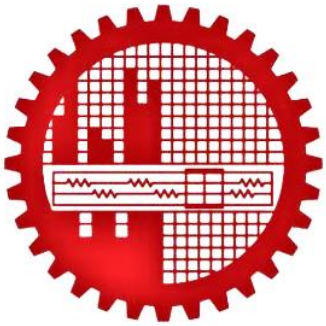
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY



DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING, BUET

Course No : EEE 318
Course Title : Control System I Laboratory
Submission Date: 27 February 2023

Submitted to

Ramit Dutta
Lecturer
Department of EEE
BUET

Mohammad Ali
Lecturer(PT)
Department of EEE
BUET

Submitted by

Md. Ashiqul Haider Chowdhury	1806086	Department of EEE Section: B1 Group: 07 Level-3 term-2
Fazle Rabbi	1806087	
Md. Ayenul Azim Jahin	1806089	
Indrojit Sarkar	1806090	

SmartFire Car: Image Processing-Based Fire Detection and Extinguishing for Enhanced Safety and Efficiency

Md. Ashiqul Haider Chowdhury[1], Fazle Rabbi[1], Md. Ayenul Azim Jahin[1], Indrojit Sarkar[1]

[1] student, Dept of Electrical And Electronic Engineering, BUET

ABSTRACT

The SmartFire Car is a novel approach to fire detection and extinguishing that utilizes image processing for enhanced safety and efficiency. Unlike traditional fire detection systems that rely on smoke or flame sensors, the SmartFire Car uses the Haar cascade algorithm for fire detection and an ESP32-CAM module to capture and analyze images. This image-based approach enables the SmartFire Car to detect fires at a greater range and with greater accuracy. The system also includes two NRF24L01+ modules for wireless communication between the car and a remote PC. The movement of the car is determined by processing data on the PC, and then sending movement commands to the NRF24L01+ module connected to the car. In addition to fire detection, the SmartFire Car is equipped with a fire extinguisher system that can be remotely activated to suppress the fire. The system uses two Arduinos for control and communication. This report details the design and implementation of the SmartFire Car, and explores potential future improvements, such as detecting wild animals and prioritizing extinguishing efforts in areas with endangered wildlife.

Keywords: Image processing, Haar cascade algorithm, ESP32-CAM, Arduino, NRF24L01, Wild animal detection.

I. INTRODUCTION

Wildfires are a significant threat to both human and animal life, as well as property and natural resources. To combat this issue, the development of a fire extinguishing system that can detect and extinguish fires quickly and efficiently is crucial. The traditional methods of fire detection, such as smoke or flame sensors, have proven to be unreliable, may not detect fires accurately, especially in outdoor environments and it can take a considerable amount of time for the fire department to reach remote locations. To tackle these challenges, we have developed the SmartFire Car, which uses image processing-based fire detection for enhanced safety and efficiency.

The SmartFire Car uses an ESP32-CAM module to capture real time images of the environment and employs the Haar cascade algorithm for fire detection. This system eliminates the need for traditional flame and smoke sensors and increases the detection range, making it ideal for extinguishing forest fires or fires in other locations where cars can easily access. In addition, the SmartFire Car can detect the presence of wild animals and prioritize the fire extinguishing process accordingly.

This report provides a detailed description of the SmartFire Car, including its hardware components and software algorithms. The report also presents the results of our experiments, which demonstrate the system's ability to accurately detect and extinguish fires. The SmartFire Car has the potential to significantly enhance safety and efficiency in firefighting operations, and we hope that this report will inspire further research and development in this area.

II. RELATED WORK

Automated fire extinguisher robo-cars have been developed in recent years to aid in fire suppression and rescue missions. A variety of approaches have been used to create these robots. In the field of fire extinguishing robots, several studies have been conducted to develop effective fire-fighting systems. In one study, an infrared flame sensor and a high-pressure water pump were used to create a fire extinguishing robot. Another study presented the design of a robot that utilized a smoke sensor and a flame sensor for fire detection, along with a water pump for extinguishing fires. Similarly, a mobile robot was developed for fire extinguishing purposes, which employed a flame sensor and a water tank, and was remotely controlled. However, our SmartFire Car uses an ESP32cam module for fire detection and an NRF24L01+ module for wireless communication. These related works provide valuable insight into the development and implementation of fire-fighting robots.

