
FMCW – frequency modulated continuous wave
FOV – field of view
GPIO – General-purpose input/output
GPS – Global Positioning System
GUI – graphical user interface
IMU – Inertial measurement unit
mmWave – millimeter-wave
ML – machine learning
ReLU – rectified linear unit
RPi – Raspberry Pi
SAR – search and rescue
SDK – Software Development Kit

References

[1] R. Tariq, M. Rahim, N. Aslam, N. Bawany and U. Faseeha, "DronAID : A Smart Human Detection Drone for Rescue," 2018 15th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT (HONET-ICT), Islamabad, Pakistan, 2018, pp. 33-37, doi: 10.1109/HONET.2018.8551326.

[2] "Inspire 2 - DJI." *DJI Official*, DJI, https://www.dji.com/inspire-2.

[3] Vision Aerial, Vision Aerial. "How to Use Drones for Search and Rescue." *Vision Aerial*, Vision Aerial, Inc., 15 June 2021, https://visionaerial.com/how-to-use-drones-for-search-and-rescue/.

[4] Texas Instruments, "AWR1843AOP Single-chip 77- and 79-GHz FMCW mmWave Sensor Antennas-On- Package (AOP)," 1 Features datasheet, March 2021 [Revised July 2022]

[5] "At What Temperature Does Plastic Melt?" KIVO Flexible Plastics, KIVO, 9 Apr. 2021, https://www.kivo.nl/en/knowledge-base/faq-about-pe/at-what-temperature-does-plastic-melt/

[6] Huilgol, Purva. "Accuracy vs. F1-Score." *Medium*, Analytics Vidhya, 24 Aug. 2019, https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2.

[7] Palffy, Andras, et al. IEEE ROBOTICS AND AUTOMATION LETTERS, 2020, *CNN Based Road User Detection Using the 3D Radar Cube*, https://arxiv.org/pdf/2004.12165.pdf.

[8] "120 GHz Transceiver trx_120_001 - Silicon Radar Gmbh." Silicon Radar GmbH - FORGET EVERYTHING You Thought You Knew about Radar, Silicon Radar, 28 Oct. 2022, https://siliconradar.com/products/single-product/120-ghz-radar-transceiver/#datasheet

[9] K. Aziz, E. De Greef, M. Rykunov, A. Bourdoux and H. Sahli, "Radar-camera Fusion for Road Target Classification," 2020 IEEE Radar Conference (RadarConf20), Florence, Italy, 2020, pp. 1-6, doi: 10.1109/RadarConf2043947.2020.9266510.

[10] Safa, Ali, et al. IDLab, Ghent University, Leuven, Belgium, 2023, *FMCW Radar Sensing for Indoor Drones Using Learned*

Representations, https://arxiv.org/pdf/2301.02451.pdf. Accessed 2 Mar. 2023.

[11] Dronedek. "How Much Weight Can a Delivery Drone Carry?" *Dronedek The Mailbox Of The Future*, Dronedek, 10 May 2021, https://www.dronedek.com/news/how-much-weight-can-a-delivery-drone-carry/.

[12] Zunair, Hasib. "Keras Documentation: 3D Image Classification from CT Scans." *Keras*, Keras, 23 Sept. 2020, https://keras.io/examples/vision/3D_image_classification/.

TABLE III. BILL OF MATERIALS

Total budget: $83.44, Market price: $549.49

Description: mmWave radar **Name:** AWR1843BOOST **Manufacturer:** Texas Instruments **Cost:** $0 ($352.82 on the market) **Notes:** Radar borrowed from CyLab
Description: Controller for the sensors **Name:** Raspberry Pi 4 Model B with 8GB RAM **Manufacturer:** Element14 **Cost:** $0 ($75 on the market) **Notes:**
Description: Temperature sensor **Name:** TMP102 **Manufacturer:** SparkFun **Cost:** $0 ($5.50 on the market) **Notes:** Had from previous classes
Description: GPS and IMU sensor **Name:** BerryGPS-IMUv4 **Manufacturer:** OzzMaker **Cost:** $71.20 **Notes:** Single board with separate power supplies and communications
Description: Speakers **Name:** Degraw DIY Speaker Kit **Manufacturer:** Degraw **Cost:** $11.99 **Notes:** Ordered on Amazon
Description: 5V battery pack **Name:** Plank 5000 mAh Bamboo Wireless Power Bank **Manufacturer:** PCNA **Cost:** $0 ($32.98 on the market) **Notes:** Had from a previous event
Description: NPN transistor **Cost:** $0 **Notes:** From previous coursework
Description: HERE Maps API **Name:** HERE Maps API **Manufacturer:** HERE **Cost:** $0 **Notes:** Downloaded from online source
Description: Resistors **Name:** 500 Ω, 9 kΩ, 1 kΩ resistors **Cost:** $0 **Notes:** From previous coursework
Description: Plastic chassis **Name:** Plastic chassis **Manufacturer:** Miscellaneous **Cost:** $0 **Notes:** Self-designed from spare parts we already had
Description: Wires **Manufacturer:** **Cost:** $0 **Notes:** From previous coursework