

INTELLIGENT FIRE DETECTION AND RESPONSE SYSTEM WITH DYNAMIC NOZZLE CONTROL AND EVACUATION PLANNING

PROPOSAL PRESENTATION

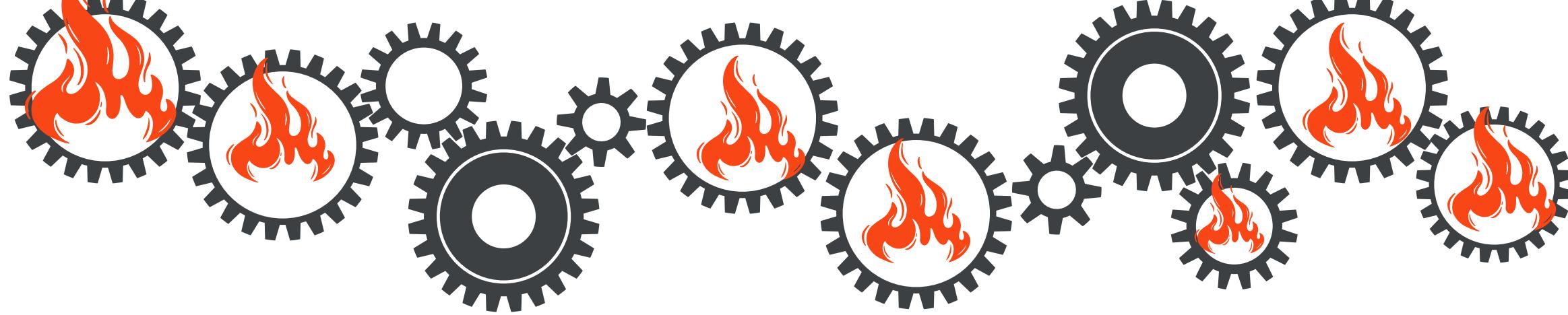
R24-098



SLIIT
FACULTY OF COMPUTING

08/05/2024

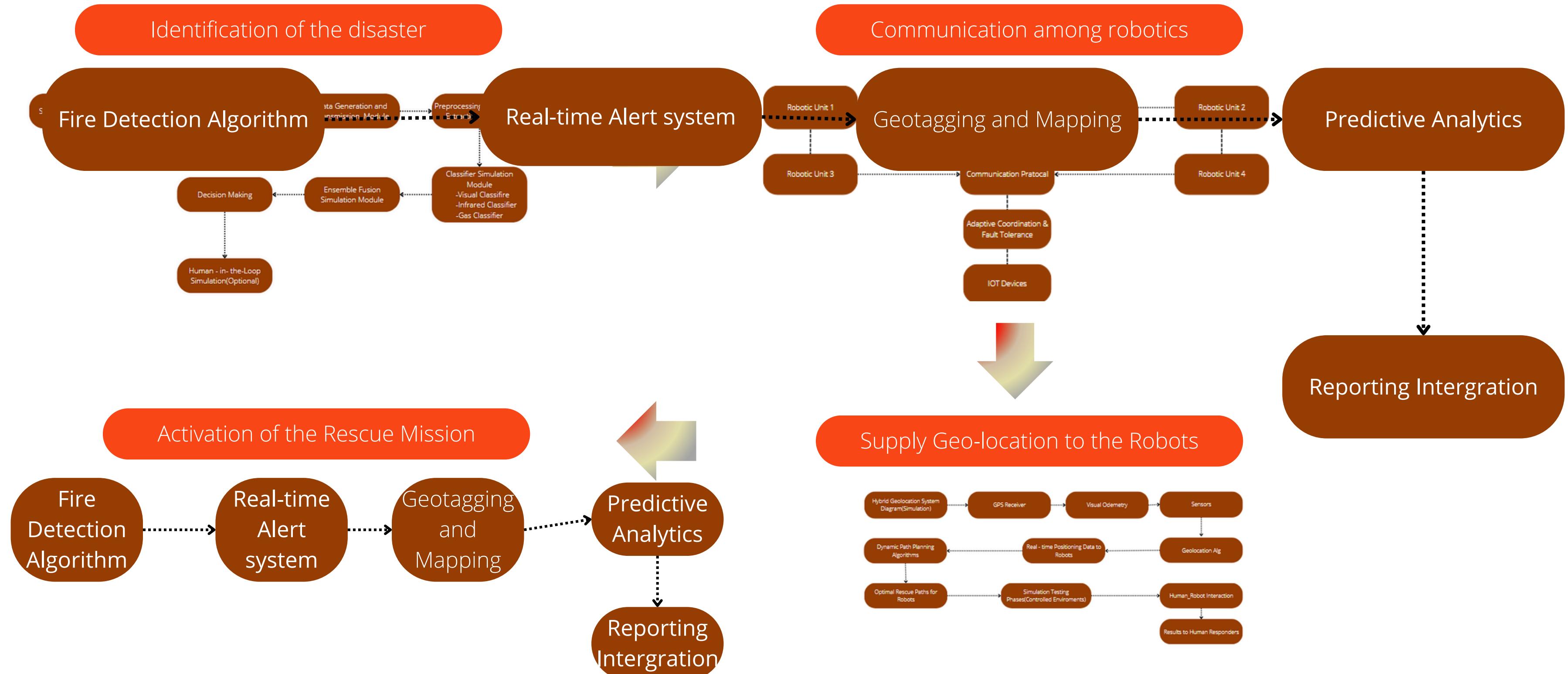
01



INTRODUCTION

Introducing our Dynamic Fire Response System, a cutting-edge solution that redefines fire management. Incorporating advanced fire detection technology, encrypted communication, and dynamic nozzle control, it swiftly identifies flames and optimizes water dispersal for efficient suppression. Beyond firefighting, it employs intelligent communication to relay crucial information and maps out optimal evacuation routes, ensuring swift and safe egress from affected areas. By integrating detection, communication, fire control, and evacuation planning, our system sets a new standard in fire management, prioritizing safety and efficiency in emergencies.

System Overview Diagram



Technologies

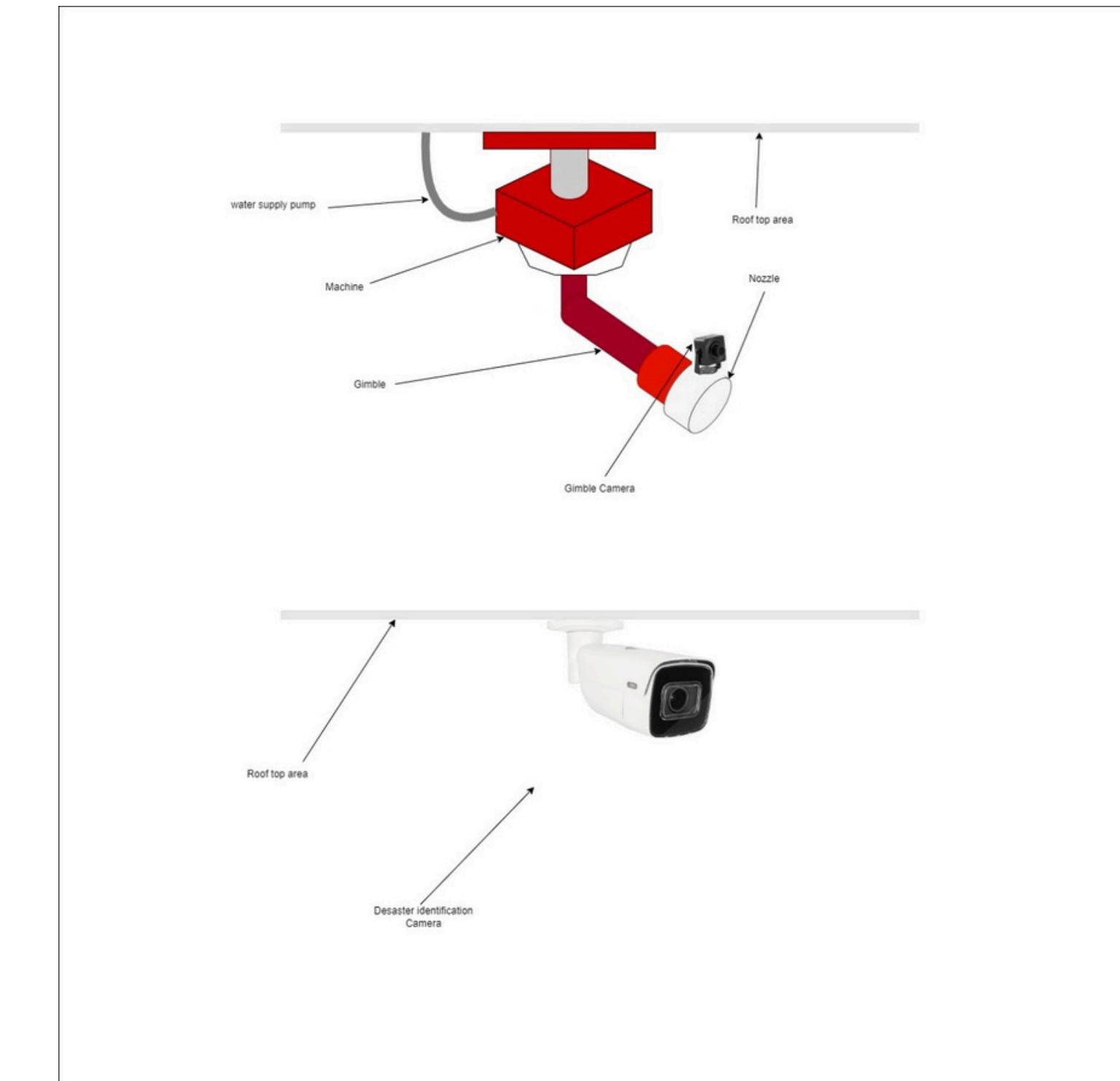
ML

Python

Arduino

MongoDB

IOT



Research Questions



How can Optimize
Decentralized
Communication
Protocol?



How to enhance
the Accuracy of
Hybrid Geolocation
System?



Way of Improving
Real-Time
Disaster
Identification.



Adaptive
Activation
Algorithm
Optimization



Integration and
Interoperability
Challenges.

Objectives



Developing Robust Communication Protocols



Enhancing Geolocation Systems



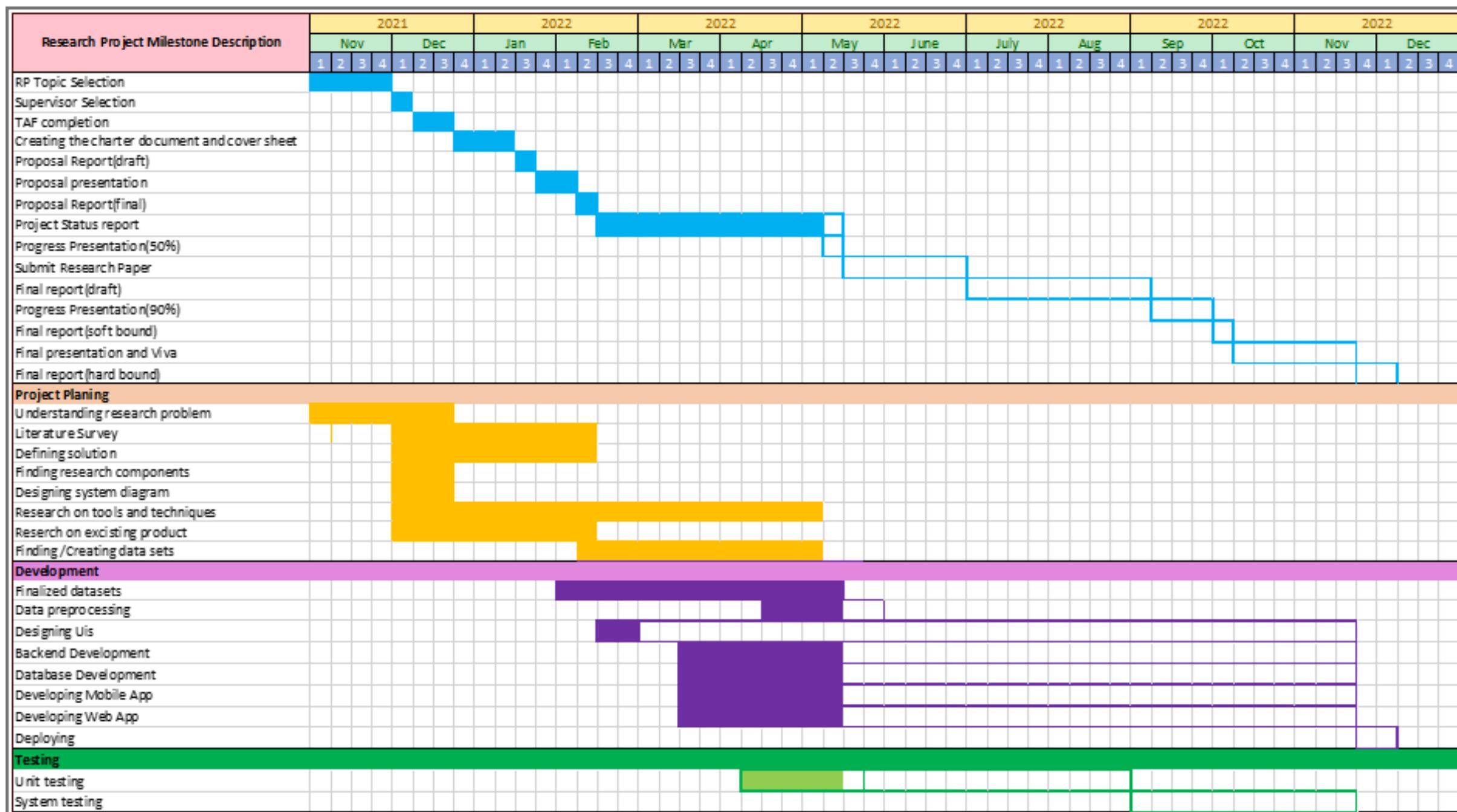
Implementing Disaster Detection Systems



Adapting Activation Algorithms

05

TIMELINE



Sachintha Gayashan W.K

IT20154462

Specialization : Information Technology

Communication system



BACKGROUND



In a fire suppression system, the efficient extinguishing of fires relies on accurate detection of the fire, followed by timely and precise water discharge from a nozzle. This process involves encoding the detection signal for transmission to an Arduino board, where it is decoded and used to control the nozzle's operation. The key challenge lies in determining the appropriate flow rate of water, the distance at which it should be sprayed, and the speed at which the nozzle should be operated to effectively extinguish the fire while conserving water and minimizing damage.



How many cubic meters of water per second should be released from the nozzle to extinguish the fire? It is calculated how far and how fast it should be pressed and how much the nozzle should be opened for that.

RESEARCH QUESTION

How many cubic meters of water per second should be released from the nozzle to extinguish the fire? It is calculated how far and how fast it should be pressed and how much the nozzle should be opened for that.



RESEARCH GAP

Despite advancements in fire detection and suppression technologies, there remains a significant gap in integrating real-time data analytics and adaptive control mechanisms into firefighting systems. Existing approaches often lack the capability to dynamically adjust water discharge rates, pressure levels, and nozzle configurations based on evolving fire dynamics and environmental factors. Bridging this gap requires innovative solutions that leverage data-driven insights to optimize firefighting efficiency, minimize resource wastage, and enhance overall fire safety.

SPECIFIC AND SUB OBJECTIVE

Develop an
Integrated
Fire Tracking
and Water
Discharge
System



Develop an Integrated Fire Tracking and Water Discharge System

SPECIFIC AND SUB OBJECTIVE

Sub objectives



Develop real-time fire detection and tracking algorithms for integration into the system.



Design an efficient data encoding method to compress tracked fire data for transmission and implement a decoding mechanism on the Arduino board.

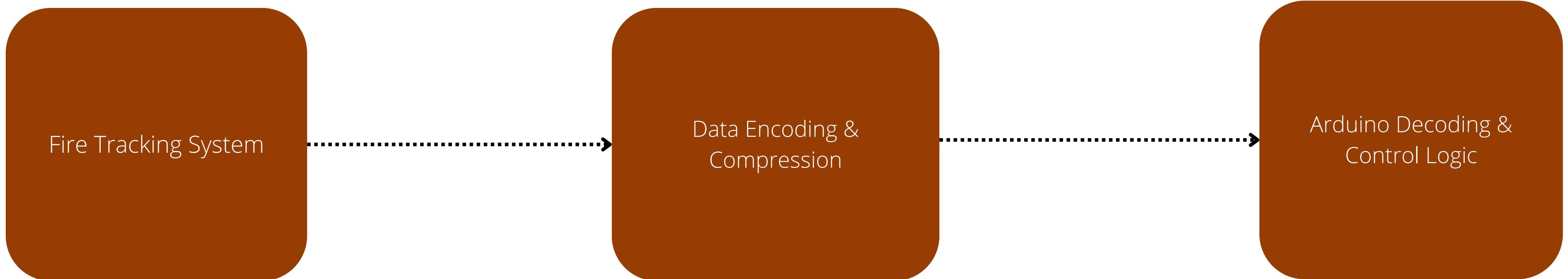


Conduct empirical studies to determine the required water flow rate based on fire size and intensity.

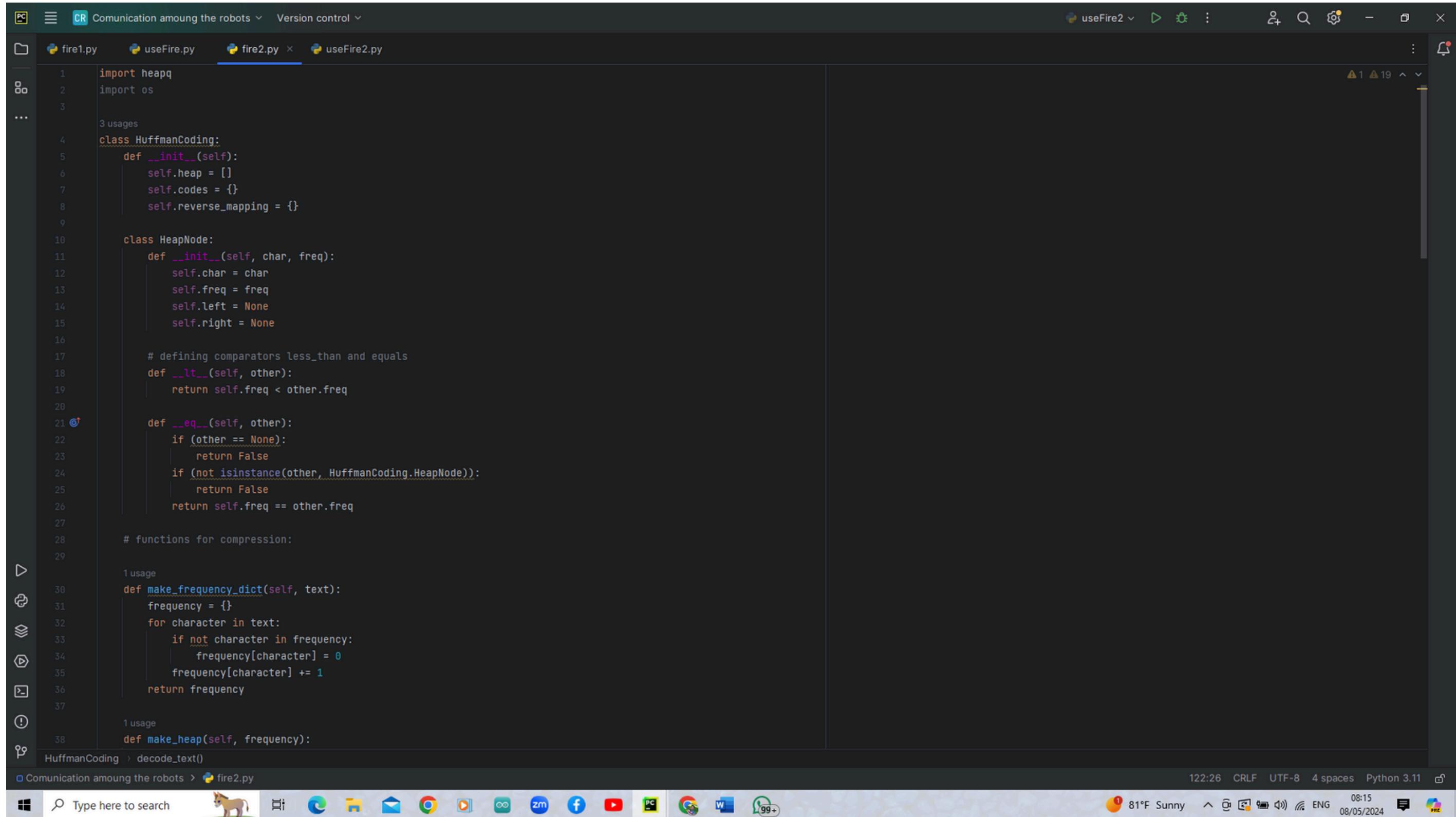


Develop algorithms to calculate optimal nozzle opening based on fire distance and size and implement control logic for adjusting nozzle pressure and angle.

