Self-Activated Network Utilizing Fire Fighter (SNUFF)

Tanner Nelson
Weber State University
Ogden, Utah USA
tannernelson3@mail.weber.edu

***Abstract*—Fire suppression systems are often costly, cumbersome, and non-modular. A cost-effective modular unit would be beneficial in commercial and residential applications. In utilizing modern robotics architecture across a network, I will create a directional fire suppression unit that combines image recognition and low power Infra-Red detection to identify the source of a fire and extinguish the flames with expediency.**

***Index Terms*—robotics, flame, fire, detection, suppression**

I. Introduction

To implement this system, we start with a low power IR sensor, or array of sensors, that are wireless and communicate data over the network. A camera mounted on top of corresponding motors allow for a pan/tilt movement. When the IR sensor detects a flame, this will start the scanning process to locate the flame. Fire retardant will then be aimed and ejected to extinguish the flames.

A. Goals

In creating this robotic fire suppression system, I have identified key areas that need to be researched, designed, and ultimately tested and retested to ensure accurate and reliable fire suppression.

- Computer/Microprocessor
- Modular Low Power Infra-Red Detection With Integrated Network Communication
- Pan/Tilt Robotic Movement
- Image Processing via Typical USB-type Camera
- Fire Suppression
- Software Simulation
- Overall ROS(Robot Operating System) Software Integration

B. Applications

There is potential for many applications, both industrial and commercial. Fire suppression is desired in many situations, however, a characteristic of the space that inspired many design decisions was it being an indoor, small to medium size room. To quantify this, imagine the size of an average kitchen in the US. This also highlights the idea that many of our imagined design characteristics were inspired by placement in the kitchen, as many fires in the home happen here. It should also be noted that any room that often has carbon consuming flames would not be recommended, as the fire suppression services could be deployed in error.

II. Methods

A. Microprocessor Selection

1) Hardware: Though many different microcontrollers and microprocessor systems were considered, the Intel NUC was ultimately chosen to run the software driving the robot. Though Raspberry Pi, or NVidia Jetson could have been viable possibilities, a powerful and adaptable system was preferred where the robot system is stationary.

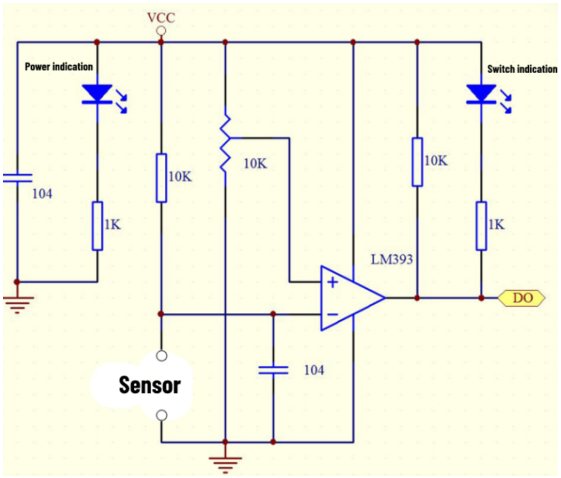
2) Software: ROS2 was the decided framework for the robot system. Due to the benefits of compatibility and latest updates, the ”iron” release was the chosen release. The operating system installed on the Intel NUC that was determined to best utilize the capabilities of ROS2, development enivronment, and repository maintenance was Ubuntu 22.04 LTS. This provided full compatibility with ROS2 - iron, Gazebo Simulation Software, Python, and MicroPython.

B. IR Detection

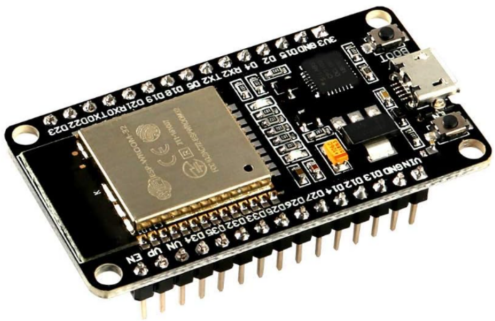
The purpose of the IR detection is to allow for an initial low-power stage. Constant image processing is taxing on power and hardware. Another benefit of the wireless IR module is the ability to sense the fire more quickly because of the potential closer range to the flame. The goal for this part of the design was to have something compact, low cost, and simple to use. This would potentially allow for the purchase of many sensors to allow for better coverage for potential fire risk areas.