III. METHODOLOGY

Hardware Components

The SmartFire Car was built using the following hardware components:

- Arduino Mega
- Arduino Uno
- L293D motor shield
- NRF24L01+ wireless transceiver module (x2)
- ESP32-CAM module
- Four 3.7V batteries
- Breadboard and jumper wires

Fire Detection

The fire detection algorithm used in the SmartFire Car is the Haar Cascade algorithm. Haar Cascade algorithm works by identifying features in an image by comparing them to a set of trained patterns. In the case of fire detection, we used patterns that represent the edge and color properties of fire. The algorithm detects the presence of these patterns in the image to determine whether a fire is present or not. Although Haar Cascade algorithm has relatively lower accuracy compared to some other fire detection algorithms, it was chosen for its ease of implementation and efficiency.
To improve the accuracy of the fire detection algorithm, future work could explore the use of other image processing techniques such as YOLO, ViLD, CLIP/CLIPort.

Distance Calculation

Once a fire is detected in the video feed, the next step is to determine its distance from the camera. This is done by using the known distance between the camera and the object and the concept of focal length to estimate the distance. The code captures the video feed using OpenCV's VideoCapture method and detects the fire using the Haar Cascade classifier, which is trained on the fire detection dataset. Once the fire is detected, its position is determined by drawing a bounding box around it, which provides information about its x and y coordinates, width, and height. To estimate the distance, the focal length of the camera is calculated using the known distance and width of the object. This value is then used to estimate the distance of the fire from the camera.

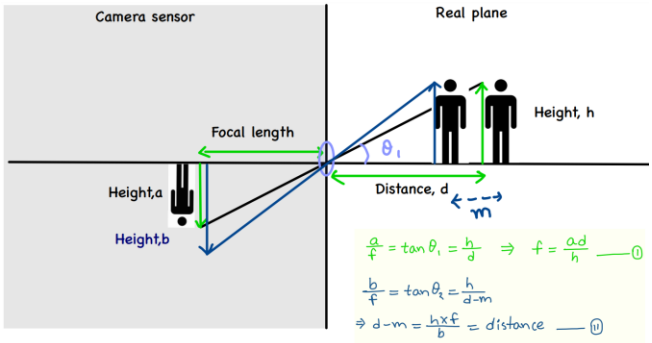


Figure 1: To measure the distance of a fire from the SmartFire Car, a reference image with a known width and distance from the camera is required. Using this reference image, the focal length is calculated through formula (i). To calculate the distance of a live fire image, formula (ii) is used where b represents the height of the virtual image of the camera lens and m represents the distance of the live image from the reference image.

The information of calculated distance and direction of the fire is then used to control the movement of the SmartFire Car. The car is controlled by sending a combination of commands to its Arduino Mega controller, including moving forward, moving backward, turning left, turning right, and stopping. These commands are sent as an array of numbers using the NRF module.

Controlling the car:

The movement of the SmartFire Car is controlled using a straightforward logic that takes into account the distance and direction of the fire. The logic follows a set of predefined rules. For instance, if the distance between the camera and the fire is more than a safe distance, the SmartFire Car moves forward towards the fire (direction = 1). In case the fire is located on the left (direction = 3) or right side (direction = 4) of the bounded box, the SmartFire Car turns in the corresponding direction. Once the car reaches the fire, it stops (direction = 0) and uses its built-in fire extinguisher to extinguish the fire. The logic used in this project is straightforward yet effective, ensuring the successful extinguishing of the fire in a controlled and efficient manner.

Figure 2: The image in the top-left corner depicts the SmartFire Car moving forward, while the image in the top-right corner shows it moving to the right. The bottom-left image shows the SmartFire Car moving to the left, and the bottom-right image shows it moving backward.

The algorithm uses the values obtained from the distance measurement process to determine the optimal direction of movement for the SmartFire Car. The movement of the car

is synchronized with the captured images, and commands (assigned numbers for particular direction) are sent to the car via the wireless communication module (NRF24L01+).

Wireless Communication

The wireless communication module used in this project is the NRF24L01, which is a low-power wireless transceiver that operates on the 2.4 GHz ISM band. We chose this module because it provides a reliable wireless connection and has a low power consumption, which makes it a suitable choice for our application. Furthermore, we implemented capacitors to optimize the performance of the NRF24L01+ and reduce the interference from other wireless communication technologies, such as Wi-Fi, RF, mobile networks.