SYSTEM DIAGRAM



The part that encodes the signal coming through the fire tracking system



```
import heapq
import os

3 usages
class HuffmanCoding:
    def __init__(self):
        self.heap = []
        self.codes = {}
        self.reverse_mapping = {}

    class HeapNode:
        def __init__(self, char, freq):
            self.char = char
            self.freq = freq
            self.left = None
            self.right = None

        # defining comparators less_than and equals
        def __lt__(self, other):
            return self.freq < other.freq

        def __eq__(self, other):
            if (other == None):
                return False
            if (not isinstance(other, HuffmanCoding.HeapNode)):
                return False
            return self.freq == other.freq

    # functions for compression:

    1 usage
    def make_frequency_dict(self, text):
        frequency = {}
        for character in text:
            if not character in frequency:
                frequency[character] = 0
            frequency[character] += 1
        return frequency

    1 usage
    def make_heap(self, frequency):
        HuffmanCoding > decode_text()

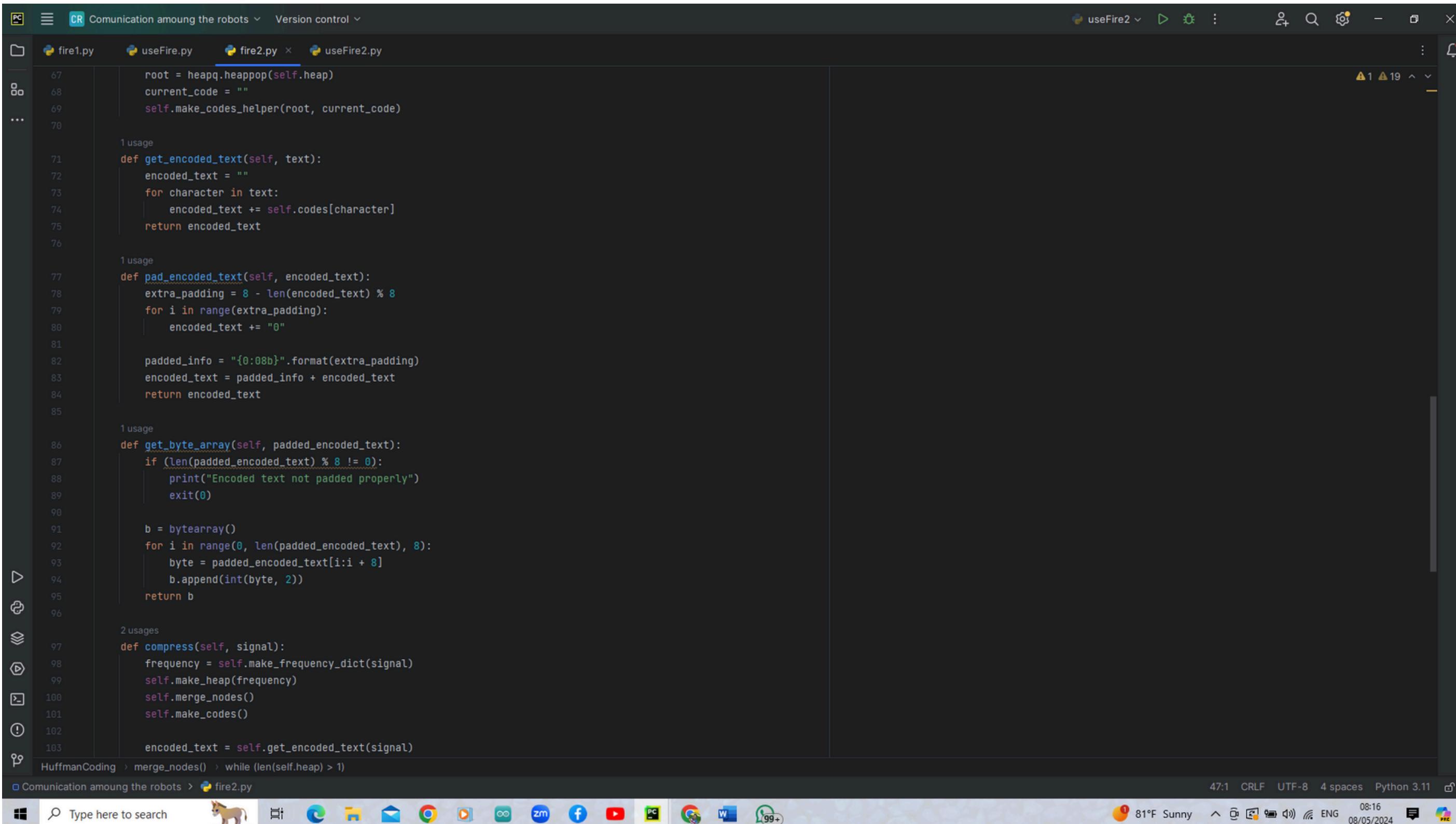
Communication among the robots > fire2.py
```

122:26 CRLF UTF-8 4 spaces Python 3.11

81°F Sunny 08:15 08/05/2024



The part that encodes the signal coming through the fire tracking system



```
fire1.py useFire.py fire2.py useFire2.py

67     root = heapq.heappop(self.heap)
68     current_code = ""
69     self.make_codes_helper(root, current_code)
...
70
71     1 usage
72     def get_encoded_text(self, text):
73         encoded_text = ""
74         for character in text:
75             encoded_text += self.codes[character]
76         return encoded_text
77
78     1 usage
79     def pad_encoded_text(self, encoded_text):
80         extra_padding = 8 - len(encoded_text) % 8
81         for i in range(extra_padding):
82             encoded_text += "0"
83
84         padded_info = "{0:08b}".format(extra_padding)
85         encoded_text = padded_info + encoded_text
86         return encoded_text
87
88     1 usage
89     def get_byte_array(self, padded_encoded_text):
90         if (len(padded_encoded_text) % 8 != 0):
91             print("Encoded text not padded properly")
92             exit(0)
93
94         b = bytearray()
95         for i in range(0, len(padded_encoded_text), 8):
96             byte = padded_encoded_text[i:i + 8]
97             b.append(int(byte, 2))
98
99         return b
100
101
102     2 usages
103     def compress(self, signal):
104         frequency = self.make_frequency_dict(signal)
105         self.make_heap(frequency)
106         self.merge_nodes()
107         self.make_codes()
108
109         encoded_text = self.get_encoded_text(signal)

HuffmanCoding > merge_nodes() > while (len(self.heap) > 1)

Communication among the robots > fire2.py

47:1 CRLF UTF-8 4 spaces Python 3.11
81°F Sunny 08:16 08/05/2024
```



The part that encodes the signal coming through the fire tracking system

```
35     frequency[character] += 1
36     return frequency
37
38     1 usage
39     def make_heap(self, frequency):
40         for key in frequency:
41             node = self.HeapNode(key, frequency[key])
42             heapq.heappush(*args: self.heap, node)
43
44         1 usage
45         def merge_nodes(self):
46             while (len(self.heap) > 1):
47                 node1 = heapq.heappop(self.heap)
48                 node2 = heapq.heappop(self.heap)
49
50                 merged = self.HeapNode( char: None, node1.freq + node2.freq)
51                 merged.left = node1
52                 merged.right = node2
53
54                 heapq.heappush(*args: self.heap, merged)
55
56             3 usages
57             def make_codes_helper(self, root, current_code):
58                 if (root == None):
59                     return
60
61                 if (root.char != None):
62                     self.codes[root.char] = current_code
63                     self.reverse_mapping[current_code] = root.char
64                     return
65
66                 self.make_codes_helper(root.left, current_code + "0")
67                 self.make_codes_helper(root.right, current_code + "1")
68
69             1 usage
70             def make_codes(self):
71                 root = heapq.heappop(self.heap)
72                 current_code = ""
73                 self.make_codes_helper(root, current_code)
74
75             1 usage
```

HuffmanCoding > merge_nodes() > while (len(self.heap) > 1)

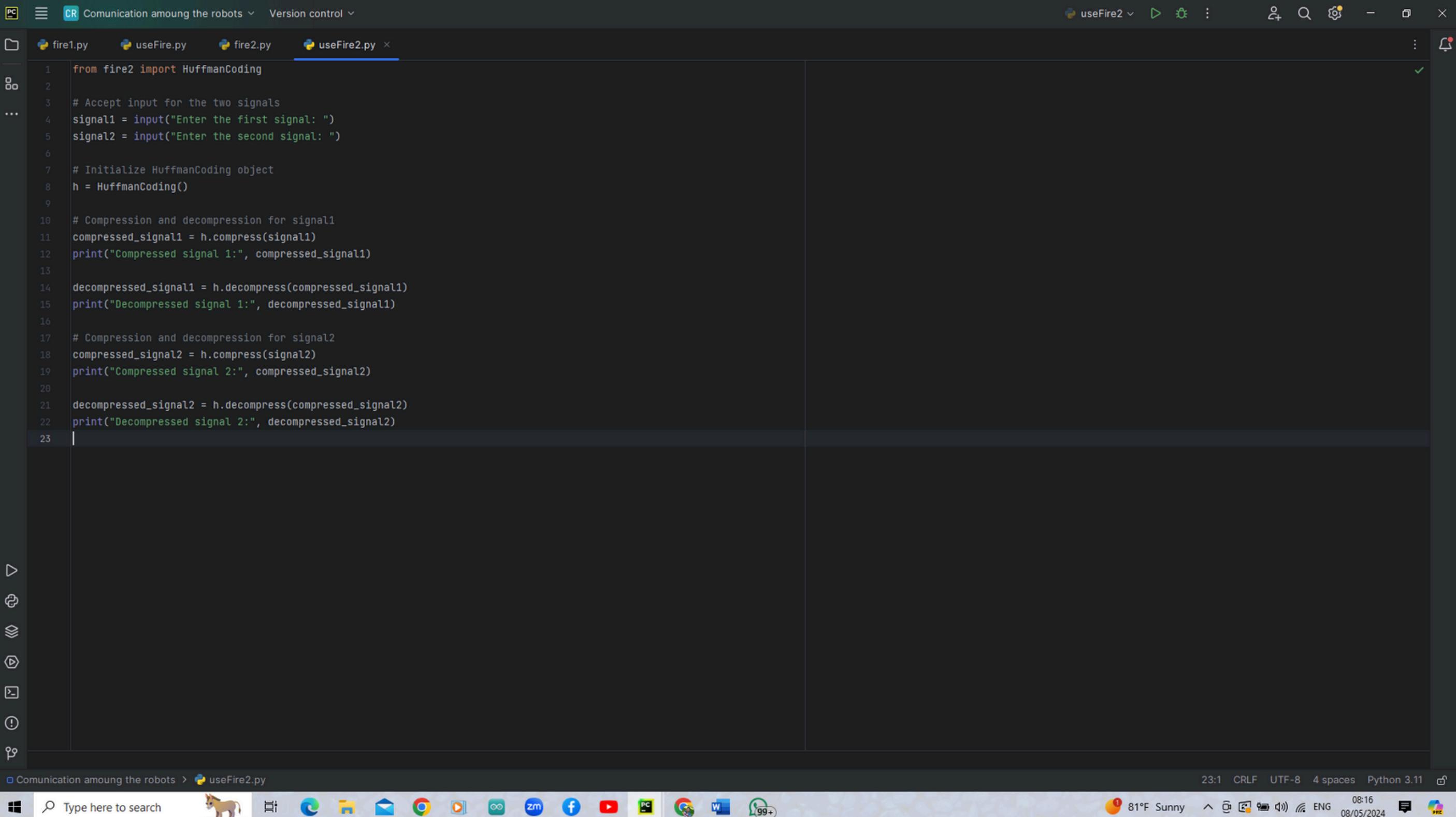
Communication among the robots > fire2.py

47:1 CRLF UTF-8 4 spaces Python 3.11

81°F Sunny 08:16 08/05/2024



The part that encodes the signal coming through the fire tracking system



```
from fire2 import HuffmanCoding

# Accept input for the two signals
signal1 = input("Enter the first signal: ")
signal2 = input("Enter the second signal: ")

# Initialize HuffmanCoding object
h = HuffmanCoding()

# Compression and decompression for signal1
compressed_signal1 = h.compress(signal1)
print("Compressed signal 1:", compressed_signal1)

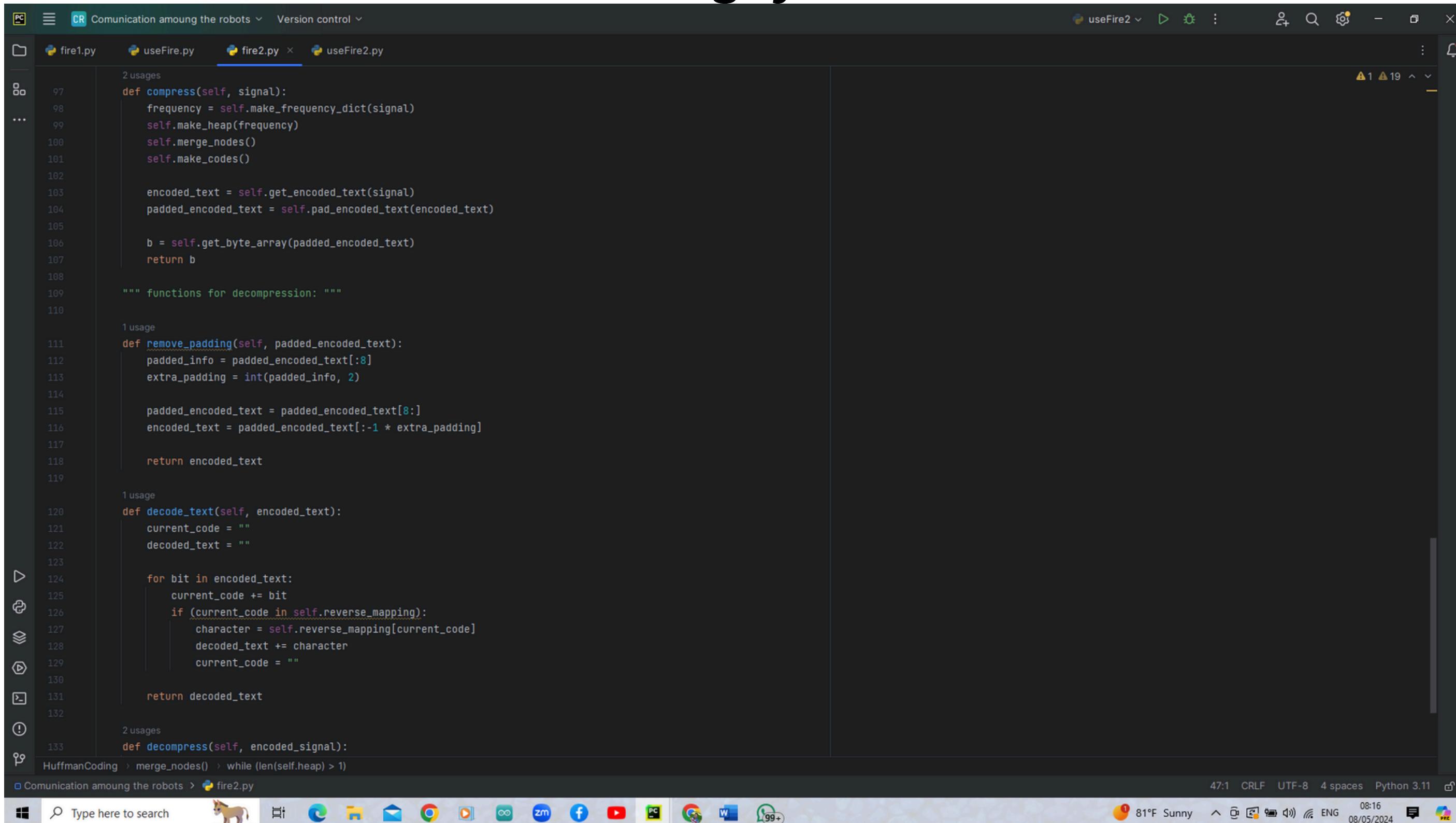
decompressed_signal1 = h.decompress(compressed_signal1)
print("Decompressed signal 1:", decompressed_signal1)

# Compression and decompression for signal2
compressed_signal2 = h.compress(signal2)
print("Compressed signal 2:", compressed_signal2)

decompressed_signal2 = h.decompress(compressed_signal2)
print("Decompressed signal 2:", decompressed_signal2)
```



The part that encodes the signal coming through the fire tracking system



```
PC Communication among the robots Version control
fire1.py useFire.py fire2.py useFire2.py
2 usages
97     def compress(self, signal):
98         frequency = self.make_frequency_dict(signal)
99         self.make_heap(frequency)
100        self.merge_nodes()
101        self.make_codes()
102
103        encoded_text = self.get_encoded_text(signal)
104        padded_encoded_text = self.pad_encoded_text(encoded_text)
105
106        b = self.get_byte_array(padded_encoded_text)
107        return b
108
109    """ functions for decompression: """
110
111    1 usage
112    def remove_padding(self, padded_encoded_text):
113        padded_info = padded_encoded_text[:8]
114        extra_padding = int(padded_info, 2)
115
116        padded_encoded_text = padded_encoded_text[8:]
117        encoded_text = padded_encoded_text[:-1 * extra_padding]
118
119        return encoded_text
120
121    1 usage
122    def decode_text(self, encoded_text):
123        current_code = ""
124        decoded_text = ""
125
126        for bit in encoded_text:
127            current_code += bit
128            if (current_code in self.reverse_mapping):
129                character = self.reverse_mapping[current_code]
130                decoded_text += character
131                current_code = ""
132
133    2 usages
134    def decompress(self, encoded_signal):
HuffmanCoding > merge_nodes() > while (len(self.heap) > 1)
Communication among the robots > fire2.py
47:1 CRLF UTF-8 4 spaces Python 3.11
Type here to search 81°F Sunny 08:16 ENG 08/05/2024
```

The part that encodes the signal coming through the fire tracking system

```
106     b = self.get_byte_array(padded_encoded_text)
107     return b
108
109     """ functions for decompression: """
110
111     1 usage
112     def remove_padding(self, padded_encoded_text):
113         padded_info = padded_encoded_text[:8]
114         extra_padding = int(padded_info, 2)
115
116         padded_encoded_text = padded_encoded_text[8:]
117         encoded_text = padded_encoded_text[:-1 * extra_padding]
118
119         return encoded_text
120
121     1 usage
122     def decode_text(self, encoded_text):
123         current_code = ""
124         decoded_text = ""
125
126         for bit in encoded_text:
127             current_code += bit
128             if (current_code in self.reverse_mapping):
129                 character = self.reverse_mapping[current_code]
130                 decoded_text += character
131                 current_code = ""
132
133         return decoded_text
134
135     2 usages
136     def decompress(self, encoded_signal):
137         bit_string = "".join(format(byte, '08b') for byte in encoded_signal)
138         encoded_text = self.remove_padding(bit_string)
139         decompressed_text = self.decode_text(encoded_text)
140
141         return decompressed_text
```



The output of the part that encodes the signal coming through the fire tracking system

The screenshot shows a Python development environment with the following details:

- Project Explorer:** Shows a folder named "Communication among the robots" containing several Python files: com.py, fire1.py, fire2.py, mainGestures.py, mainGestures2.py, test.py, useFire.py, useFire2.py, utilis2.py, utilis1.py, and utilis.py.
- Code Editor:** The file "useFire2.py" is open, displaying the following code:

```
from fire2 import HuffmanCoding
# Accept input for the two signals
signal1 = input("Enter the first signal: ")
signal2 = input("Enter the second signal: ")

# Initialize HuffmanCoding object
h = HuffmanCoding()

# Compression and decompression for signal1
compressed_signal1 = h.compress(signal1)
print("Compressed signal 1:", compressed_signal1)

decompressed_signal1 = h.decompress(compressed_signal1)
print("Decompressed signal 1:", decompressed_signal1)

# Compression and decompression for signal2
compressed_signal2 = h.compress(signal2)
print("Compressed signal 2:", compressed_signal2)

decompressed_signal2 = h.decompress(compressed_signal2)
print("Decompressed signal 2:", decompressed_signal2)
```

- Run Tab:** The tab shows the command: C:\Users\lap\AppData\Local\Programs\Python\Python311\python.exe "E:\4 year semester 1(IS My Supper Semester)\Research\Research topic with deshan\Communication among the robots\useFire2.py". The output window displays the following results:

```
Enter the first signal: PAN 34
Enter the second signal: TILT23
Compressed signal 1: bytearray(b'\x08\xc7l\x00')
Decompressed signal 1: PAN 34
Compressed signal 2: bytearray(b'\x02\xe5\xd0')
Decompressed signal 2: TILT23

Process finished with exit code 0
```

- System Tray:** Shows the date (08/05/2024), time (08:14), weather (81°F Sunny), and battery status.