1) Hardware:

Keeping cost and usability in mind, a simple off-the-shelf IR sensor was selected. At about $2 each, these small PCBs utilize an IR photodiode in a circuit with an LM393 opamp to give digital-type output. A potentiometer is then used to adjust it's sensitivity.



A real problem with these units is their sensitivity to IR from sunlight. However, when the potentiometer is adjusted, and they're placed in an area without direct sunlight, false positives are mitigated significantly.



The microcontroller that seemed most fitting for this application was the ESP32. In this particular case, the ESP32 WROOM-32 development board was chosen to utilize its ability to be physically reformatted. The IR sensor output pins were connected to the 3V3, GND, and D15 pins and the ESP32 board.

A 3d printed enclosure was also procured to house these two units and fix them to a portable USB power supply for demonstration purposes.

2) Software: A few different methods for broadcasting messages were researched were MQTT, and Micro-ROS. Partial to MQTT because of my experience, I quickly learned that MQTT to ROS conversions are not well developed. Micro-ROS became the target firmware and broadcasting message. As is the case with ROS and ROS2, Micro-ROS is under constant developed. Being relatively newly adapted to the ESP32, implementation was challenging. The "host" computer holds the Micro-ROS package, and, when run, expects the connection from the ESP32 module. There are many different methods for installing the firmware onto the ESP32. Utilizing Platformio became the path of least resistance. Micro-ROS libraries are loaded and the firmware script takes the shape of an Arduino-IDE-like program.

The firmware script includes all necessary libraries and begins with a standard setup function. This function initiates serial communication for debugging, sets up the WiFi connection and IP address and port for the host computer, and initiates the Micro-ROS functionality. A timer callback function then reads gpio 15 data periodically and broadcasts the data as a ROS message under the "ir_sensor_data" topic.

C. Pan-Tilt Robot

1) Hardware: A number of different robotics systems were provided from Weber State University to choose from. Given my need to pan, tilt, and collect visual information, my choice became clear. The PhantomX Vision Tracking Robot Turret was adopted as the main moving parts of my robot system. This unit is equipped with two DYNAMIXEL X-Series smart servos, a Microsoft LifeCam Webcam, a DYNAMIXEL U2D2 interface board, and 3d-printed housing.

2) Software: The unfortunate drawback to this system was the outdated software integration. The existing repositories currently support only ROS. Because I wanted maximum capability and in the interest of "future-proofing", ROS2 was

chosen for the system. The repositories weren't completely devoid of helpfulness, however. 3d models and some low level functionality scripts allowed for a basic start. A DYNAMIXEL library allows for serial communication to the motors via USB. Utilizing the various commands, the motors can be enabled and instructed to move to any position. These commands are sent by first finding the address of the intended functionality, then sending the appropriate command.
For the "X Series" DYNAMIXEL motors, some critical addresses and commands are shown below.

Addresses and Other Variables:

```
self.ADDR_TORQUE_ENABLE           = 64
self.ADDR_GOAL_POSITION           = 116
self.ADDR_PRESENT_POSITION        = 132
self.TORQUE_ENABLE                = 1
```

Commands:

```
#enable movement
dxl_comm_result, dxl_error =
self.packetHandler.write1ByteTxRx
(self.portHandler, i+1,
self.ADDR_TORQUE_ENABLE,
self.TORQUE_ENABLE)
#joint movement
dxl_comm_result, dxl_error =
self.packetHandler.write4ByteTxRx
(self.portHandler, i+1,
self.ADDR_GOAL_POSITION,
joint_positions[i])
```

A script that subscribes to a ROS topic then must be created to send the appropriate commands to the DYNAMIXEL library. The scripts declares the port needed for communication and sends commands to move the motors when the ROS topic is updated. The ROS topic for this communication was chosen to be "/turret".
Another script for determining the logic for movements is then created. This script is the main "executive" node for the SNUFF ROS2 package and will be explored more later in this document. As it applies to the turret, a ROS publisher sends appropriate messages to the "/turret" topic. The script also determines which messages to send based on a series of logic. First, the pan and tilt positions are put to 100 and 0 respectively at the declaration of the variables. This positions the camera in the left most position of a 180 degree span, and the tilt in the full upright position. A timer is then started that starts the "continuous_panning" function. This increments the pan quickly by 0.5. The small movements done quickly allow for a smooth panning motion. If the pan movement reaches -100, being the other end of the 180 degree span, the tilt motor increments by 20 and continues to pan. Once the tilt motor reaches a position of 60, the next pan will decrement by 20 instead. And reverses the incremental movements again when reaching the tilt position of -60. All of these bounding functions are performed by if-statements.