The NRF24L01+ module is connected to a personal computer and communicates with another NRF24L01+ module linked to the Arduino Mega controller of the SmartFire Car. This communication is used to transmit commands for controlling the car's movements. The commands are sent as an array of numbers and include

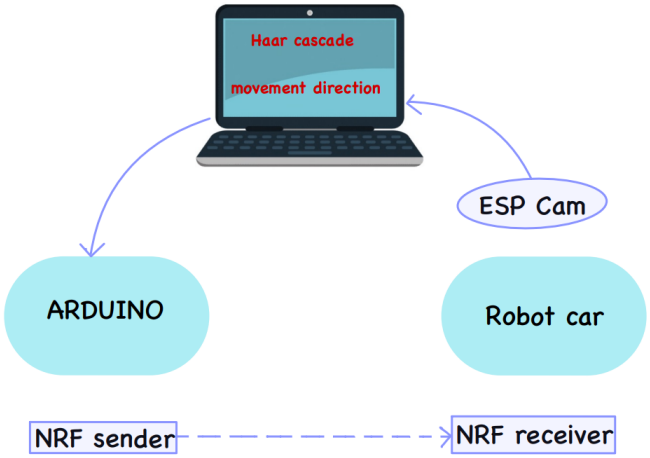


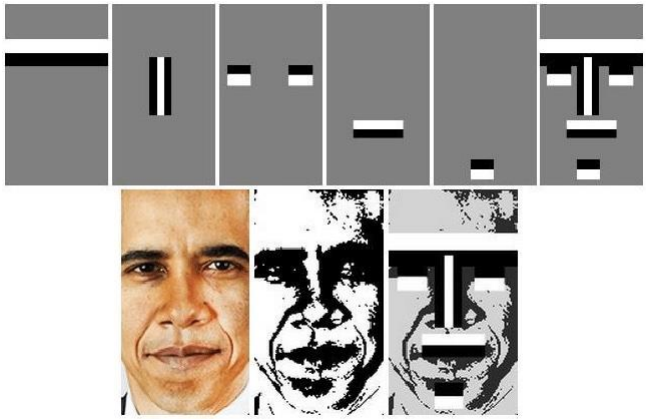
Figure 3: Right side of the fig a SmartFire car with an attached ESP32CAM that sends real-time videos to a remote PC using its built-in Wi-Fi module. An Arduino is connected to the PC via USB cable, and the PC processes the video to send movement information to the Arduino. The Arduino then wirelessly transmits this information to the SmartFire car using an NRF24L01+ module.

instructions for moving forward, moving backward, turning left, turning right, and stopping. Upon receiving these commands, the Arduino Mega controller processes them and sends the appropriate signals to the motors of the SmartFire Car to achieve the desired movement.

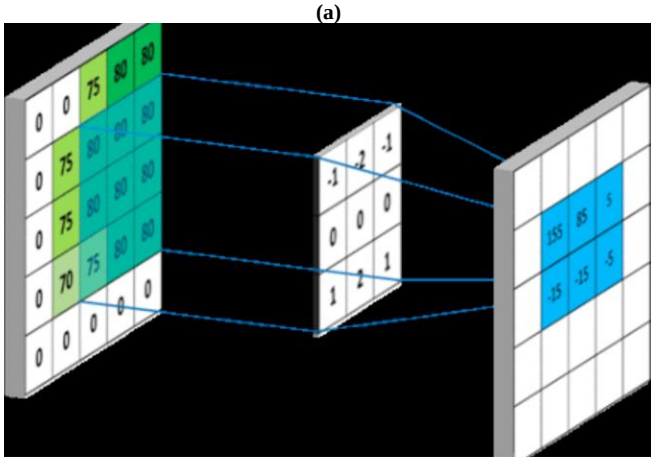
IV. SOFTWARE & HARDWARE DETAILS

Haar Cascade:

The Haar Cascade [1] algorithm is a machine learning-based object detection technique that can be used to detect fires in images or video. The algorithm works by first training a classifier on a set of positive and negative examples. The positive examples are images of fires, while the negative examples are images of non-fire scenes. The classifier learns to distinguish between these two types of images based on a set of features that are defined by Haar-like rectangles. Once the classifier is trained, it can be used to detect fires in new images or video frames. The algorithm works by sliding a rectangular window over the image and applying the classifier to each window. If the classifier determines that the window contains a fire, it is labeled as a positive detection.