Nozzle on off Part

```
1 usage
2 def calculate_flow_rate(fire_size, fuel_type, extinguishing_time):
3     # Example calculation based on assumptions (actual calculations may vary)
4     # Assume fire_size is in square meters, fuel_type is a categorical variable (e.g., "wood", "paper", "electrical", etc.)
5     # extinguishing_time is in seconds
6
7     # Constants for flow rate calculation (values are assumptions and can vary based on actual conditions)
8     flow_rate_constant = 0.05 # Cubic meters per square meter per second
9     distance_constant = 2 # Meters
10
11     # Calculate flow rate based on fire size and fuel type
12     flow_rate = fire_size * flow_rate_constant
13
14     # Calculate spraying distance based on extinguishing time
15     spraying_distance = extinguishing_time * distance_constant
16
17     return flow_rate, spraying_distance
18
19
20 usage
21 def turn_device_on_off(flow_rate, threshold):
22     # Example implementation for turning the device on/off based on flow rate and a threshold value
23     if flow_rate >= threshold:
24         return "Device turned on"
25     else:
26         return "Device turned off"
27
28
29 # Input values
30 fire_size = float(input("Enter the size of the fire in square meters: "))
31 fuel_type = input("Enter the type of fire (e.g., small,medium,large): ")
32 extinguishing_time = float(input("Enter the desired extinguishing time in seconds: "))
33 threshold = float(input("Enter the threshold flow rate in cubic meters per second: "))
34
35
36 # Calculate flow rate and spraying distance
37 flow_rate, spraying_distance = calculate_flow_rate(fire_size, fuel_type, extinguishing_time)
38
39
40 # Output results
41 print("Flow rate:", flow_rate, "cubic meters per second")
42 print("Spraying distance:", spraying_distance, "meters")
```



Nozzle on off part

```
nozzle.py
9
10     # Calculate flow rate based on fire size and fuel type
11     flow_rate = fire_size * flow_rate_constant
12
13     # Calculate spraying distance based on extinguishing time
14     spraying_distance = extinguishing_time * distance_constant
15
16     return flow_rate, spraying_distance
17
18
19 usage
20 def turn_device_on_off(flow_rate, threshold):
21     # Example implementation for turning the device on/off based on flow rate and a threshold value
22     if flow_rate >= threshold:
23         return "Device turned on"
24     else:
25         return "Device turned off"
26
27
28 # Input values
29 fire_size = float(input("Enter the size of the fire in square meters: "))
30 fuel_type = input("Enter the type of fire (e.g., small,medium,large): ")
31 extinguishing_time = float(input("Enter the desired extinguishing time in seconds: "))
32 threshold = float(input("Enter the threshold flow rate in cubic meters per second: "))
33
34 # Calculate flow rate and spraying distance
35 flow_rate, spraying_distance = calculate_flow_rate(fire_size, fuel_type, extinguishing_time)
36
37 # Output results
38 print("Flow rate:", flow_rate, "cubic meters per second")
39 print("Spraying distance:", spraying_distance, "meters")
40
41 # Turn device on/off based on flow rate and threshold
42 device_state = turn_device_on_off(flow_rate, threshold)
43
44
45 calculate_flow_rate()
46
47 Communication among the robots > nozzle.py
48
49 14:44 CRLF UTF-8 4 spaces Python 3.11
50
51 83°F Mostly sunny 09:35 ENG 08/05/2024
```



Nozzle on off part

The screenshot shows a Windows desktop environment with a Visual Studio Code (VS Code) window open. The title bar of the VS Code window reads "Communication amoung the robots". The left sidebar shows a project structure with files like com.py, fire1.py, fire2.py, mainGestures.py, mainGestures2.py, test.py, useFire.py, useFire2.py, utilis2.py, utilis1.py, and utilis.py. The main editor tab is "useFire2.py". The code in "useFire2.py" is as follows:

```
from fire2 import HuffmanCoding
# Accept input for the two signals
signal1 = input("Enter the first signal: ")
signal2 = input("Enter the second signal: ")

# Initialize HuffmanCoding object
h = HuffmanCoding()

# Compression and decompression for signal1
compressed_signal1 = h.compress(signal1)
print("Compressed signal 1:", compressed_signal1)

decompressed_signal1 = h.decompress(compressed_signal1)
print("Decompressed signal 1:", decompressed_signal1)

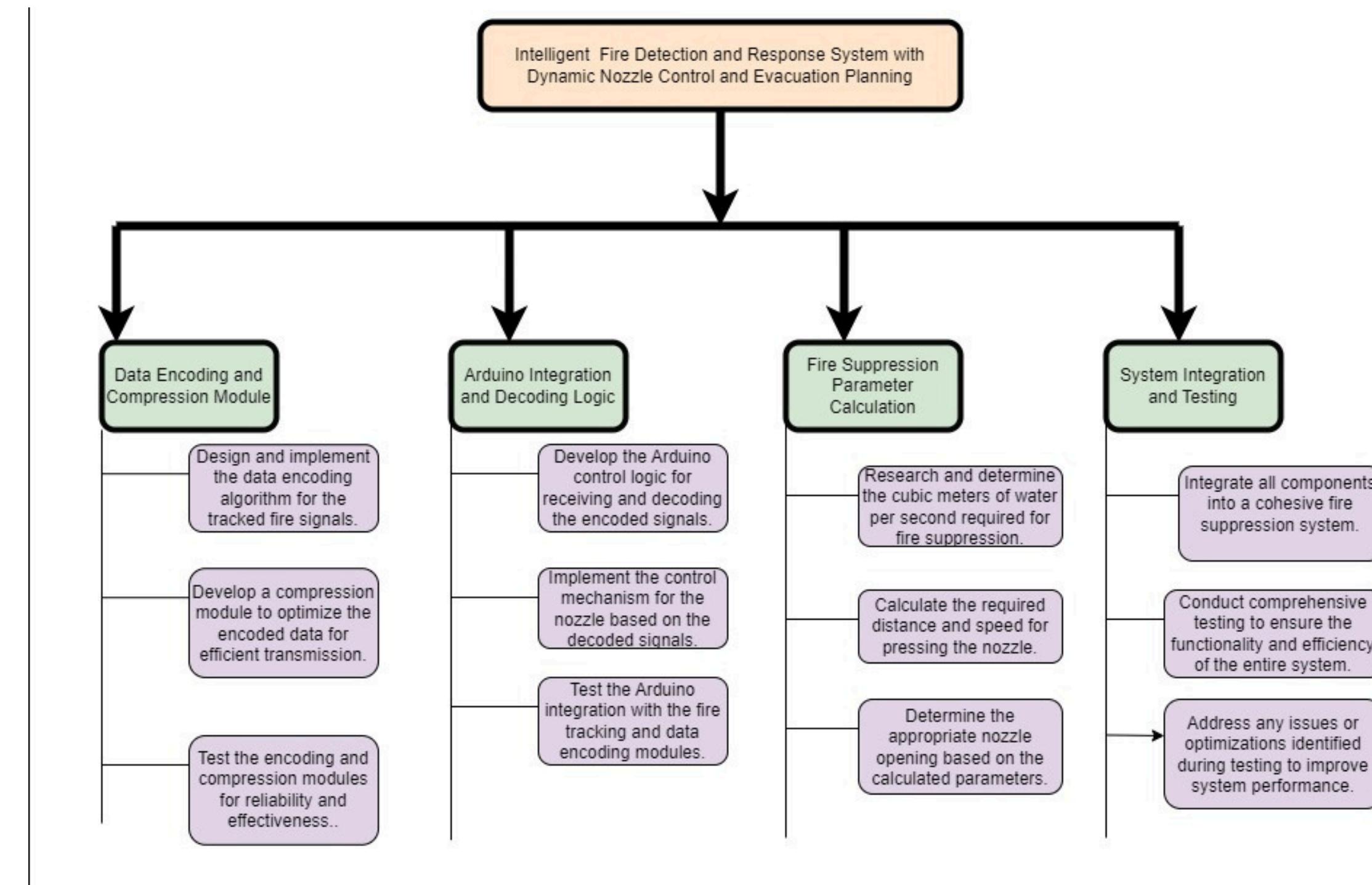
# Compression and decompression for signal2
compressed_signal2 = h.compress(signal2)
print("Compressed signal 2:", compressed_signal2)

decompressed_signal2 = h.decompress(compressed_signal2)
print("Decompressed signal 2:", decompressed_signal2)
```

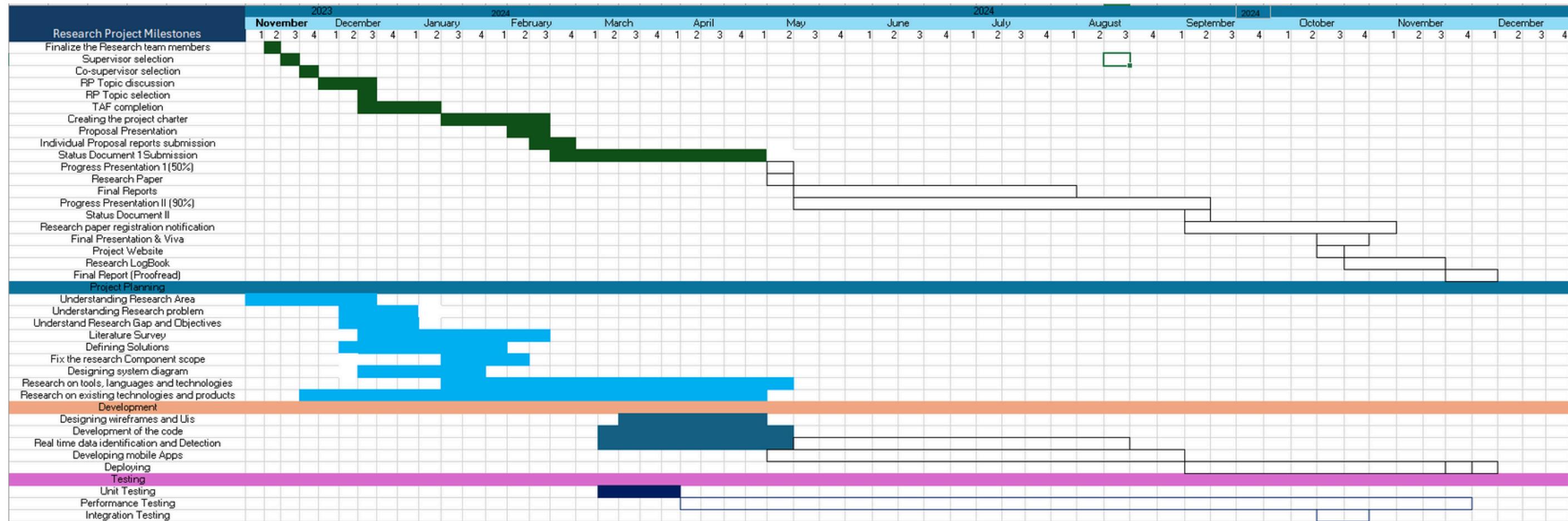
The "Run" tab at the bottom shows the command "C:\Users\lap\AppData\Local\Programs\Python\Python311\python.exe "E:\4 year semester 1(IS My Supper Semester)\Research\Research topic with deshan\Communication amoung the robots\useFire2.py"" and the output of the program. The output shows the user entering "PAN 34" and "TILT23" as signals, followed by the compressed and decompressed versions of these signals.



Work Break Down Structure



Grant Chart



Project Completion

Completed Tasks



- ✓ Implemented the fire tracking system to track the fire.
- ✓ Developed the data encoding and compression module to encode the incoming signal.
- ✓ Integrated the Arduino decoding and control logic to decode the received signal and control the nozzle.
- ✓ Determine the cubic meters of water per second required to extinguish the fire

Tasks to be completed

-  Calculate the required distance and speed to press the nozzle
-  Determine the nozzle opening based on the calculated parameters.
-  Test the entire system for functionality and efficiency.

References

-  Daniele Calisi, A. F. (2007, September 07). Multi-objective exploration and search for autonomous rescue robots. Retrieved from <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20216>
-  Flushing, E. F., Gambardella, L. M., & Caro, G. A. (2016, December 15). On Using Mobile Robotic Relays for Adaptive Communication in Search and Rescue Missions. Retrieved from <https://ieeexplore.ieee.org/document/7784329>
-  Jason Gregory, J. F. (2016, March 16). Application of Multi-Robot Systems to Disaster-Relief Scenarios with Limited Communication. Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-27702-8_42
-  JORGE PEÑA QUERALTA 1, (. S. (2020). Collaborative Multi-Robot Search and Rescue: Planning, Coordination, Perception, and Active Vision. 27. Retrieved from <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9220149>

Manorathna A.Y.S

IT21086670

Specialization : Information Technology

Fire Tracking System



BACKGROUND



The background involves the development of a fire tracking system that utilizes a camera to detect and track fires. The system's main objective is to transmit tracking data, including pan and tilt information, to the communication part of the overall system.

Additionally, insights into the performance of the fire detection process, such as the time taken for different stages and the average processing speed per image, are provided.

As a future enhancement, expanding the system's capabilities to calculate the area covered by the fire and the time taken for its spread is planned. This background emphasizes the importance of real-time fire monitoring and data transmission for effective fire management and mitigation.

RESEARCH QUESTION

How can a fire tracking system utilizing camera technology effectively track, and transmit real-time data about fires, including pan and tilt information, and provide insights into the performance of the fire detection process for efficient fire management and mitigation?



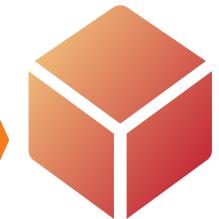
RESEARCH GAP

The research gap for your part could be the lack of a comprehensive fire tracking system that integrates camera technology to detect, track, and transmit real-time data about fires, including pan and tilt information, while also providing insights into the performance of the fire detection process. Additionally, there may be a gap in the existing systems in terms of their ability to calculate the area covered by the fire and the time taken for its spread, highlighting the need for further development in this area.



SPECIFIC AND SUB OBJECTIVES

**Specific
Objective**



To develop a fire tracking system that utilizes a camera to track fires, and to transmit the tracking data to the communication part of the system.

SPECIFIC AND SUB OBJECTIVES

Sub objectives



Develop algorithms for fire detection and continuous tracking using camera feed.



Gather tracking data and analyze system performance metrics.

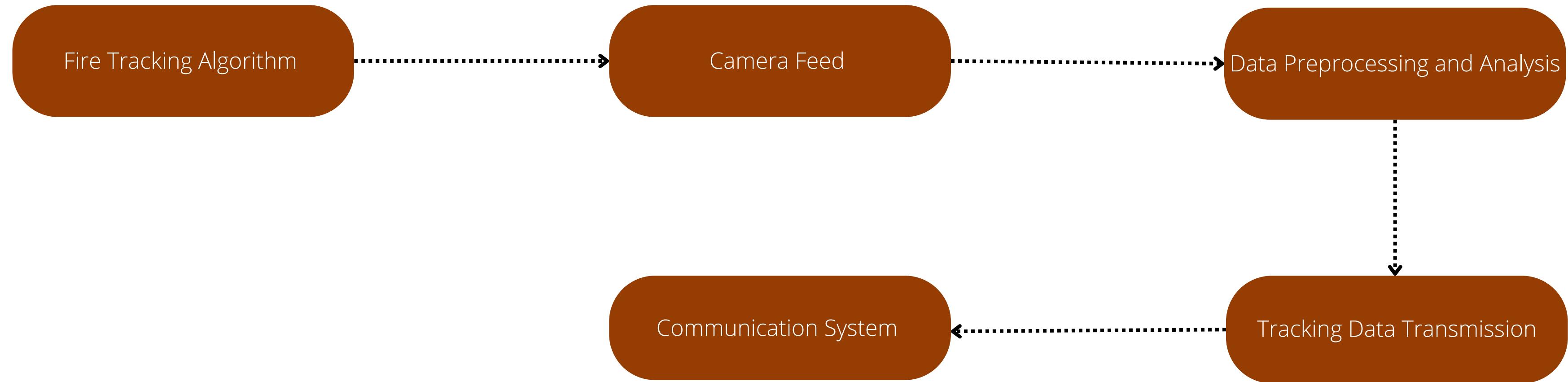


Evaluate system accuracy and efficiency under varied conditions and optimize as needed.



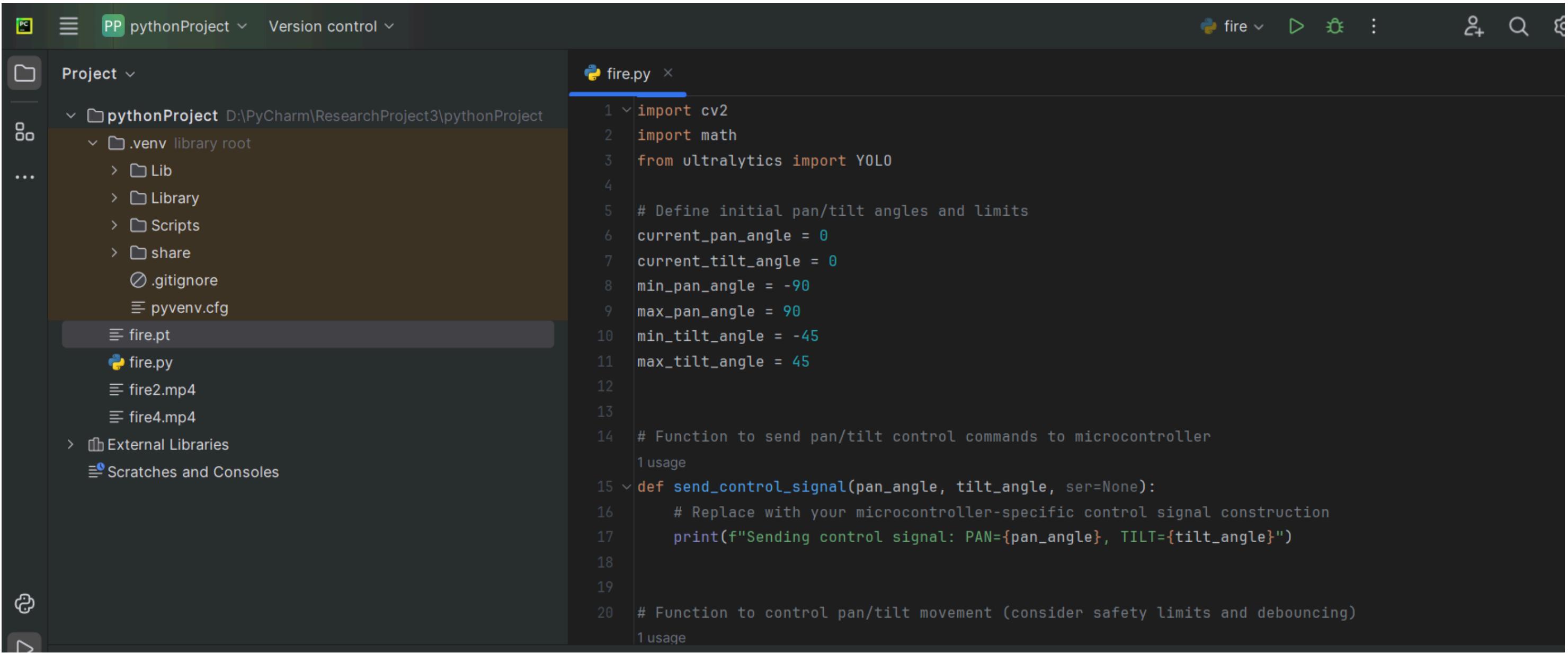
Develop algorithms to measure fire spread area and rate over time, analyzing influencing factors.

SYSTEM DIAGRAM



Project Evidence

Initiation setup and functions definitions for camera Control

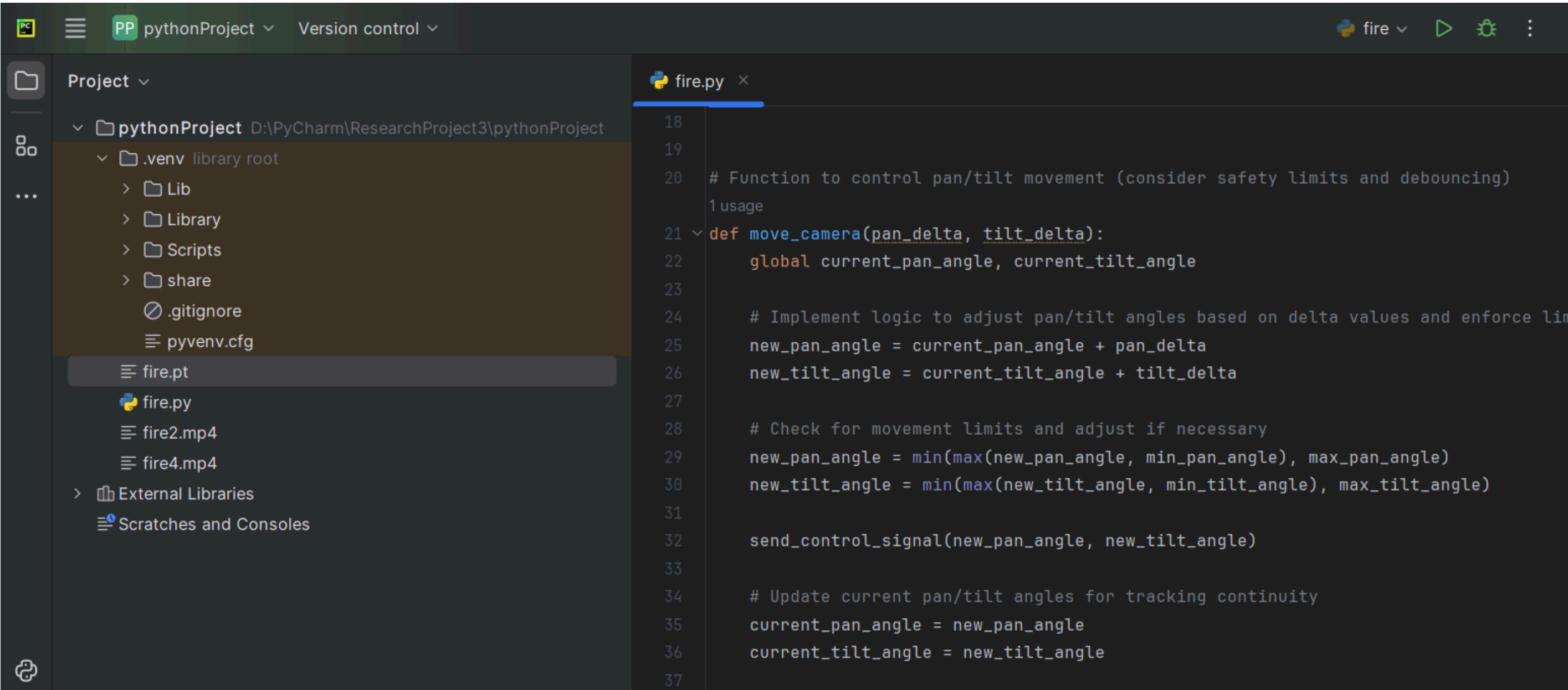


```
fire.py
1 import cv2
2 import math
3 from ultralytics import YOLO
4
5 # Define initial pan/tilt angles and limits
6 current_pan_angle = 0
7 current_tilt_angle = 0
8 min_pan_angle = -90
9 max_pan_angle = 90
10 min_tilt_angle = -45
11 max_tilt_angle = 45
12
13
14 # Function to send pan/tilt control commands to microcontroller
15 def send_control_signal(pan_angle, tilt_angle, ser=None):
16     # Replace with your microcontroller-specific control signal construction
17     print(f"Sending control signal: PAN={pan_angle}, TILT={tilt_angle}")
18
19
20 # Function to control pan/tilt movement (consider safety limits and debouncing)
21 usage
```