Once an object is detected, an if statement waits for the object to be centered along the x axis. Then another timer is started and the tilt function makes the second motor increment backward or forward depending on the location of the object. Another if-statement checks when the object is centered along the y axis, and another timer starts an oscillation function. Using a sine equation, the pan motor oscillates slowly back and forth, simulating a spraying motion. After x seconds, the program shuts down.

D. Image Processing

The Python package OpenCV is used for image recognition and ultimately fire recognition. To implement this, a ROS publisher script is made to take the data from the camera, analyze it, and send that data in a ROS message; subscribed to be the "executive" node. This flame state publisher starts by subscribing to the ROS package that receives the data for the USB camera. It then runs an "image_callback" function which shows the camera feed and then runs the "detect_flame" function to process the data. An if statement then checks if the "detect_flame" function finds a match and displays a confirmation messge.
The "detect_flame" function is where the main image processing happens. First the "hsv" tool from the OpenCV library allows us to set the upper and lower bounds for our color profile which includes hue, saturation, and brightness. Two different profiles were determined to work well in either low light, or bright conditions. These two profiles are shown below.

```
# Define a color range for flames in HSV

# Uncomment for Bright Areas
# lower_bound = np.array([18, 0, 210],
dtype=np.uint8)
# upper_bound = np.array([35, 50, 255],
dtype=np.uint8)

# Uncomment for Dark Areas
lower_bound = np.array([0, 0, 255],
dtype=np.uint8)
upper_bound = np.array([0, 30, 255],
dtype=np.uint8)
```

A mask is made with these lower and upper bounds and applied to the image. Utilizing the "cvtColor", "threshold", and "findContours" functions of OpenCV, the image is converted to grayscale, the threshold is identified, and from there the contours are found to identify a match to our parameters. An if-statement utilizing the contours tries to find an area greater than 100. If this happens a ROS string message is sent with the image data.

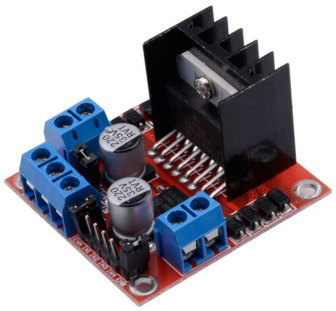
```
flame_msg.data = f"Flame detected at x: {x},
y: {y}, width: {w}, height: {h}"
```

The x and y parameters of this message are used to determine if the detected flame is in the center of the camera frame as shown previously.

E. Fire Suppression

Many different contained fire suppressants are a availble and get easily be adapted to this project. However for testing and safety, an electrically activated water jet was chosen instead.

1) Hardware: To maintain simplicity and low cost, a 5 volt or less electrical system seemed the best fit. To accomplish this, we started with a solenoid valve rated for 4.5V and worked our way out.
The solenoid valve has a 1/2 inch threaded inlet and outlet. A 1/2 inch thread to 1/4 inch barbed fitting can then be attached to allow connection to 1/4 inch I.D. PVC Clear vinyl tubing. A water tank with a hydraulic hand pump is connected to the inlet tube and a 1/4 inch spray nozzle is connected to the outlet tube and affixed to the top of the camera turret.
To control the opening and closing of the valve an ESP32 was chosen to keep the design simple, keep costs low, and stay consistent. The solenoid valve needs to draw at least 150mA which can't be provided by any GPIO pin on the ESP32. Because of this and the fact that the solenoid takes positive and negative pulses to open and close, it's fitting to use an h-bridge. Typically for motor control, these h-bridges provide the perfect output for a valve like this.



Originally, the L298N was selected to perform this function. However, upon testing, the output of the H-bridge was not high enough to get the valve to fully open and close, likely due to protection circuitry for the motors. After further investigation, a more simplistic modified version of this board was found utilizing the MX1508 chip. This board is referred to as a "Mini L298N".