(a)



(b)

Figure 4: (a) The figure illustrates the results obtained of a classifier during training.
(b) Convolution used in Haar Cascade classifier

To improve the performance of the algorithm, a sliding window approach is typically used, where the size of the window is gradually increased to detect fires at different

scales. Additionally, the algorithm may incorporate other techniques such as non-maximum suppression to remove overlapping detections and improve the overall accuracy of the algorithm. Also we fine tune this classifier using fire images. Overall, the Haar Cascade algorithm can be a powerful tool for detecting fires in images or video, but its performance depends on the quality of the training data and the specific features used to distinguish fires from non-fire scenes.

ATmega328P microcontroller:

The ATmega microcontroller is a family of microcontrollers developed by Atmel Corporation (now a part of Microchip Technology). The ATmega microcontrollers are widely used in embedded systems and are particularly popular in the Arduino platform.

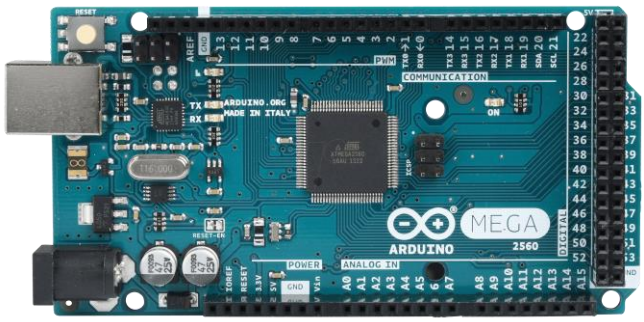


Figure 5: Arduino Mega Board

The Arduino Mega is a microcontroller board based on the ATmega2560 microcontroller. It has 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, and a power jack. The Arduino Mega is designed for more complex projects that require more input/output pins and more program memory compared to the smaller Arduino boards. It is widely used in robotics, automation, and other electronic projects. The ATmega microcontroller and the Arduino platform have made it easier for hobbyists, students, and professionals to develop and prototype electronic projects with ease and flexibility.

ESP32cam:

The ESP32cam is a compact module that combines an ESP32 microcontroller, an OV2640 camera module, and a built-in Wi-Fi antenna. The ESP32 microcontroller is a powerful and versatile system-on-chip (SoC) that features dual-core processing, Bluetooth connectivity, and a wide range of input/output interfaces. The OV2640 camera module is capable of capturing high-quality images and video at resolutions up to 1600x1200 pixels. With its built-in Wi-Fi antenna, the ESP32cam can connect to a wireless network and transmit images and video over the internet.

NRF24L01+:

NRF24L01+ is a low-cost 2.4 GHz wireless transceiver module designed for short-range communication. It is developed by Nordic Semiconductor and is widely used for wireless communication in many applications. The module operates in the ISM band and has a range of up to 100 meters with a data rate of up to 2 Mbps. NRF24L01 is a popular choice for wireless communication in projects that require low power consumption and reliability. It is compatible with many microcontrollers, including Arduino, and can be easily interfaced with them.

V. RESULT AND ANALYSIS

The SmartFire Car was designed to detect fires and extinguish them in a safe and efficient manner. The use of an ESP32cam module for fire detection provided a reliable and accurate method for detecting fires. The SmartFire Car was able to move towards the fire using a straightforward logic that took into account the distance and direction of the fire. This logic ensured that the SmartFire Car moved in the correct direction and at an appropriate speed to reach the fire quickly and safely.

In addition to the fire detection and movement capabilities, the SmartFire Car was also equipped with a built-in fire extinguisher that allowed it to quickly and effectively extinguish fires. The fire extinguisher was triggered automatically once the SmartFire Car reached the fire, ensuring that the fire was extinguished as quickly as possible.

However, further analysis and testing of the SmartFire Car would be beneficial to evaluate its performance in different scenarios. For instance, it would be useful to test the SmartFire Car's ability to detect and extinguish fires of varying sizes and locations. It would also be valuable to evaluate the response time and accuracy of the fire detection and extinguishing process to ensure that the SmartFire Car can effectively extinguish fires in a timely and safe manner.

Overall, the SmartFire Car provides a promising solution for fire extinguishing that can be further developed and optimized with additional analysis and testing.