Project Evidence

Initiation setup and functions definitions for camera Control

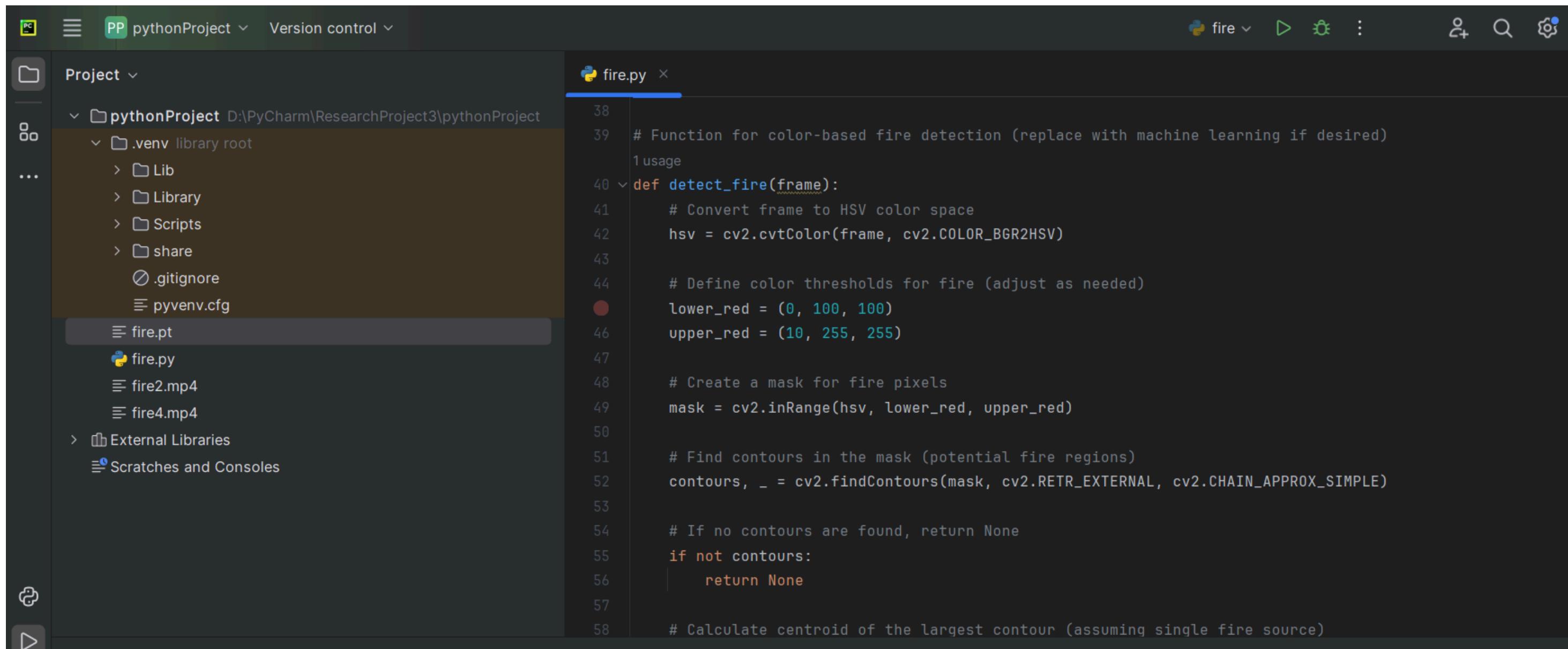


```
18
19
20 # Function to control pan/tilt movement (consider safety limits and debouncing)
1 usage
21 def move_camera(pan_delta, tilt_delta):
22     global current_pan_angle, current_tilt_angle
23
24     # Implement logic to adjust pan/tilt angles based on delta values and enforce lim
25     new_pan_angle = current_pan_angle + pan_delta
26     new_tilt_angle = current_tilt_angle + tilt_delta
27
28     # Check for movement limits and adjust if necessary
29     new_pan_angle = min(max(new_pan_angle, min_pan_angle), max_pan_angle)
30     new_tilt_angle = min(max(new_tilt_angle, min_tilt_angle), max_tilt_angle)
31
32     send_control_signal(new_pan_angle, new_tilt_angle)
33
34     # Update current pan/tilt angles for tracking continuity
35     current_pan_angle = new_pan_angle
36     current_tilt_angle = new_tilt_angle
37
```



Project Evidence

Fire Detection Function



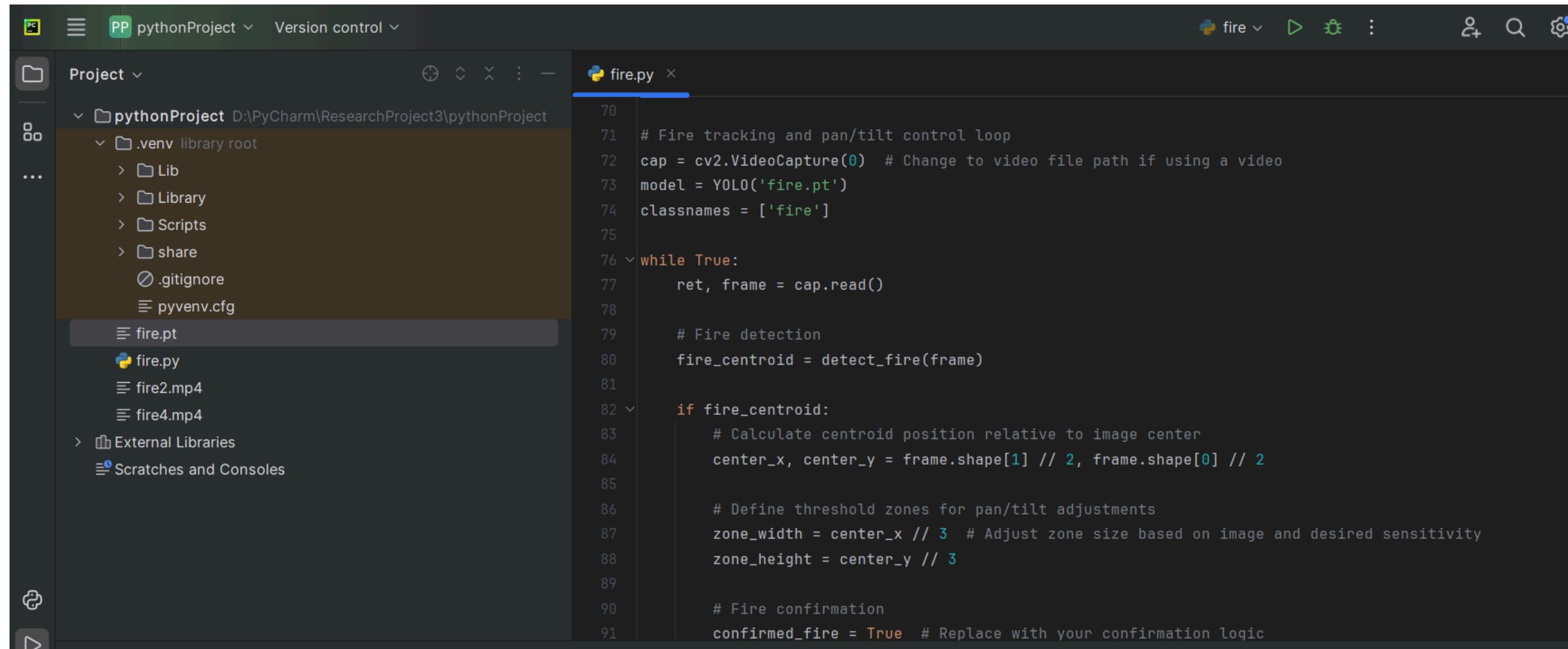
The screenshot shows the PyCharm IDE interface. The top bar displays the project name "pythonProject" and the file "fire.py". The left sidebar shows the project structure with files like .gitignore, pyvenv.cfg, fire.pt, fire.py, fire2.mp4, fire4.mp4, External Libraries, and Scratches and Consoles. The main editor window displays the following Python code:

```
38 # Function for color-based fire detection (replace with machine learning if desired)
39 # usage
40 def detect_fire(frame):
41     # Convert frame to HSV color space
42     hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
43
44     # Define color thresholds for fire (adjust as needed)
45     lower_red = (0, 100, 100)
46     upper_red = (10, 255, 255)
47
48     # Create a mask for fire pixels
49     mask = cv2.inRange(hsv, lower_red, upper_red)
50
51     # Find contours in the mask (potential fire regions)
52     contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
53
54     # If no contours are found, return None
55     if not contours:
56         return None
57
58     # Calculate centroid of the largest contour (assuming single fire source)
```



Project Evidence

Main Loop for Fire Tracking and Fire Detection Model (YOLO) Integration



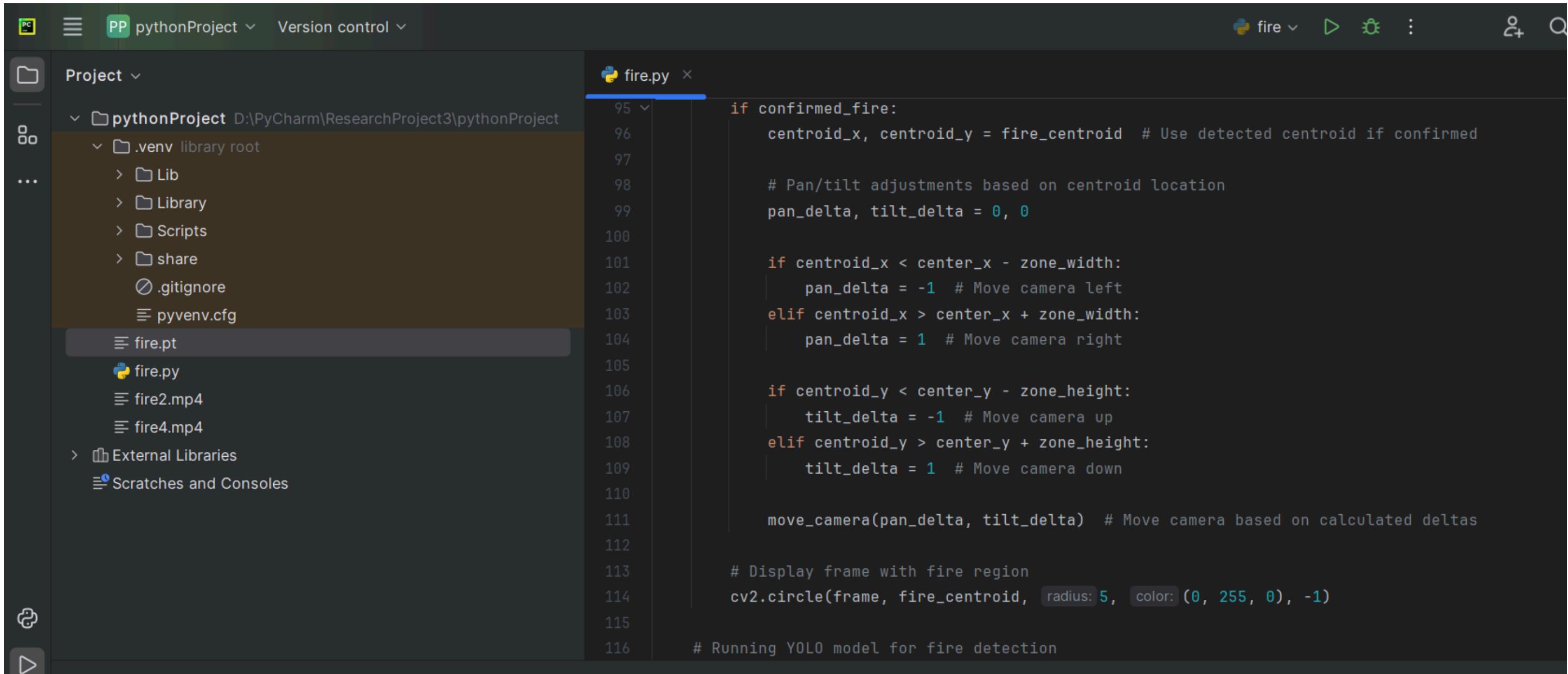
The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'pythonProject'. It includes a '.venv' folder containing 'Lib', 'Library', 'Scripts', 'share', '.gitignore', and 'pyvenv.cfg'. A file named 'fire.pt' is currently selected. The main editor window on the right contains the 'fire.py' script:

```
70 # Fire tracking and pan/tilt control loop
71 cap = cv2.VideoCapture(0) # Change to video file path if using a video
72 model = YOLO('fire.pt')
73 classnames = ['fire']
74
75 while True:
76     ret, frame = cap.read()
77
78     # Fire detection
79     fire_centroid = detect_fire(frame)
80
81     if fire_centroid:
82         # Calculate centroid position relative to image center
83         center_x, center_y = frame.shape[1] // 2, frame.shape[0] // 2
84
85         # Define threshold zones for pan/tilt adjustments
86         zone_width = center_x // 3 # Adjust zone size based on image and desired sensitivity
87         zone_height = center_y // 3
88
89         # Fire confirmation
90         confirmed_fire = True # Replace with your confirmation logic
```



Project Evidence

Main Loop for Fire Tracking and Fire Detection Model (YOLO)



The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'pythonProject'. Inside the 'pythonProject' folder, there is a '.venv' folder containing 'Lib', 'Library', 'Scripts', and 'share', along with files '.gitignore' and 'pyvenv.cfg'. Below these are files 'fire.pt', 'fire.py', 'fire2.mp4', and 'fire4.mp4'. There are also 'External Libraries' and 'Scratches and Consoles' sections. The right pane shows the code editor for 'fire.py'. The code is a Python script for fire tracking and detection using YOLO. It includes logic for centroid calculations, pan/tilt adjustments, camera movement, and frame display.

```
if confirmed_fire:
    centroid_x, centroid_y = fire_centroid # Use detected centroid if confirmed

    # Pan/tilt adjustments based on centroid location
    pan_delta, tilt_delta = 0, 0

    if centroid_x < center_x - zone_width:
        pan_delta = -1 # Move camera left
    elif centroid_x > center_x + zone_width:
        pan_delta = 1 # Move camera right

    if centroid_y < center_y - zone_height:
        tilt_delta = -1 # Move camera up
    elif centroid_y > center_y + zone_height:
        tilt_delta = 1 # Move camera down

    move_camera(pan_delta, tilt_delta) # Move camera based on calculated deltas

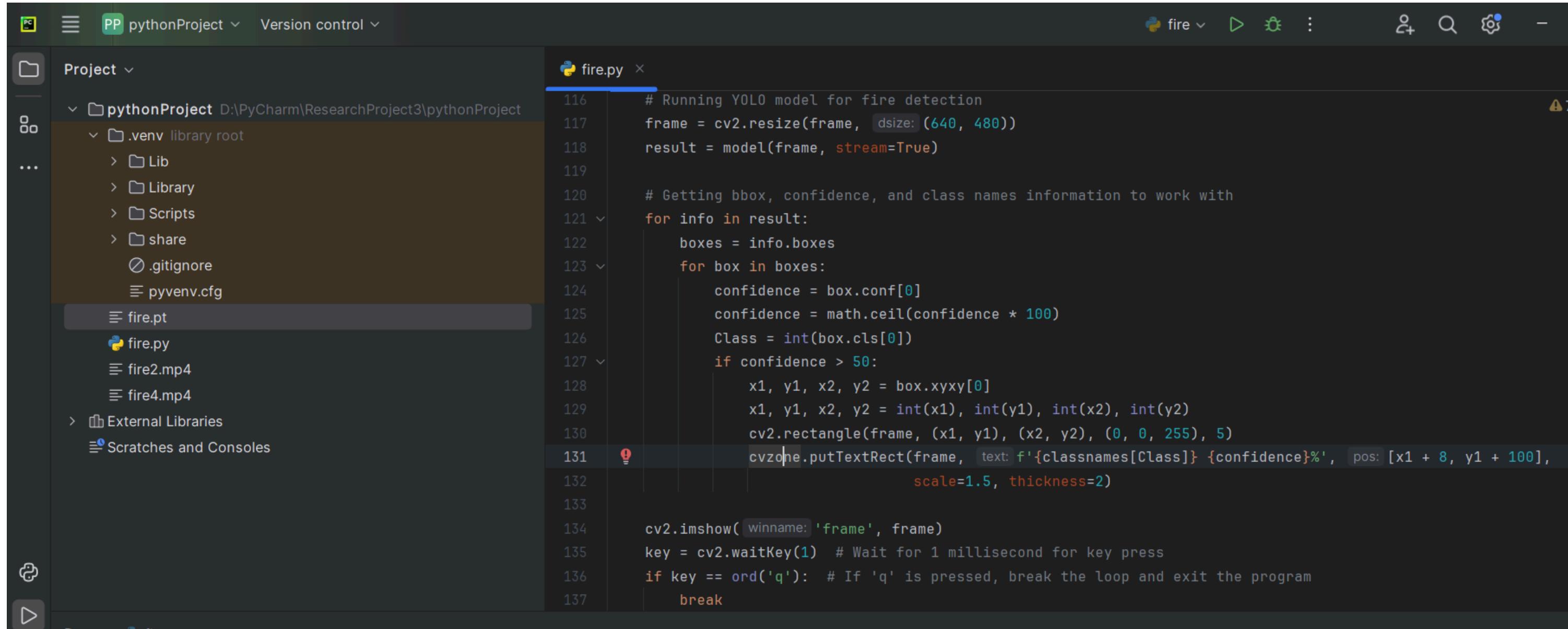
    # Display frame with fire region
    cv2.circle(frame, fire_centroid, radius: 5, color: (0, 255, 0), -1)

# Running YOLO model for fire detection
```



Project Evidence

Main Loop for Fire Tracking and Fire Detection Model (YOLO)



The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'pythonProject'. It includes a '.venv' folder containing 'Lib', 'Library', 'Scripts', and 'share', along with files like '.gitignore', 'pyenv.cfg', 'fire.pt', 'fire.py', 'fire2.mp4', 'fire4.mp4', and 'External Libraries'. The right pane shows the code editor for 'fire.py'. The code implements a main loop for fire detection using a YOLO model. It resizes the frame, processes it through the model, and then iterates over the results to draw bounding boxes and class names onto the frame. It also handles user input to exit the loop.

```
# Running YOLO model for fire detection
frame = cv2.resize(frame, dsiz=(640, 480))
result = model(frame, stream=True)

# Getting bbox, confidence, and class names information to work with
for info in result:
    boxes = info.bboxes
    for box in boxes:
        confidence = box.conf[0]
        confidence = math.ceil(confidence * 100)
        Class = int(box.cls[0])
        if confidence > 50:
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 5)
            cvzone.putTextRect(frame, f'{classnames[Class]} {confidence}%', pos=[x1 + 8, y1 + 100], scale=1.5, thickness=2)

cv2.imshow(winname='frame', frame)
key = cv2.waitKey(1) # Wait for 1 millisecond for key press
if key == ord('q'): # If 'q' is pressed, break the loop and exit the program
    break
```



Project Evidence

Model Development

The screenshot shows a Google Colab notebook titled "Untitled6.ipynb". The interface includes a sidebar with "Files" and a toolbar with "Comment", "Share", and "Colab AI".

In the main area, there are two code cells:

- Cell [2] contains the command `!nvidia-smi` which outputs GPU usage information for a Tesla T4 GPU.
- Cell [4] contains the command `!pip install ultralytics` which installs the Ultralytics library.

To the right of the code cells is a figure titled "F1-Confidence Curve". The x-axis is labeled "Confidence" and ranges from 0.0 to 1.0. The y-axis is labeled "F1" and ranges from 0.0 to 1.0. The plot shows two curves: a blue line for "fire" and a red line for "all classes". A legend indicates "fire" is at 0.47 at 0.283 and "all classes" is at 0.47 at 0.283.