The voltage input of this Mini L298N is connected to 5 volts and ground on the ESP32 development board, which is connected to the Intel NUC via USB. The ESP32 GPIO pins 26 and 27 connect to the "IN1" and "IN2" on the Mini L2982N, while the valve is connected to the "MOTOR-A" output.

2) Software: A very simple python script controls this valve. Utilizing micro-python, the script can be run over serial via the "ampy" command. The script declares pins 26 and 27 and has functions for "open_valve" and "close_valve". "open_valve" turns pin 26 high for 1 second and then turns it low again. "close_valve" does the same thing with pin 27. The main part of the script runs "open_valve", waits for 8 seconds, then runs "close_valve."
To get the valve to open at the right time, a publisher is created in the executive node. When the turret locates a flame and begins oscillating, this publishes under the topic "/object_found." A subscriber then listens for this message from the "/object_found" topic. When it is received the micro-python script is run using the following command:

```
command = f'ampy --port {self.usb_port}
run /file/path/to/micro/python/script.py'
```

F. Gazebo Simulation

To verify functionality of any design, simulation is always the preferred first step. The .sdf file provided by Interbotix was edited to reflect the simulation properties needed for this robot system. The gazebo-bridge package must be properly configured for ROS2. This allows for topics to be shared freely between Gazebo and ROS2. When complete, the camera sensor and robot joints should come through as ROS topics
For this simulation, rather than image recognition through opencv, april tags are used to imitate locating a flame. After the april-tag package is properly configured in ROS2, an april-tag object is created in the .sdf file.
The logic for movement is largely the same as the logic discussed in the actual pan/tilt system. The only difference being the messages sent and received to control the joints and april-tag recognition rather than image recognition.

III. Results

Overall, the robot fire firefighter functions as intended. When initialized, the software waits for the signal from the IR sensor. When received, the camera begins scanning the area, by continuously panning. When a flame is found using OpenCV's image recognition, the water jet is aimed at the flame and projects water for 8 seconds.

A. Computer

The Intel NUC provided an excellent host for the main software functions. The only issues observed were some slow response times using the OpenCV tool in the flame recogntion. Some ways to alleviate this would be to either choose a different computer or install an external high powered graphics card for faster processing power.

B. IR Sensor

There was difficulty in using the wireless communication from micro-ros for the IR sensor. The issue observed was in repeatability. When the ESP32 is initialized and the micro-ros package begins on the main computer, connection is established and ros messages are sent and received properly.

However, when the system completes it's functionality, the ESP32 and main network must be completely shut down and reset to allow for proper re-initialization. This could likely be alleviated with more robust firmware, however, documentation surrounding this functionality in micro-ros is lacking.

C. Pan-Tilt Robot

The Interbotix camera turret performed properly and mostly without any observable issues. One minor issue observed was with USB port assignment. When testing, the USB port would randomly get reassigned causing there to be no movement.

D. Image Processing

The limitation for the image processing was left to two elements, namely, the camera sensor and the algorithm. The camera sensor is obviously not a thermal-imaging camera which forced the software to carry most of the burden image recognition. Recognition of the flame relied heavily on brightness. Very well lit areas where light fixtures are in view caused false positives. However, with the low light and bright conditional if statement, this was greatly improved.

E. Fire Suppression

The water ejected from the sprayer nozzle was sufficient in dousing small flames. One issue observed was with the mounting of the nozzle to the top of the camera. Being fixed loosley caused there to be some error in trajectory.

IV. Conclusion

A. Future Modifications

There are several specific areas of improvement that are important next steps in the continuation of this design.
Firstly, it will be critical to upgrade the IR sensor firmware, for a more robust, repeatable, and reliable sensor reading.
Next, Image Recognition needs improvement. A thermal imaging camera has the potential to significantly improve this system. However, if cost prohibits this upgrade, a more sophisticated algorithm for flame detection could be equally valuable
The fire suppression system for this prototype is rudimentary at best. A more effective, non-toxic fire retardant would greatly improve this system. However if water is to remain the chosen fire retardant, the nozzle will need a more reliable method for being fixed to the turret.

B. Results and Expectations

The expected outcome was achieved. Though, not flawless, the robot managed to successfully utilize all the properties mentioned here to extinguish a flame. Improvements will need to be made for this system to become as robust and reliable as needed in emergency situations, but prototyping is considered a success.