VI. APPLICATION

the SmartFire Car can be used in various environments where it can easily access, such as forests, fields, and open spaces. Its mobility and fire extinguishing capabilities make

it a useful tool for firefighting in areas that are difficult for humans to access or where human intervention may be too dangerous. Additionally, the SmartFire Car's ability to detect fires and extinguish them in their early stages can help prevent fires from spreading and causing significant damage to the environment.

VII. CONSIDERATIONS TO PUBLIC HEALTH AND SAFETY

When designing a fire extinguishing robot system, considerations to public health and safety are of utmost importance. Here are some key points to consider:

Use of safe and non-toxic extinguishing agents: The extinguishing agent used in the robot should be safe for humans and the environment. For example, using water as an extinguishing agent can be an effective and safe option.

Minimizing the risk of fire spreading: The robot should be designed to minimize the risk of fire spreading during operation. For example, the robot should have mechanisms to prevent the spread of flames or sparks.

Ensuring the safety of the robot operator and other personnel: The robot should be designed to operate safely, and should have safety features such as emergency stop buttons or sensors to detect obstacles or people in its path.

Adherence to safety standards and regulations: The design and operation of the robot should comply with relevant safety standards and regulations to ensure the safety of the public and the environment.

Risk assessment and contingency planning: Before deploying the robot, a thorough risk assessment should be conducted to identify potential hazards and risks. Contingency plans should be developed in case of any unforeseen events, such as malfunctioning of the robot or failure to extinguish the fire.

By considering these factors, a fire extinguishing robot system can be designed and operated in a safe and effective manner, minimizing risks to public health and safety.

VIII. CONSIDERATIONS TO ENVIRONMENT

When designing and operating a fire extinguishing robot system, it is important to consider its potential impact on the environment. Here are some environmental considerations to keep in mind:

Water usage: High-pressure water pumps can use a lot of water to extinguish fires, so it is important to make sure that the robot is not wasting water or causing water damage to surrounding areas.

Chemicals and pollutants: Some fire extinguishing agents may contain chemicals or pollutants that can be harmful to the environment. It is important to choose extinguishing agents that are environmentally friendly and safe for use in the surrounding area.

Noise pollution: The operation of the robot and its high-pressure water pump can create a significant amount of noise, which can be disruptive to nearby residents and wildlife.

Disposal of waste water: After extinguishing a fire, the water and other extinguishing agents used by the robot need to be properly disposed of. Careful consideration should be given to how this waste water will be contained and disposed of, so as to avoid contamination of the surrounding environment.

Overall, the design and operation of a fire extinguishing robot system should aim to minimize its impact on the environment while effectively and efficiently extinguishing fires.

X. FUTURE SCOPE

Integration of artificial intelligence (AI) algorithms: The system could be further improved by incorporating AI algorithms to enhance its fire detection and extinguishing capabilities.

Addition of advanced sensors: Advanced sensors such as thermal cameras and gas sensors could be added to the system for more accurate fire detection and localization.

Deployment in hazardous environments: The system could be used in hazardous environments such as chemical plants, oil refineries, and nuclear power plants, where human intervention is risky.

Implementation of autonomous navigation: The system could be made fully autonomous, where it can navigate through a building or a plant on its own and extinguish fires.

Integration with other systems: The system could be integrated with other fire safety systems such as fire alarms and sprinkler systems to provide a comprehensive fire safety solution.

Other computer vision model: Use GPT-3 to generate captions or descriptions for the images captured by smartFire car. This could be useful for providing additional information about the fire or the surroundings that might be difficult for the computer vision system to detect on its own.

Use ViLD to improve the accuracy of fire detection by training a model on a larger dataset of fire images. This could potentially help the system detect fires in a wider variety of environments and under different lighting conditions.

Use CLIP or CLIPort to improve the ability of computer vision system to detect living beings behind a fire. These models have shown promising results in zero-shot object detection, meaning they can detect objects that were not seen during training. This could potentially help the system detect

living beings even when they are partially or fully obscured by flames.
Overall, incorporating these models into the project could help improve the accuracy and capabilities of smartFire car.

12. REFERENCES

[1]https://www.researchgate.net/publication/303251696_Evaluation_of_Haar_Cascade_Classifiers_for_Face_Detection

[2] Jones, C.D., A.B. Smith, and E.F. Roberts, *Book Title*, Publisher, Location, Date.