Project Evidence

Model Development

The screenshot shows a Google Colab interface with the following details:

- Title:** Untitled6.ipynb
- File Menu:** File, Edit, View, Insert, Runtime, Tools, Help (with a note: Saving failed since 7:08PM)
- Runtime Status:** Connecting
- Colab AI:** Colab AI
- Code Cell:** !pip install ultralytics
- Output:** A long list of dependency installation messages, mostly showing requirements already satisfied.
- Visualizations:** Three files are visible in the sidebar: 2.jpg, F1_curve.png, and confusion_matrix.png. The confusion_matrix.png is selected and displayed as a heatmap titled "Confusion Matrix". The x-axis is labeled "True" with categories "fire" and "background". The y-axis is labeled "Predicted" with categories "fire" and "background". The color scale ranges from 0.0 (light blue) to 20.0 (dark blue). The matrix values are approximately:

True\Predicted	fire	background
fire	11	20
background	8	11

Project Evidence

Model Development

The screenshot shows a Google Colab notebook titled "Untitled6.ipynb". The code cell contains Python code for training a YOLO model using Ultralytics:

```
[ ] from ultralytics import YOLO
[ ] !yolo task=detect mode=train model=yolov8n.pt data=/content/drive/MyDrive/.../dataset.yaml
```

The output shows the download of the YOLOv8.2.10 model and configuration files, followed by training logs:

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8n.pt ... 100% 6.23M/6.23M [00:00<00:00, 105MB/s]
ultralytics YOLOv8.2.10 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, engine/trainer: task=detect, mode=train, model=yolov8n.pt, data=/content/drive/...
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics...' 100% 755k/755k [00:00<00:00, 27.5MB/s]
2024-05-08 02:51:05.715058: E external/local_xla/xla/stream_executor/cuda/cuda...
2024-05-08 02:51:05.715123: E external/local_xla/xla/stream_executor/cuda/cuda...
2024-05-08 02:51:05.716449: E external/local_xla/xla/stream_executor/cuda/cuda...
Overriding model.yaml nc=80 with nc=1
```

Below the code cell, a table displays training metrics:

epoch	train/box_loss	train/cls_loss	train/dfl_loss	metrics/precision(B)
1	1.8092	3.0197	2.0428	0.00354
2	1.7196	2.706	1.9678	0.00375
3	1.7862	2.6687	2.0513	0.15934
4	1.8326	2.6836	2.0401	0.206
5	1.8467	2.6181	2.0107	0.39162
6	1.9749	3.0775	2.2392	0.23629
7	1.9745	2.8546	2.2782	0.18478
8	1.9726	2.793	2.2817	0.14424
9	1.8681	2.7446	2.2186	0.18554
10	1.8698	2.654	2.2307	0.19229

Project Evidence

Model Development

Untitled6.ipynb

File Edit View Insert Runtime Tools Help Saving failed since 7:08 PM

Comment Share

Reconnect T4 Colab AI

Files

Connecting to a runtime to enable file browsing.

+ Code + Text

```
[ ] from ultralytics import YOLO
```

!yolo task = detect mode = train model = yolov8n.pt data = /content/drive/MyDrive

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/15	2.28G	1.931	3.078	2.116	45	640:
	Class all	Images	Instances	Box(P)	R	mAP50
		16	19	0.00354	0.895	0.065
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/15	2.28G	1.835	2.827	2.053	37	640:
	Class all	Images	Instances	Box(P)	R	mAP50
		16	19	0.00313	0.789	0.0639
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/15	2.25G	1.86	2.716	2.076	35	640:
	Class all	Images	Instances	Box(P)	R	mAP50
		16	19	0.00745	0.684	0.076
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
4/15	2.25G	1.77	2.603	2.012	39	640:
	Class all	Images	Instances	Box(P)	R	mAP50
		16	19	0.121	0.105	0.127
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
5/15	2.25G	1.771	2.567	2.039	40	640:
	Class all	Images	Instances	Box(P)	R	mAP50
		16	19	1	0.103	0.183

Closing dataloader mosaic
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7))
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()

P_curve.png results.csv train_batch1.jpg data ...



Project Evidence

Model Development

The screenshot shows a Google Colab notebook interface. The left sidebar displays a file tree with a 'drive' folder and a 'MyDrive' folder containing 'Colab Notebooks', 'Extra storage', 'IT21086670_HighFidel...', 'SQA', 'Seller - HCI assignme...', 'archive_2', 'test', and 'train' folders. The 'train/images' folder contains several files named '11_10_19-mjs...' and '132343342_21...'. The main workspace shows a table of training metrics for three epochs (13/15, 14/15, 15/15) using a Tesla T4 GPU. The table includes columns for Epoch, GPU_mem, box_loss, cls_loss, dfl_loss, Instances, and Size. The right side of the screen displays a detected image titled '11_10_19-mjs_ft_hotel-fire_19183862.jpg.rf.7a950e2be2e3c3d ...' showing a scene with a building on fire.

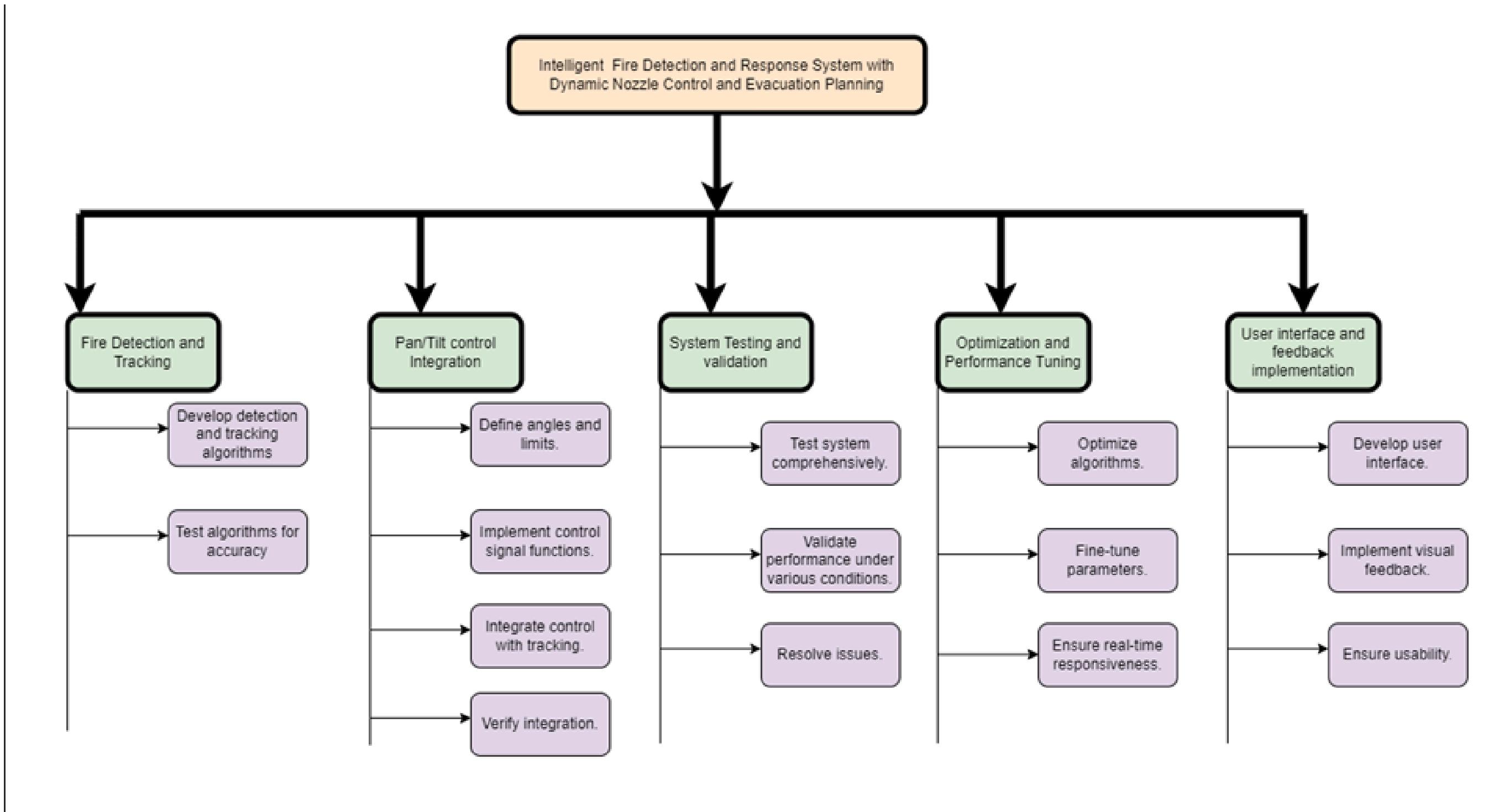
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
13/15	2.25G	1.707	2.327	2.048	18	640:
	Class	Images	Instances	Box(P	R	mAP50
	all	16	19	0.589	0.263	0.331
14/15	2.25G	1.658	2.213	1.99	20	640:
	Class	Images	Instances	Box(P	R	mAP50
	all	16	19	0.481	0.316	0.386
15/15	2.25G	1.624	2.226	1.99	19	640:
	Class	Images	Instances	Box(P	R	mAP50
	all	16	19	0.372	0.316	0.35

15 epochs completed in 0.037 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.2MB

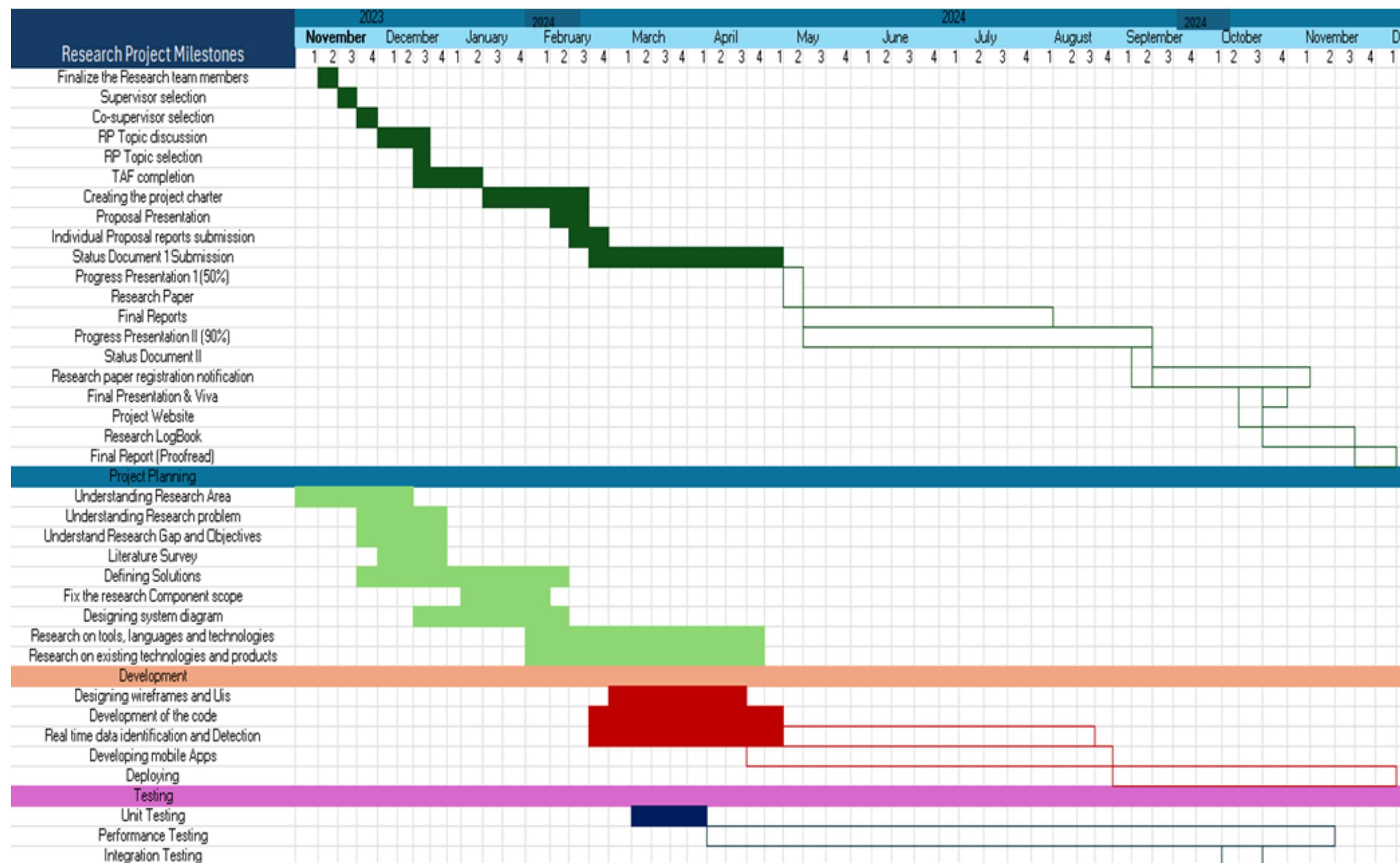
Validating runs/detect/train/weights/best.pt...
ultralytics YOLOv8.2.10 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPS Class Images Instances Box(P R mAP50 all 16 19 0.374 0.316 0.35 Speed: 0.2ms preprocess, 2.3ms inference, 0.0ms loss, 1.1ms postprocess per image Results saved to runs/detect/train Learn more at <https://docs.ultralytics.com/modes/train>

```
[1] from google.colab import drive  
/drive')
```

Work Break Down Structure



Grant Chart



Project Completion

Completed Tasks

- ✓ Developed a fire tracking system utilizing a camera to track fires.
- ✓ Implemented the transmission of tracking data to the communication part of the system.
- ✓ Provided insights into the performance of the fire detection process, including time taken for different stages and average processing speed per image.
- ✓ Implemented initial pan/tilt definitions, control signal functions, and integrated fire tracking for continuous camera position adjustment.

Tasks to be completed



Enhance the system to calculate the area covered by fire.



Enhance the system to calculate the time taken for the fire to spread.

References

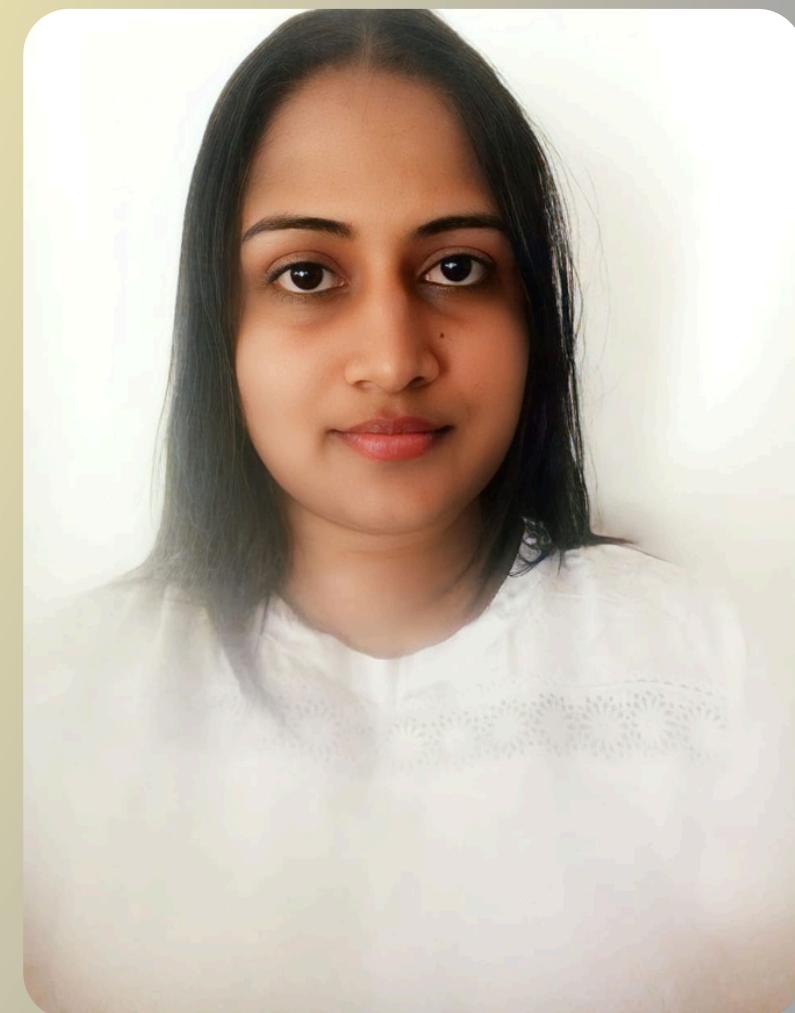
-  Afrin, A. M. (2023, 03 03). Fire and disaster detection with multimodal quadcopter By machine learning. Retrieved from <https://dspace.bracu.ac.bd/xmlui/handle/10361/20208>
-  Anastasios Dimou1, D. G. (n.d.). First Responder Advanced Technologies for Safe and Efficient Emergency Response. Retrieved from <https://www.iti.gr/iti/wp-content/uploads/m-files/document/publications/FASTER-First-Responder-Advanced-Technologies.pdf>
-  Muhammad Iqbal, D. S. (2023, July 30,). Scope of Artificial Intelligence in Enhancement of Emergency Rescue Services: Future Prospects. Retrieved from <https://alqantarajournal.com/index.php/Journal/article/view/469>
-  S. Sudhakar a, V. V. (2020, January). Unmanned Aerial Vehicle (UAV) based Forest Fire Detection and monitoring for reducing false alarms in forest-fires. Retrieved from https://docs.google.com/document/d/1gwMO_D0-RkBurqSWyxrmg8N5Bh9XtJTz/edit

THARUSHIKA W.A.V

IT21100116

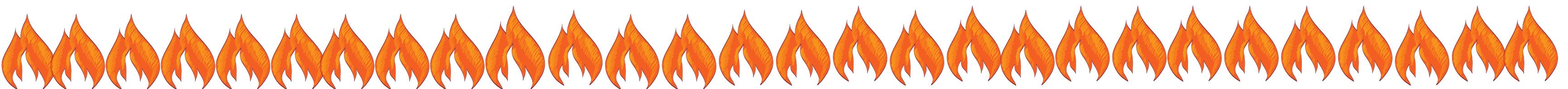
Specialization : Information Technology

Identification of the Fire disaster



BACKGROUND

1. **Fire Flame Detection:** Need for advanced flame detection; Traditional methods struggle amidst varied conditions; accurate differentiation essential..
2. **Enhancing flame detection accuracy through computer vision:** Shape and motion analysis extract vital visual features for precise identification.
3. **Critical need for robust flame detection algorithms:** Must perform effectively in diverse, challenging environments to enhance emergency response.
4. **Novel approach:** Fuse shape and motion analysis with traditional methods for enhanced flame detection accuracy, overcoming existing limitations.

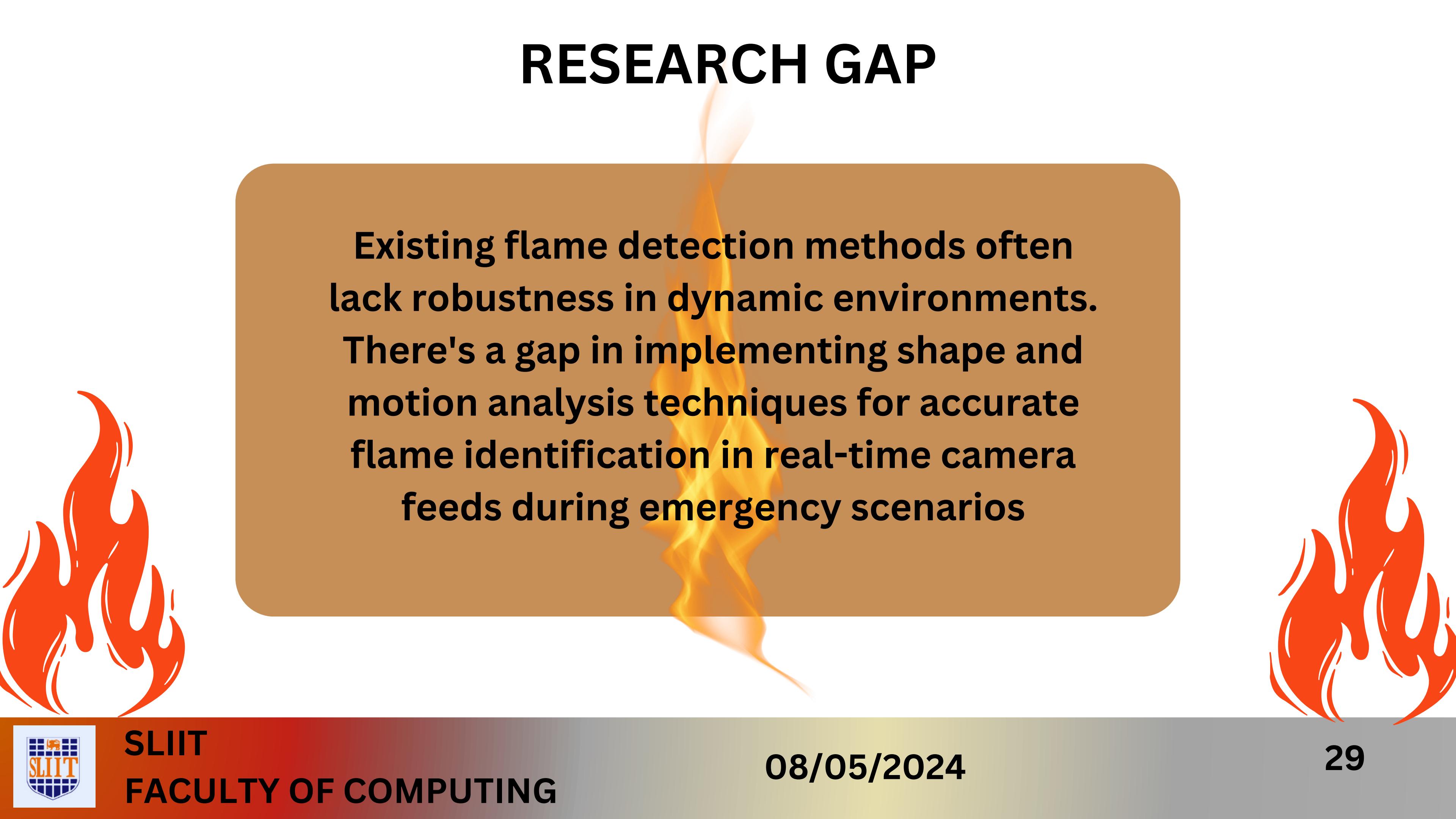


RESEARCH QUESTION



How can the integration of shape analysis and motion analysis techniques enhance the accuracy and robustness of fire flame detection in real-time camera feeds during emergency situations?

RESEARCH GAP



Existing flame detection methods often lack robustness in dynamic environments. There's a gap in implementing shape and motion analysis techniques for accurate flame identification in real-time camera feeds during emergency scenarios

SPECIFIC AND SUB OBJECTIVES

Develop Shape Analysis Techniques for Flame Detection

-  Design algorithms to detect flame contours and extract relevant shape features.
-  Implement methods to filter out non-flame regions based on shape characteristics.
-  Validate the effectiveness of shape analysis techniques through quantitative evaluation metrics.
-  Optimize shape analysis algorithms for real-time performance on camera feeds.

Implement Motion Analysis Techniques for Flame Detection



Develop algorithms to detect flame-like motion patterns in video frames.



Design mechanisms to differentiate flame motion from background movement.

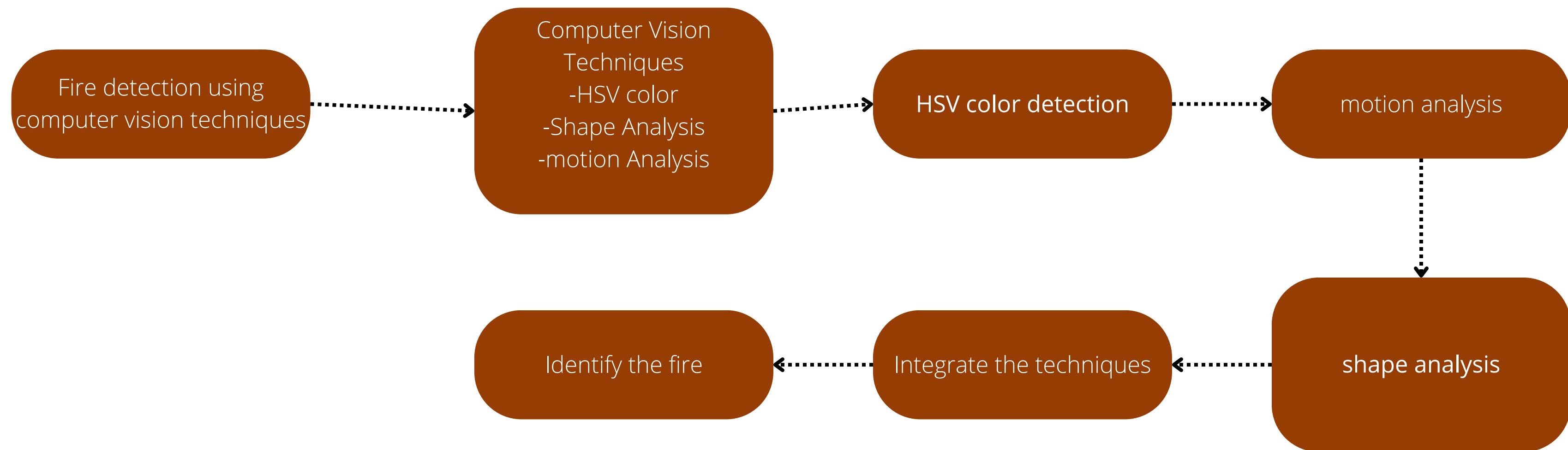


Assess the performance of motion analysis techniques under varying environmental conditions.



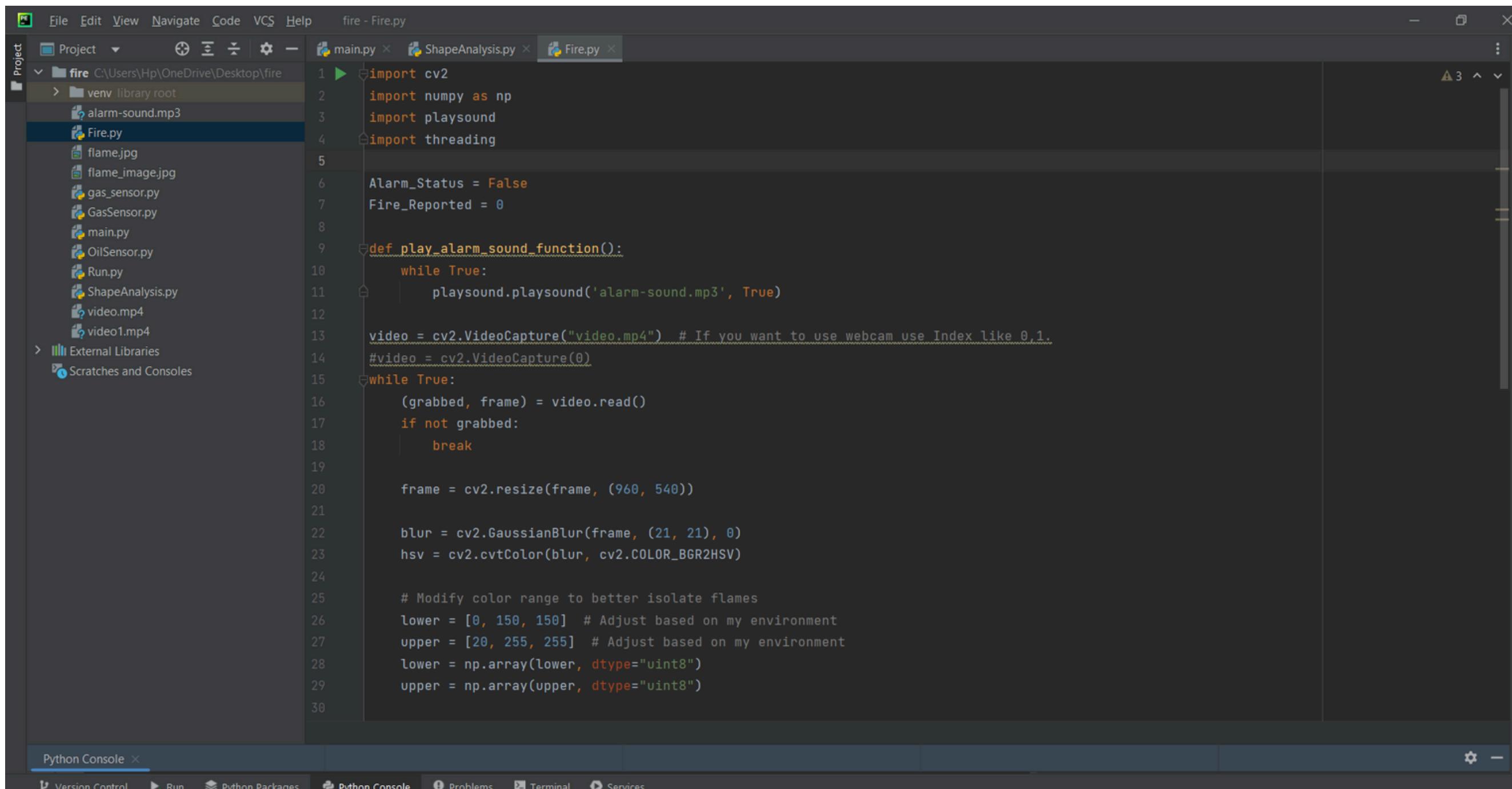
Integrate motion analysis algorithms with real-time processing systems for camera feeds.

SYSTEM DIAGRAM



Project Evidence

flame detection using HSV colors



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** fire (C:\Users\Hp\OneDrive\Desktop\fire)
- Files:** main.py, ShapeAnalysis.py, Fire.py (selected), flame.jpg, flame_image.jpg, gas_sensor.py, GasSensor.py, main.py, OilSensor.py, Run.py, ShapeAnalysis.py, video.mp4, video1.mp4
- Code Editor:** Content of Fire.py:

```
1 import cv2
2 import numpy as np
3 import playsound
4 import threading
5
6 Alarm_Status = False
7 Fire_Reported = 0
8
9 def play_alarm_sound_function():
10     while True:
11         playsound.playsound('alarm-sound.mp3', True)
12
13     video = cv2.VideoCapture("video.mp4") # If you want to use webcam use Index like 0,1.
14 #video = cv2.VideoCapture(0)
15     while True:
16         (grabbed, frame) = video.read()
17         if not grabbed:
18             break
19
20         frame = cv2.resize(frame, (960, 540))
21
22         blur = cv2.GaussianBlur(frame, (21, 21), 0)
23         hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
24
25         # Modify color range to better isolate flames
26         lower = [0, 150, 150] # Adjust based on my environment
27         upper = [20, 255, 255] # Adjust based on my environment
28         lower = np.array(lower, dtype="uint8")
29         upper = np.array(upper, dtype="uint8")
```
- Toolbars:** File, Edit, View, Navigate, Code, VCS, Help
- Bottom Bar:** Python Console, Version Control, Run, Python Packages, Python Console, Problems, Terminal, Services



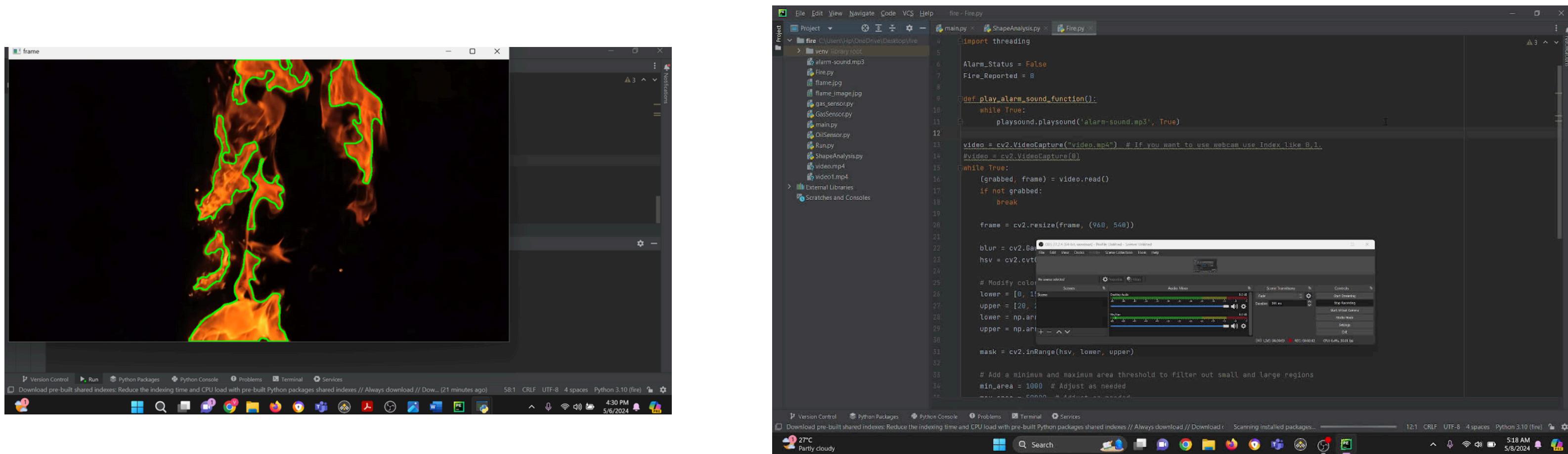
The screenshot shows the PyCharm IDE interface with a Python project named 'fire'. The 'Project' tool window on the left lists files including 'main.py', 'ShapeAnalysis.py', 'Fire.py', 'alarm-sound.mp3', 'flame.jpg', 'flame_image.jpg', 'gas_sensor.py', 'GasSensor.py', 'main.py', 'OilSensor.py', 'Run.py', 'ShapeAnalysis.py', 'video.mp4', 'video1.mp4', and 'External Libraries'. The 'Notifications' panel on the right shows 3 notifications. The 'Fire.py' file is open in the main editor, displaying code for fire detection using OpenCV. The code reads a video, applies color thresholds to detect flames, filters contours by area, and triggers an alarm if a fire is detected. It also handles user input to exit the application.

```
File Edit View Navigate Code VCS Help fire - Fire.py
Project fire C:\Users\Hp\OneDrive\Desktop\fire
main.py ShapeAnalysis.py Fire.py
28     lower = np.array(lower, dtype="uint8")
29     upper = np.array(upper, dtype="uint8")
30
31     mask = cv2.inRange(hsv, lower, upper)
32
33     # Add a minimum and maximum area threshold to filter out small and large regions
34     min_area = 1000 # Adjust as needed
35     max_area = 50000 # Adjust as needed
36
37     # Filter by area
38     contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
39     for contour in contours:
40         area = cv2.contourArea(contour)
41         if min_area < area < max_area:
42             # Perform additional analysis or trigger fire detection
43             cv2.drawContours(frame, [contour], -1, (0, 255, 0), 2)
44             Fire_Reported += 1
45
46             cv2.imshow("frame", frame)
47
48             if Fire_Reported >= 1:
49                 if not Alarm_Status:
50                     threading.Thread(target=play_alarm_sound_function).start()
51                     Alarm_Status = True
52
53             if cv2.waitKey(1) & 0xFF == ord('q'):
54                 break
55
56             cv2.destroyAllWindows()
57             video.release()
```



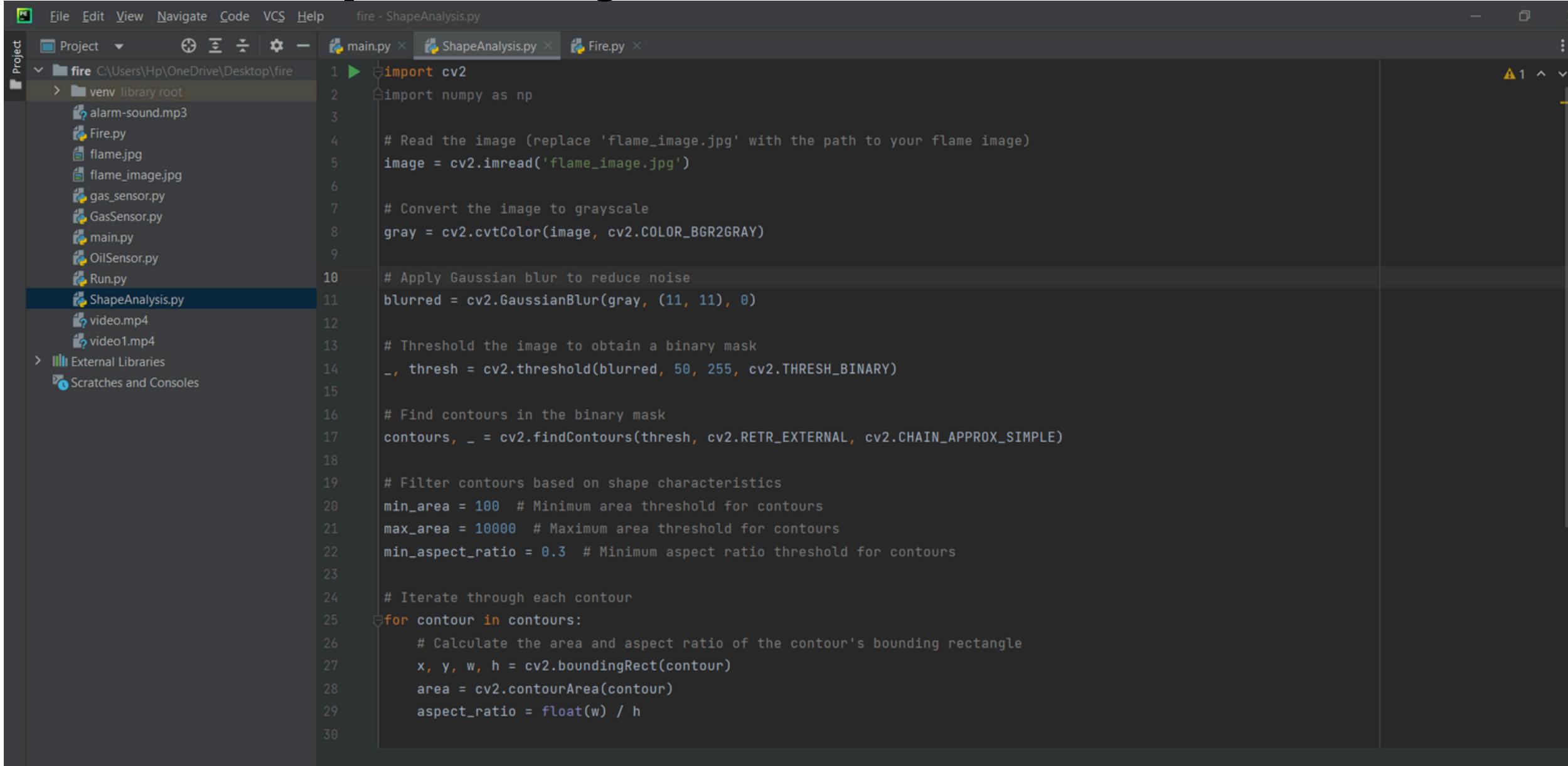
Project Evidence

Output of the flame detection using HSV colors



Project Evidence

Shape Analysis of the Fire flames



The screenshot shows a code editor window with the following details:

- Project:** fire - ShapeAnalysis.py
- Files:** main.py, ShapeAnalysis.py (selected), Fire.py, flame.jpg, flame_image.jpg, gas_sensor.py, GasSensor.py, main.py, OilSensor.py, Run.py, ShapeAnalysis.py, video.mp4, video1.mp4.
- Code Content:** Python script for shape analysis of fire flames using OpenCV. The code reads a flame image, converts it to grayscale, applies Gaussian blur, thresholds it to obtain a binary mask, finds contours, filters them based on area and aspect ratio, and then iterates through each contour to calculate its bounding rectangle and aspect ratio.

```
import cv2
import numpy as np

# Read the image (replace 'flame_image.jpg' with the path to your flame image)
image = cv2.imread('flame_image.jpg')

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Gaussian blur to reduce noise
blurred = cv2.GaussianBlur(gray, (11, 11), 0)

# Threshold the image to obtain a binary mask
_, thresh = cv2.threshold(blurred, 50, 255, cv2.THRESH_BINARY)

# Find contours in the binary mask
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

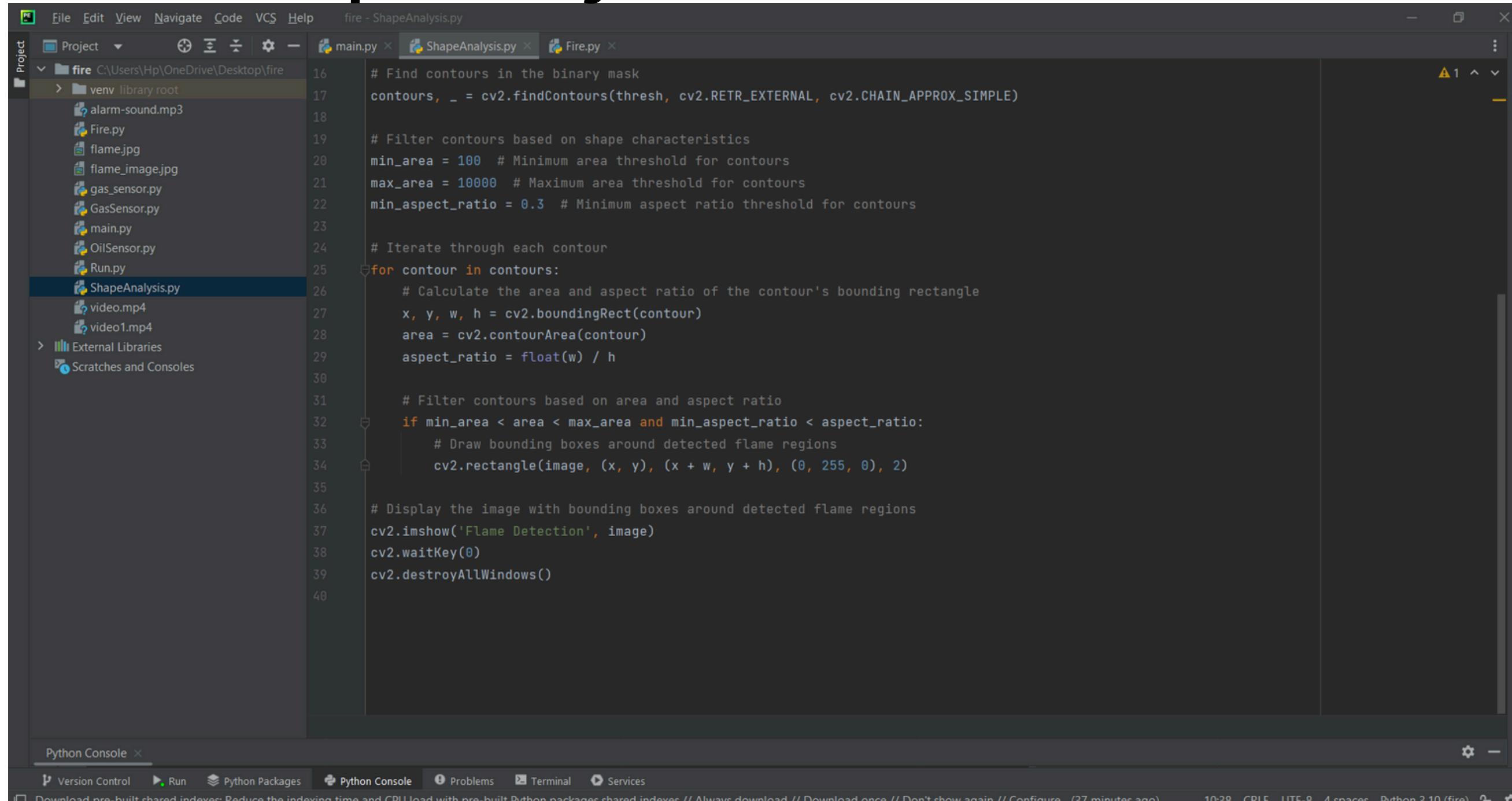
# Filter contours based on shape characteristics
min_area = 100 # Minimum area threshold for contours
max_area = 10000 # Maximum area threshold for contours
min_aspect_ratio = 0.3 # Minimum aspect ratio threshold for contours

# Iterate through each contour
for contour in contours:
    # Calculate the area and aspect ratio of the contour's bounding rectangle
    x, y, w, h = cv2.boundingRect(contour)
    area = cv2.contourArea(contour)
    aspect_ratio = float(w) / h
```



Project Evidence

Shape Analysis of the Fire flames



The screenshot shows a code editor interface with a dark theme. The left sidebar displays a project structure for a 'fire' project located at C:\Users\Hp\OneDrive\Desktop\fire. The 'venv library root' folder contains several files: alarm-sound.mp3, Fire.py, flame.jpg, flame_image.jpg, gas_sensor.py, GasSensor.py, main.py, OilSensor.py, Run.py, ShapeAnalysis.py (which is currently selected), video.mp4, and video1.mp4. Below the project tree are sections for 'External Libraries' and 'Scratches and Consoles'. The main code editor area shows a Python script named 'ShapeAnalysis.py' with the following content:

```
16 # Find contours in the binary mask
17 contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
18
19 # Filter contours based on shape characteristics
20 min_area = 100 # Minimum area threshold for contours
21 max_area = 10000 # Maximum area threshold for contours
22 min_aspect_ratio = 0.3 # Minimum aspect ratio threshold for contours
23
24 # Iterate through each contour
25 for contour in contours:
26     # Calculate the area and aspect ratio of the contour's bounding rectangle
27     x, y, w, h = cv2.boundingRect(contour)
28     area = cv2.contourArea(contour)
29     aspect_ratio = float(w) / h
30
31     # Filter contours based on area and aspect ratio
32     if min_area < area < max_area and min_aspect_ratio < aspect_ratio:
33         # Draw bounding boxes around detected flame regions
34         cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
35
36     # Display the image with bounding boxes around detected flame regions
37     cv2.imshow('Flame Detection', image)
38     cv2.waitKey(0)
39     cv2.destroyAllWindows()
40
```

At the bottom of the code editor, there is a tab bar with 'Python Console' and other tabs like 'Version Control', 'Run', 'Python Packages', 'Problems', 'Terminal', and 'Services'. A status bar at the bottom shows the path 'Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Configure...', the date '10:38', and the file 'CRLF UTF-8 4 spaces Python 3.10 (fire)'.



Project Evidence

Output of the flame detection using Shape Analysis



Project Evidence

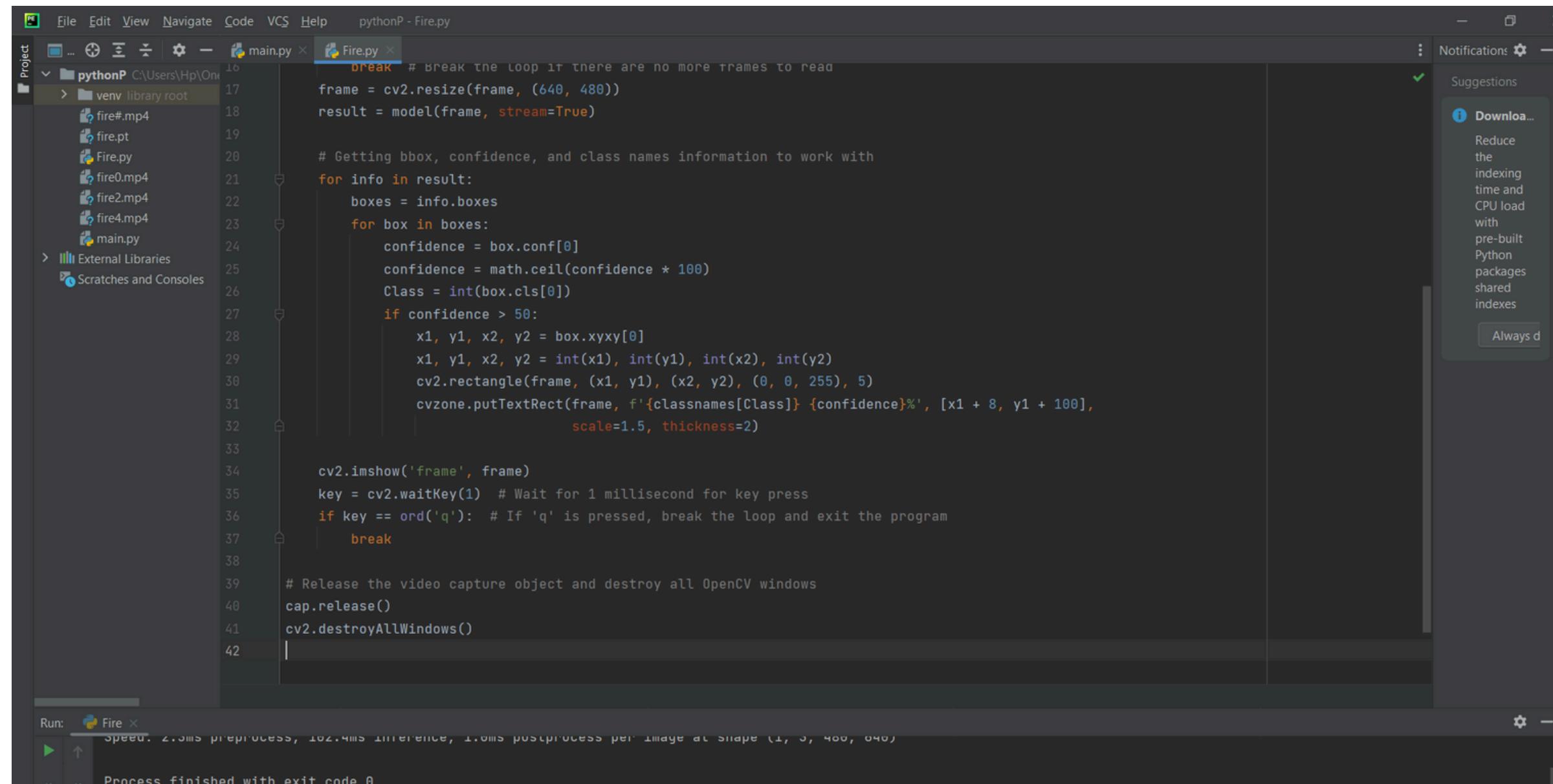
Code of the fire flame detection using data model

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, VCS, Help.
- Project Bar:** pythonP - Fire.py
- Toolbars:** Standard, Main, Run, Version Control.
- Left Sidebar:** Shows the project structure with files like main.py, Fire.py, and various mp4 files. It also includes External Libraries and Scratches and Consoles.
- Code Editor:** Displays the Python script for fire detection. The code uses the ultralytics YOLO library to process video frames from a webcam or file. It identifies objects and filters them based on confidence levels.
- Right Sidebar:** Notifications and Suggestions. A suggestion is shown: "Reduce the indexing time and CPU load with pre-built Python packages shared indexes". An "Always" button is present.
- Bottom Status Bar:** Shows the current process status: Speed: 2.5ms preprocess, 102.4ms inference, 1.0ms postprocess per image at Shape (1, 3, 480, 640). It also indicates "Process finished with exit code 0".

Project Evidence

Code of the fire flame detection using data model



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** pythonP
- Files:** main.py, Fire.py, fire#mp4, fire.pt, Fire.py, fire0.mp4, fire2.mp4, fire4.mp4, main.py
- Main.py Content:**

```
16     break # Break the loop if there are no more frames to read
17
18     frame = cv2.resize(frame, (640, 480))
19     result = model(frame, stream=True)
20
21     # Getting bbox, confidence, and class names information to work with
22     for info in result:
23         boxes = info.bboxes
24         for box in boxes:
25             confidence = box.conf[0]
26             confidence = math.ceil(confidence * 100)
27             Class = int(box.cls[0])
28             if confidence > 50:
29                 x1, y1, x2, y2 = box.xyxy[0]
30                 x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
31                 cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 5)
32                 cvzone.putTextRect(frame, f'{classnames[Class]} {confidence}%', [x1 + 8, y1 + 100],
33                                   scale=1.5, thickness=2)
34
35                 cv2.imshow('frame', frame)
36                 key = cv2.waitKey(1) # Wait for 1 millisecond for key press
37                 if key == ord('q'): # If 'q' is pressed, break the loop and exit the program
38                     break
39
40             # Release the video capture object and destroy all OpenCV windows
41             cap.release()
42             cv2.destroyAllWindows()
```

- Run Tab:** Fire
- Output:** speed. 2.0ms preprocess, 102.4ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
Process finished with exit code 0



Project Evidence

output of the fire flame detection using data model

The screenshot shows the PyCharm IDE interface. The top part displays the code for a file named `Fire.py`. The code imports `ultralytics`, `cvzone`, `cv2`, and `math`. It sets up a video capture from a file named `fire#.mp4` and loads a YOLO model from `fire.pt`. A loop reads frames from the video, processes them with the model, and breaks if no more frames are available. The bottom part shows the terminal window with the output of the script. The output consists of five lines, each starting with '0:' followed by a timestamp and a message indicating 'no detections'. The times range from 110.7ms to 128.4ms. The terminal also shows the PyCharm interface at the bottom, including the status bar which indicates the current temperature is 27°C.

```
from ultralytics import YOLO
import cvzone
import cv2
import math

# Running real time from webcam
cap = cv2.VideoCapture("fire#.mp4")
model = YOLO('fire.pt')

classnames = ['fire']

while True:
    ret, frame = cap.read()
    if not ret:
        break # Break the loop if there are no more frames to read
```

```
0: 480x640 (no detections), 110.7ms
Speed: 3.0ms preprocess, 118.9ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)

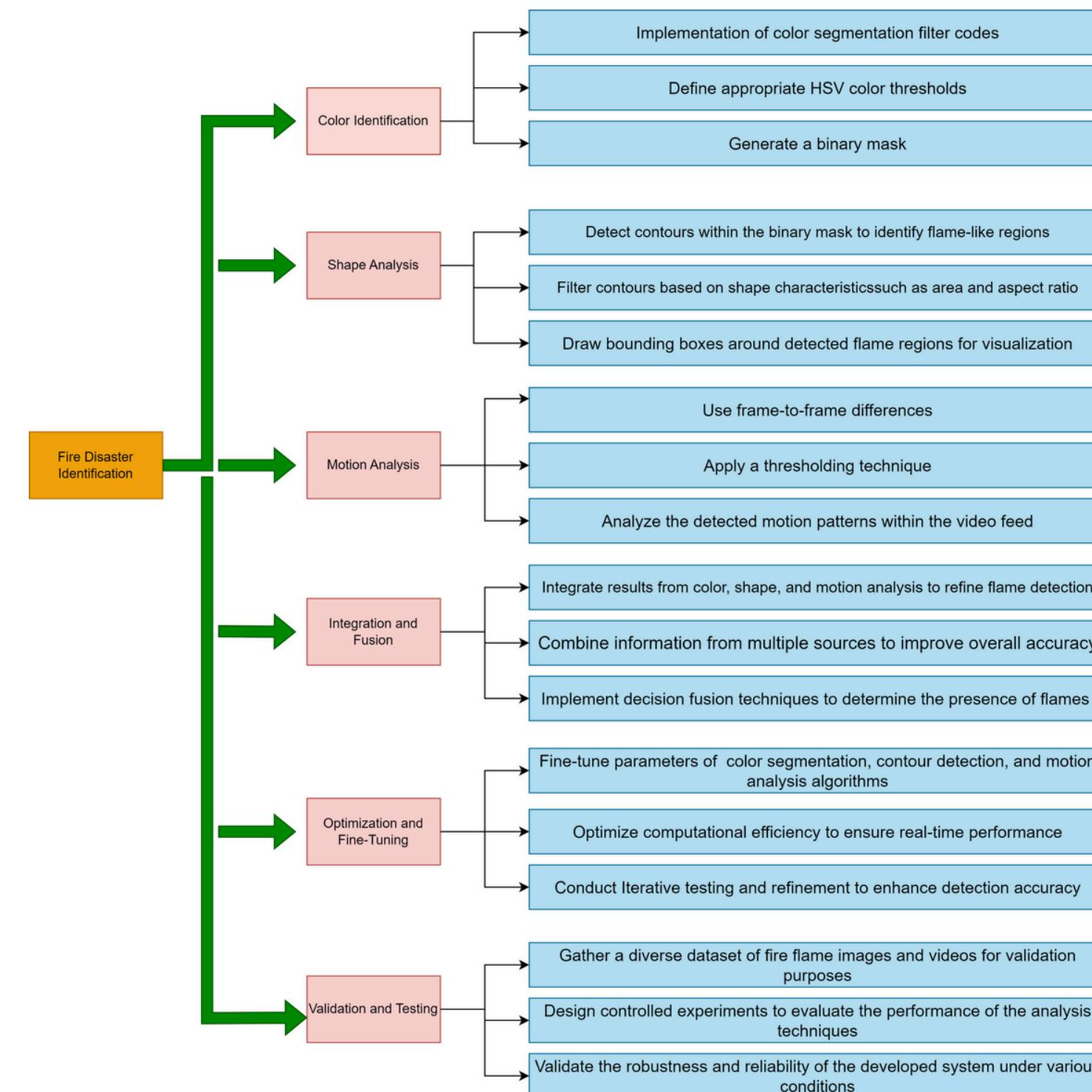
0: 480x640 (no detections), 123.5ms
Speed: 1.0ms preprocess, 123.5ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 (no detections), 128.4ms
Speed: 1.0ms preprocess, 128.4ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)

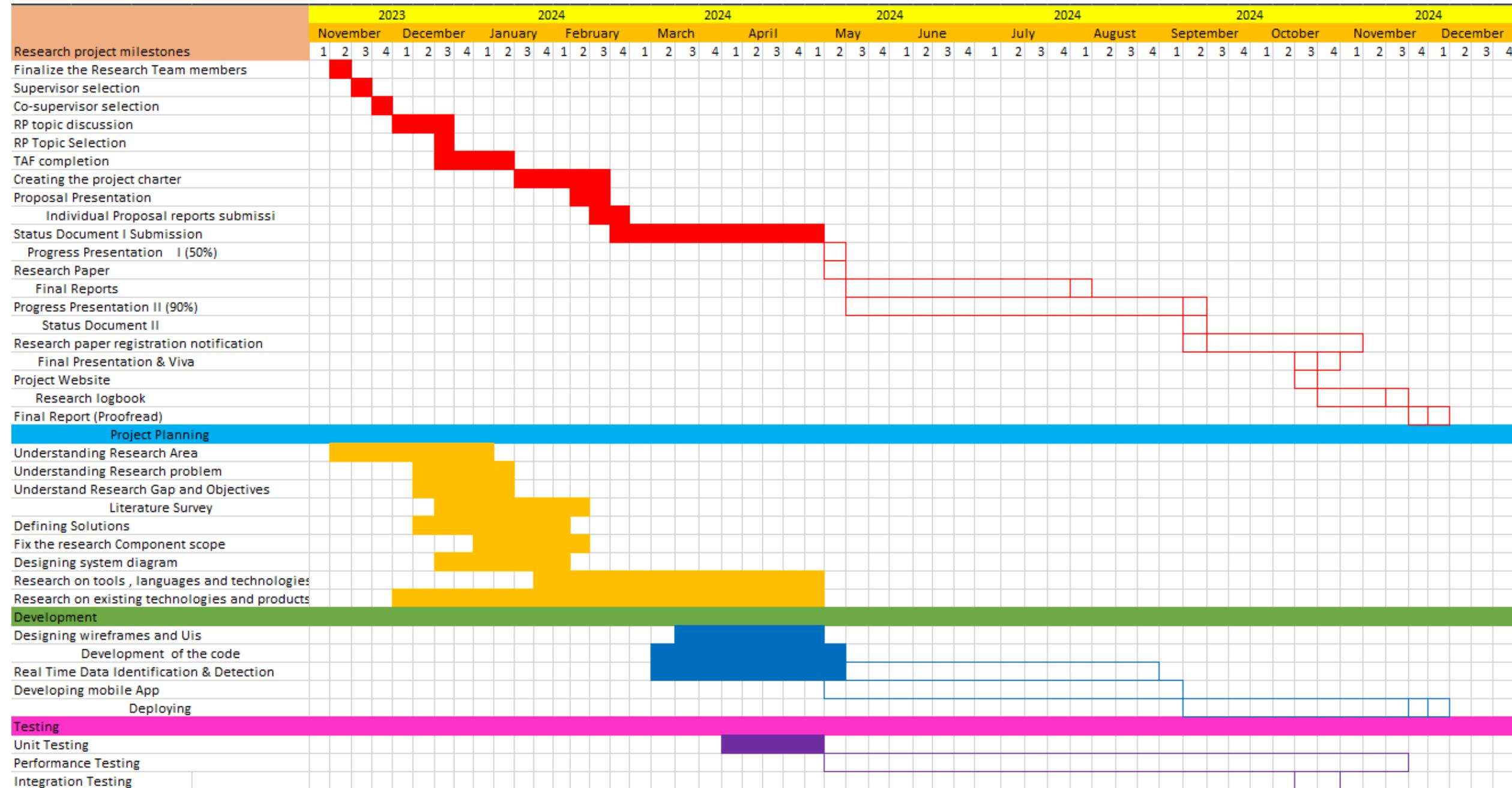
0: 480x640 (no detections), 115.4ms
Speed: 1.0ms preprocess, 115.4ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
```



Work Break Down Structure



Grant Chart



Project Completion



Completed Tasks

- ✓ Identify and detect fire using data model
- ✓ Identify fire using HSV colors
- ✓ Working on shape Analysis part

Tasks to be completed

-  Motion analysis and shape analysis
-  Integrate the computer vision techniques
-  Identify the most effective method for disaster identification

References

-  Caleb Boadi, S. K.-d. (2015). Modelling of fire count data: fire disaster risk in Ghana. Retrieved from <https://springerplus.springeropen.com/articles/10.1186/s40064-015-1585-3>
-  Kihila, J. M. (2017, February 21). Fire disaster preparedness and situational analysis in higher learning institutions of Tanzania. Retrieved from <https://journals.co.za/doi/abs/10.4102/jamba.v9i1.311>
-  L Kamelia1, N. I. (n.d.). Journal of Physics: Conference Series. Retrieved from <https://iopscience.iop.org/article/10.1088/1742-6596/1402/4/044001/meta>
-  Waheed, M. A. (2014). Approach to Fire-related Disaster Management in High Density Urban-area. Retrieved from <https://www.sciencedirect.com/science/article/pii/S187705814009849>

Rathnayaka R.M.S.T.

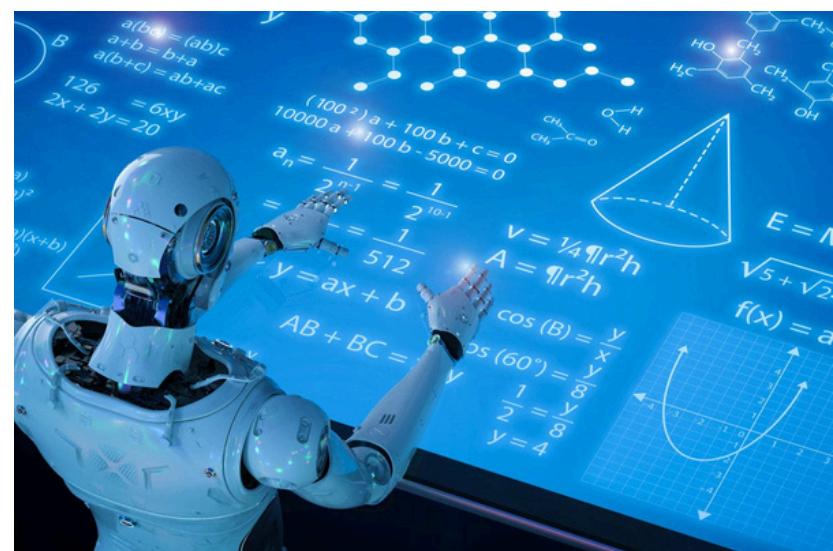
IT21101342

Specialization : Information Technology

Activation of the Rescue Mission



BACKGROUND



The background of a system or project typically refers to the context, history, and underlying factors that have led to its development or implementation. It provides an understanding of the problem or opportunity that the system aims to address and the reasons why it is necessary or beneficial.

RESEARCH QUESTION



How can computer vision and machine learning techniques be leveraged to enhance the accuracy and efficiency of real-time fire detection in outdoor environments?

RESEARCH GAP

Limited Integration of Robotic Systems

Current approaches to fire disaster response often lack efficient integration of multiple robotic units, leading to suboptimal coordination and resource utilization during rescue operations.



Inadequate Autonomous Decision-Making

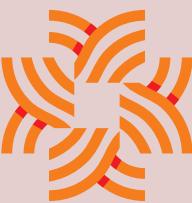
Existing systems may rely heavily on human intervention for decision-making, resulting in delays and inefficiencies in response times, particularly in dynamic and unpredictable fire environments.



SPECIFIC AND SUB OBJECTIVES



Real-Time Data Analysis



Geotagging & mapping

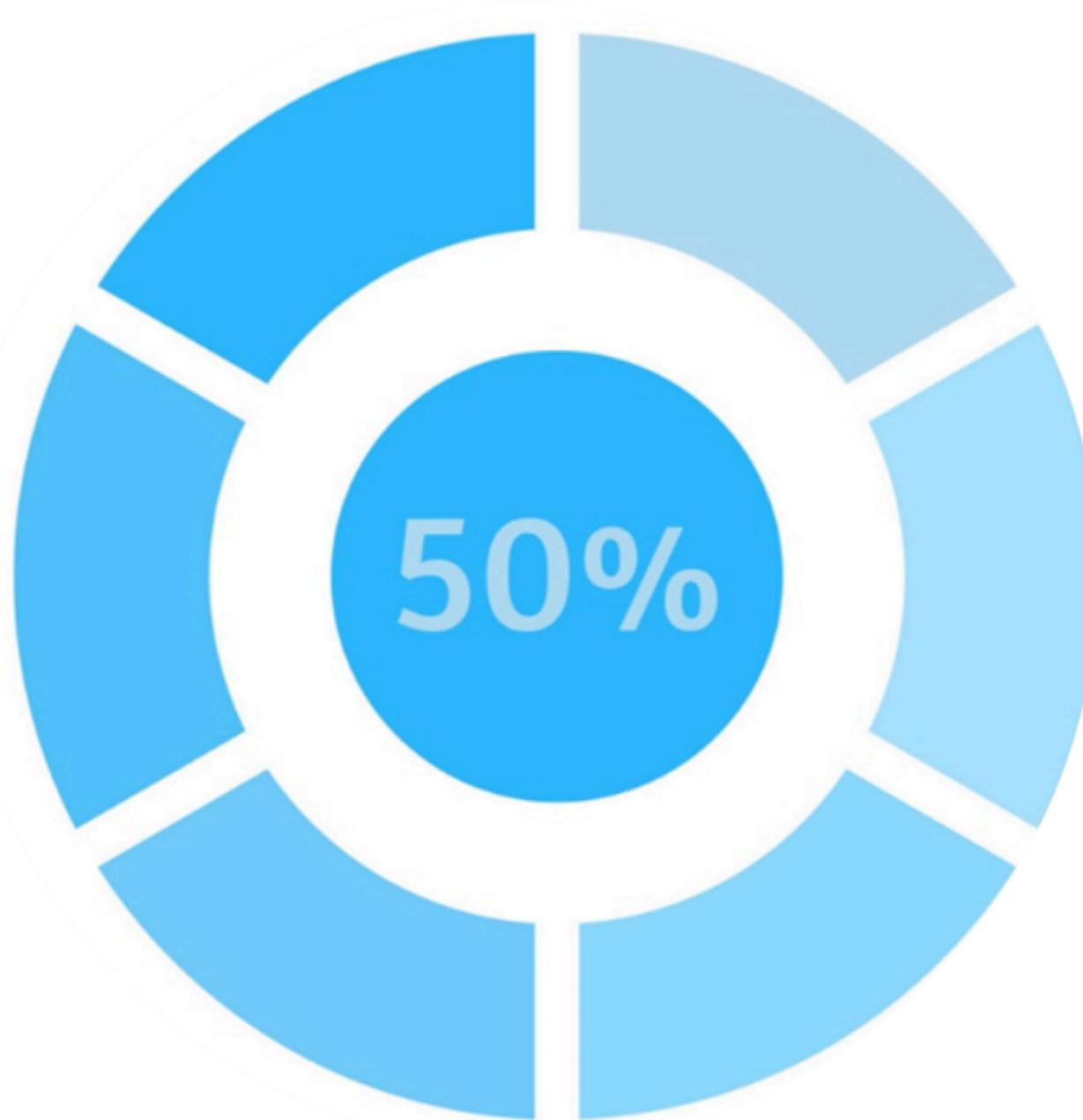


Predictive Analytics

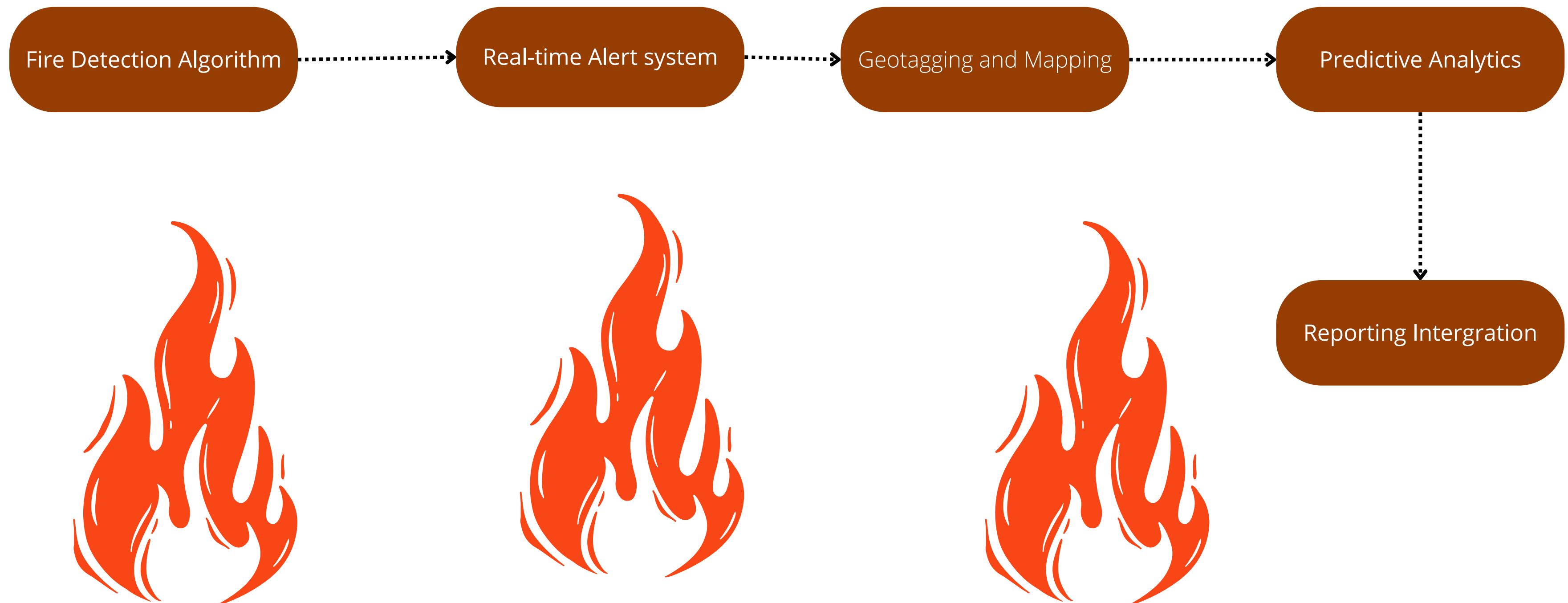


Reporting Integrate

CURRENT PROGRESS



SYSTEM DIAGRAM



Project Evidence

Create real-time alert