Appendix:

Arduino code:

```
// Adding necessery Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <AFMotor.h>
#include <SoftwareSerial.h>
#include <string.h>
#include <Servo.h>


#define MAX_DISTANCE 300              // sets maximum useable sensor measuring distance to 300cm
#define MAX_SPEED 110                 // sets speed of DC traction motors to 150/250 or about 70% of full speed - to get power drain down.
#define MAX_SPEED_OFFSET 10           // this sets offset to allow for differences between the two DC traction motors
AF_DCMotor leftMotor1(1, MOTOR12_1KHZ);   // create motor #1 using M1 output on Motor Drive Shield, set to 1kHz PWM frequency
AF_DCMotor leftMotor2(2, MOTOR12_1KHZ);   // create motor #2, using M2 output, set to 1kHz PWM frequency
AF_DCMotor rightMotor1(3, MOTOR34_1KHZ);  // create motor #3, using M3 output, set to 1kHz PWM frequency
AF_DCMotor rightMotor2(4, MOTOR34_1KHZ);  // create motor #4, using M4 output, set to 1kHz PWM frequency

Servo myservo;


String motorSet = "";
int speedSet = 0;

int condition;
int LED = 13;
int pump = 30;


RF24 radio(24, 26);             // nRF24L01 (CE, CSN)
const byte address[6] = "node1";  // Address

unsigned long LastReceivedTime = 0;
unsigned long CurrentTime = 0;

void setup() {
  Serial.begin(9600);
  delay(2000);
  myservo.attach(9);
  myservo.write(115);  // Angle of rotation
  delay(2000);


  radio.begin();  // Stratig the radio
  // setting up Radio Perameters
  radio.openReadingPipe(0, address);

  radio.setPALevel(RF24_PA_MAX);
  radio.startListening();  //  Set the module as receiver
  pinMode(LED, OUTPUT);
  pinMode(pump, OUTPUT);

  delay(1000);
}

void loop() {

  if (radio.available()) {
    int direction = 0;
    radio.read(&direction, sizeof(direction));

    LastReceivedTime = millis();

    if (direction == '0') {
      moveStop();
      delay(50);
      //     myservo.write(30);


    } else if (direction == '1') {
      moveForward();
      delay(20);
    }

    else if (direction == '2') {
      moveBackward();
      delay(20);
```

```
}

else if (direction == '3') {
  unsigned long time_now = 0;
  int period = 100;
  //moveBackward();
  //delay(20);
  moveStop();
  delay(20);
  turnLeft();
  time_now = millis();
  while (millis() < time_now + period) {
    if (direction != '3') {
      break;
    }
  }

}

else if (direction == '4') {
  unsigned long time_now = 0;
  int period = 100;
  //moveBackward();
  //delay(20);
  moveStop();
  delay(20);
  turnRight();
  time_now = millis();
  while (millis() < time_now + period) {
    if (direction != '4') {
      break;
    }
  }

}

else if (direction == '5') {
  moveStop();
  delay(20);
  digitalWrite(pump, HIGH);
  delay(10);
  if (direction == 0) {
    digitalWrite(pump, LOW);
    digitalWrite(LED, LOW);
  }
    delay(100);

    digitalWrite(LED, HIGH);
    myservo.write(50);  // Rotation
    delay(20);
    //     yield();
    myservo.write(115);


  }

  else {
    moveStop();
    delay(50);
  }
 }
}
void moveStop() {
  leftMotor1.run(RELEASE);
  leftMotor2.run(RELEASE);