```
Activation_Rescue_Mission > Real_time_Alert.py > ...
1  import smtplib
2  from email.mime.text import MIMEText
3  from email.mime.multipart import MIMEMultipart
4
5  # Define sender email
6  sender_email = 'sandunirathnayaka80@gmail.com'
7
8  def send_email_alert(sender_email, sender_password, recipient_email, subject, message):
9      # Set up SMTP server
10     smtp_server = 'smtp.example.com'
11     smtp_port = 587 # Change to your SMTP port if different
12
13     # Create email message
14     email = MIMEMultipart()
15     email['From'] = sender_email
16     email['To'] = recipient_email
17     email['Subject'] = subject
18     email.attach(MIMEText(message, 'plain'))
19
20     server = None
21     try:
22         # Connect to SMTP server
23         server = smtplib.SMTP(smtp_server, smtp_port)
24         server.starttls() # Enable encryption
25         # Login to sender's email account
26         server.login(sender_email, sender_password)
```



```
def send_email_alert(sender_email, sender_password, recipient_email, subject, message):
    # Connect to SMTP server
    server = smtplib.SMTP(smtp_server, smtp_port)
    server.starttls() # Enable encryption
    # Login to sender's email account
    server.login(sender_email, sender_password)
    # Send email
    server.sendmail(sender_email, recipient_email, email.as_string())
    print("Email alert sent successfully!")
except Exception as e:
    print("Failed to send email alert:", str(e))
finally:
    # Close SMTP connection if it's open
    if server:
        server.quit()

# Usage
sender_password = 'Sandu1999#'
recipient_email = 'thakshilarathnayaka80@gmail.com'
subject = 'Fire Detected!'
message = 'A fire has been detected at the following location: Latitude: xx.xxxx, Longitude: yy.yyyy'
send_email_alert(sender_email, sender_password, recipient_email, subject, message)
```

```
[Running] python -u
"c:\Users\saduni\Desktop\fire-detection-system-in-python-opencv-main\fire-detection-system-in-python-opencv-main\Activation_Rescue_Mission\Real_time_Alert.py"
Failed to send email alert: [Errno 11001] getaddrinfo failed

[Done] exited with code=0 in 1.605 seconds
```



Create Geotagging & Mapping

```
import requests

def update_map_with_fire_location(latitude, longitude):
    api_key = 'your_api_key' # Replace 'your_api_key' with your actual Google Maps API key
    map_size = '400x400' # Adjust map size as needed
    marker_color = 'red'
    marker_label = 'F'

    # Construct the URL for the static map image
    url = f'https://maps.googleapis.com/maps/api/staticmap?center={latitude},{longitude}&zoom=12&size={map_size}&maptype=roadmap&markers=color:{marker_color}|{latitude},{longitude}&label={marker_label}'

    # Send a GET request to the Google Maps Static API
    response = requests.get(url)

    # Save the map image to a file
    with open('fire_location_map.png', 'wb') as f:
        f.write(response.content)

    print("Map updated with fire location!")

# Usage
latitude = 37.7749 # Replace with the actual latitude of the fire location
longitude = -122.4194 # Replace with the actual longitude of the fire location
update_map_with_fire_location(latitude, longitude)
```

```
[Running] python -u
"c:\Users\saduni\Desktop\fire-detection-system-in-python-opencv-main\fire-detection-system-in-python-opencv-main\Activation_Rescue_Mission\Geotagging _Analytics.py"
Map updated with fire location!
```



Create Predictive Analytics

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

def predict_fire_outbreak(weather_data, historical_fire_data):
    # Split data into features (weather_data) and target (historical_fire_data)
    X = np.array(weather_data).reshape(-1, 1)
    y = np.array(historical_fire_data)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Create and train linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate model performance
    mse = mean_squared_error(y_test, y_pred)
    print("Mean Squared Error:", mse)

    # Example prediction
    new_weather_data = [[30]] # Example: temperature of 30 degrees Celsius
    predicted_outbreak = model.predict(new_weather_data)
    print("Predicted fire outbreak probability:", predicted_outbreak[0])
```



```
    return model

# Usage
weather_data = [25, 28, 30, 35, 20, 22, 18] # Example: temperature data
historical_fire_data = [0, 0, 1, 1, 0, 0, 0] # Example: binary fire outbreak data (0=no fire, 1=fire)
predict_fire_outbreak(weather_data, historical_fire_data)
```

```
[Running] python -u
"c:\Users\saduni\Desktop\fire-detection-system-in-python-opencv-main\fire-detection-system-in-python-opencv-main\Activation_Rescue_Mission\Predictive_Analytics.py"
Mean Squared Error: 0.26994129068047323
Predicted fire outbreak probability: 0.7605769230769228

[Done] exited with code=0 in 11.541 seconds
```



Create Reporting Integrate

```
from flask import Flask, request, render_template

app = Flask(__name__)

# Dummy data structure to store reported fire incidents
reported_incidents = []

@app.route('/')
def home():
    return render_template('index.html', incidents=reported_incidents)

@app.route('/report', methods=['GET', 'POST'])
def report_fire():
    if request.method == 'POST':
        location = request.form['location']
        description = request.form['description']
        reported_incidents.append({'location': location, 'description': description})
        return 'Fire incident reported successfully!'
    else:
        return render_template('report.html')

if __name__ == '__main__':
    app.run(debug=True)
```



Report Fire Incident

Location:

Description:

Submit

Reported Fire Incidents

- {% for incident in incidents %}
- **Location:** {{ incident.location }} - **Description:** {{ incident.description }}
- {% endfor %}

[Report a Fire Incident](#)



Requirement

Functional Requirement



- Fire Detection Algorithm
- Real-time Processing
- Alert Generation
- Localization of Fire
- Integration with Surveillance Systems:
- Scalability
- Configurability

Requirement

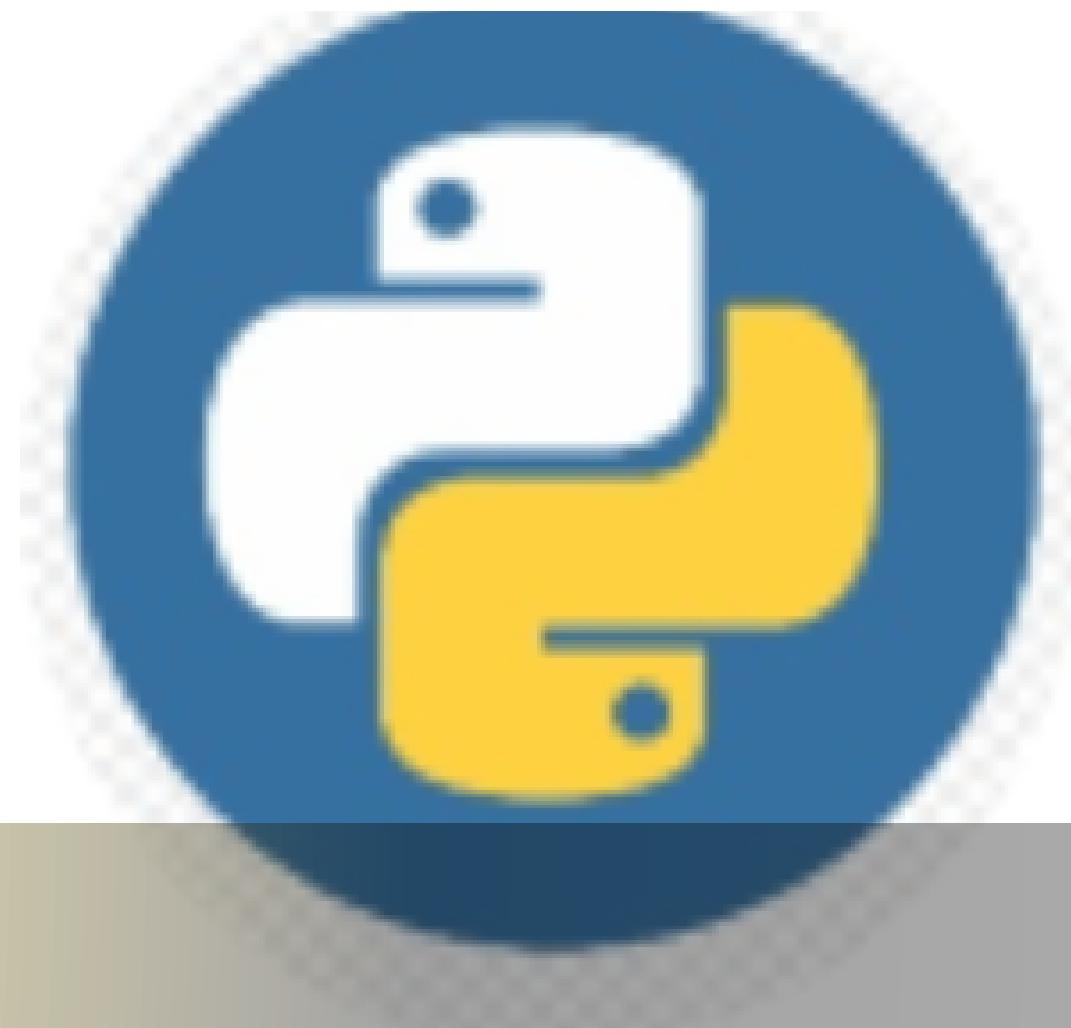
Non-functional Requirement



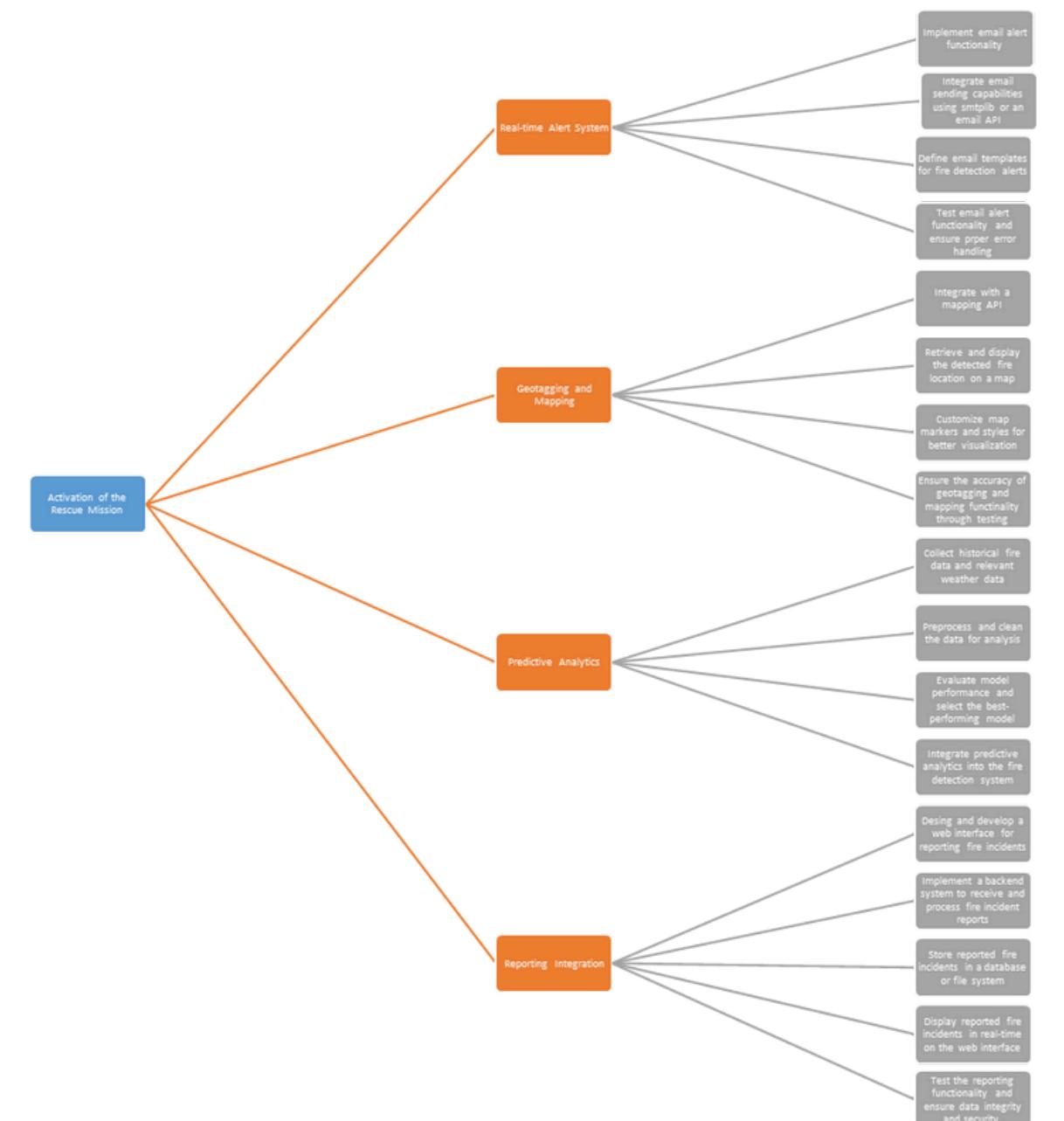
- Accuracy
- Speed
- Robustness
- Reliability
- Security
- Scalability
- Usability
- Maintainability
- Compliance

Technology

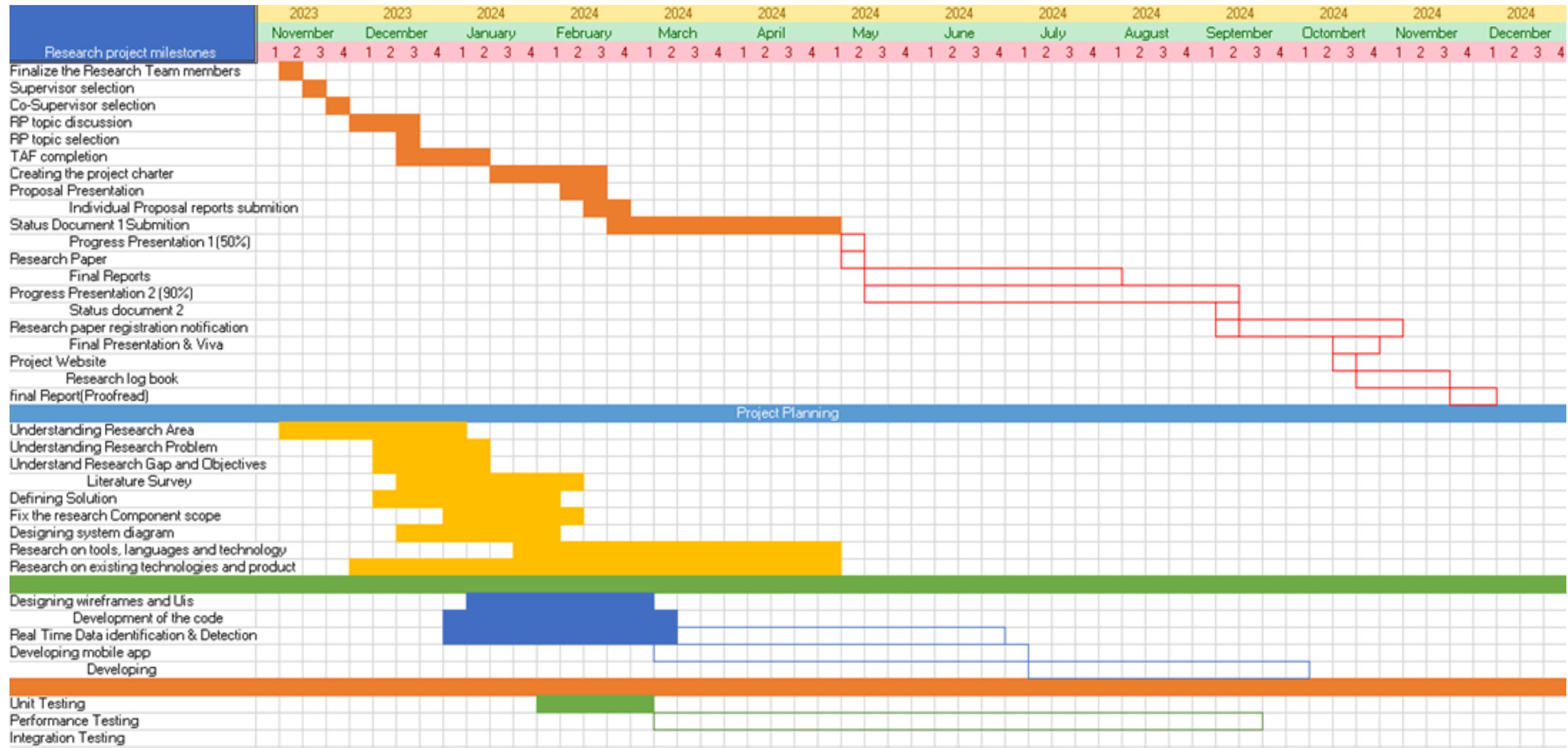
- **Frontend/ Backend**
 - Python
 - Jupyter Notebook
- **Libraries and Algorithm**
 - Machine Learning
 - Deep Q-network Algorithm
 - TensorFlow
 - NumPy, Pandas, Scikit-learn



Work Brake Down Structure



Gantt chart



References

-  Lai, C. L., Yang, J. C., & Chen, Y. H. (2007, June 25). A Real Time Video Processing Based Surveillance System for Early Fire and Flood Detection. Retrieved from <https://ieeexplore.ieee.org/abstract/document/4258251>

-  Masellis M., F. M. (n.d.). FIRE DISASTER AND BURN DISASTER: PLANNING AND MANAGEMENT. Retrieved from http://www.medbc.com/annals/review/vol_12/num_2/text/vol12n2p67.htm

-  Vladimir M. Cvetković, A. D. (2022, June 15). Fire safety behavior model for residential buildings: Implications for disaster risk reduction. Retrieved from <https://www.sciencedirect.com/science/article/pii/S221242092200200X>

-  Waheed, M. A. (2014). BIM integrated smart monitoring technique for building fire prevention and disaster relief. Retrieved from <https://www.sciencedirect.com/science/article/pii/S187705814009849>

Conclusion

the proposed system presents a comprehensive approach to disaster response, combining decentralized communication, precise geolocation, ensemble-based disaster detection, and adaptive activation algorithms. Through innovative technologies and adaptive strategies, it enhances coordination, navigation, and response effectiveness, crucial for managing dynamic disaster scenarios with efficiency and resilience.





Thank You!!