  rightMotor1.run(RELEASE);
  rightMotor2.run(RELEASE);
}  // stop the motors.
//-----------------------------------------------------------------
-------------------------------------------------
void moveForward() {
  motorSet = "FORWARD";
  leftMotor1.run(FORWARD);                      // turn it
on going forward
  leftMotor2.run(FORWARD);                      // turn it
on going forward
  rightMotor1.run(FORWARD);                      // turn it
on going forward
  rightMotor2.run(FORWARD);                      // turn it
on going forward
  for (speedSet = 0; speedSet < MAX_SPEED; speedSet +=
2)  // slowly bring the speed up to avoid loading down the
batteries too quickly
  {
    leftMotor1.setSpeed(speedSet);
    leftMotor2.setSpeed(speedSet);
    rightMotor1.setSpeed(speedSet);
    rightMotor2.setSpeed(speedSet);
    delay(5);
  }
```

```
}
//-----------------------------------------------------------------------
-------------------------------------------------------
void moveBackward() {
  motorSet = "BACKWARD";
  leftMotor1.run(BACKWARD);                    // turn
it on going backward
  leftMotor2.run(BACKWARD);                    // turn
it on going backward
  rightMotor1.run(BACKWARD);                    // turn
it on going backward
  rightMotor2.run(BACKWARD);                    // turn
it on going backward
  for (speedSet = 0; speedSet < MAX_SPEED; speedSet +=
2)  // slowly bring the speed up to avoid loading down the
batteries too quickly
  {
    leftMotor1.setSpeed(speedSet);
    leftMotor2.setSpeed(speedSet);
    rightMotor1.setSpeed(speedSet);
    rightMotor2.setSpeed(speedSet);
    delay(5);
  }
}
//-----------------------------------------------------------------------
-------------------------------------------------------
void turnRight() {
  rightMotor1.run(BACKWARD);  // turn motor 3 backward
  rightMotor2.run(BACKWARD);  // turn motor 4 backward
  rightMotor1.setSpeed(speedSet +
MAX_SPEED_OFFSET);
  rightMotor2.setSpeed(speedSet +
MAX_SPEED_OFFSET);
  delay(100);  // run motors this way for 500
  motorSet = "FORWARD";
  leftMotor1.run(FORWARD);  // set both motors back to
forward
  leftMotor2.run(FORWARD);
  rightMotor1.run(FORWARD);
  rightMotor2.run(FORWARD);
  moveStop();
}
//-----------------------------------------------------------------------
-------------------------------------------------------
void turnLeft() {
  motorSet = "LEFT";
  leftMotor1.run(BACKWARD);  // turn motor 1 backward
  leftMotor2.run(BACKWARD);  // turn motor 2 backward
  leftMotor1.setSpeed(speedSet + MAX_SPEED_OFFSET);
  leftMotor2.setSpeed(speedSet + MAX_SPEED_OFFSET);
  rightMotor1.run(FORWARD);  // turn motor 3 forward
  rightMotor2.run(FORWARD);  // turn motor 4 forward
  delay(100);               // run motors this way for 1500
  motorSet = "FORWARD";
  leftMotor1.run(FORWARD);   // turn it on going forward
  leftMotor2.run(FORWARD);   // turn it on going forward
  rightMotor1.run(FORWARD);  // turn it on going forward
  rightMotor2.run(FORWARD);  // turn it on going forward
}
```

Python code:

```
# Loading Necessary Modules
import cv2
import serial # Allow us to Communicate with Arduino
import time
# variables
# distance from camera to object(fire) measured
x, y , h , w = 0,0 ,0 ,0
DISTANCE=0

Known_distance = 35 # Inches
#mine is 14.3 something, measure your fire width, are
google it
Known_width=3 #Inches
# Colors  >>> BGR Format(BLUE, GREEN, RED)
GREEN = (0,255,0)
RED = (0,0,255)
BLACK = (0,0,0)
YELLOW =(0,255,255)
PERPEL = (255,0,255)
WHITE = (255,255,255)

fonts = cv2.FONT_HERSHEY_COMPLEX
fonts2 = cv2.FONT_HERSHEY_SCRIPT_SIMPLEX
fonts3 =cv2.FONT_HERSHEY_COMPLEX_SMALL
fonts4 =cv2.FONT_HERSHEY_TRIPLEX

url = "http://172.20.10.2:81/stream"
cap = cv2.VideoCapture(url)
Distance_level =0
# fire detector object
fire_detector =
cv2.CascadeClassifier("fire_detection_cascade_model.xml")
# focal length finder function
def FocalLength(measured_distance, real_width,
width_in_rf_image):
    focal_length = (width_in_rf_image* measured_distance)/
real_width
    return focal_length

# distance estimation function
def Distance_finder (Focal_Length, real_fire_width,
fire_width_in_frame):

    distance = (real_fire_width *
Focal_Length)/fire_width_in_frame

    return distance

#fire detection Fauction
def fire_data(image, CallOut, Distance_level):
    # Function Discrption (Doc String)
    '''
    This function Detect fire and Draw Rectangle and display
the distance over Screen

    :param1 Image(Mat): simply the frame
    :param2 Call_Out(bool): If want show Distance and
Rectangle on the Screen or not
    :param3 Distance_Level(int): which change the line
according the Distance changes(Intractivate)
    :return1  fire_width(int): it is width of fire in the frame
which allow us to calculate the distance and find focal length
    :return2 fire(list): length of fire and (fire paramters)
    :return3 fire_center_x: fire centroid_x coordinate(x)
    :return4 fire_center_y: fire centroid_y coordinate(y)

    '''
    fire_width = 0
    fire_x, fire_y =0,0
    fire_center_x =0
    fire_center_y =0
#    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
#convert image to grayscale
    minSize=(30, 30),
    # flags=cv2.cv.CV_HAAR_SCALE_IMAGE
    fires = fire_detector.detectMultiScale(image, 1.2, 5) #
better detection at scaling factor 1.21/ conumse more cpu.
    for (x, y, h, w) in fires:
        # cv2.rectangle(image, (x, y), (x+w, y+h), BLACK, 1)
        fire_width = w
        fire_center=[]
        # Drwaing circle at the center of the fire
        fire_center_x =int(w/2)+x
        fire_center_y =int(h/2)+y
        if Distance_level <10:
            Distance_level=10

        # cv2.circle(image, (fire_center_x, fire_center_y),5,
(255,0,255), 3 )
        if CallOut==True:
            LLV = int(h*0.12)
             # print(LLV)
            line_thickness =2

            # cv2.rectangle(image, (x, y), (x+w, y+h), BLACK,
1)
            cv2.line(image, (x,y+LLV), (x+w, y+LLV),
(GREEN),line_thickness)
            cv2.line(image, (x,y+h), (x+w, y+h),
(GREEN),line_thickness)
            cv2.line(image, (x,y+LLV), (x, y+LLV+LLV),
(GREEN),line_thickness)
            cv2.line(image, (x+w,y+LLV), (x+w, y+LLV+LLV),
(GREEN),line_thickness)
            cv2.line(image, (x,y+h), (x, y+h-LLV),
(GREEN),line_thickness)
```

```
        cv2.line(image, (x+w,y+h), (x+w, y+h-LLV),
(GREEN),line_thickness)

        cv2.line(image, (x,y), (fire_center_x,fire_center_y ),
(155,155,155),1)
        cv2.line(image, (x,y-11), (x+210, y-11),
(YELLOW), 25)
        cv2.line(image, (x,y-11), (x+Distance_level, y-11),
(GREEN), 25)

        cv2.circle(image, (fire_center_x, fire_center_y),2,
(255,0,255), 1 )
        cv2.circle(image, (x, y),2, (255,0,255), 1 )

      # fire_x = x
      # fire_y = y

   return fire_width, fires, fire_center_x, fire_center_y

# reading reference image from directory
ref_image = cv2.imread("imgg6.png")

ref_image_fire_width,_, _,_= fire_data(ref_image, False,
Distance_level)
Focal_length_found = FocalLength(Known_distance,
Known_width, ref_image_fire_width)
print(Focal_length_found)

# cv2.imshow("ref_image", ref_image)
# Setting up Arduino For Communication
Arduino = serial.Serial(baudrate=9600, port = 'COM6')
# variable for Arduino Communication
Direction =0
# Max 0 and Min 255 Speed of Motors
Motor1_Speed =0 # Speed of motor Accurding to PMW
values in Arduino
Motor2_Speed =0
Truing_Speed =110
net_Speed =180

while True:
   _, frame = cap.read()
   frame_height, frame_width, _ = frame.shape
   # print(frame_height, frame_width)
   # calling fire_data function
   # Distance_leve =0
   RightBound = frame_width-140
   Left_Bound =140
   fire_width_in_frame,Fires ,FC_X, FC_Y=
fire_data(frame, True, Distance_level)
   # finding the distance by calling function Distance finder
   if fire_width_in_frame ==0:
      # Writing The motor Speed
         Motor1_Speed=0
         Motor2_Speed=0
         # Direction of movement
         Direction=0
         print("stop")

         cv2.line(frame, (50,55), (200, 55), (BLACK), 15)
         cv2.putText(frame, f"stop", (50,58), fonts,0.4,
(PERPEL),1)
   else:
      for (fire_x, fire_y, fire_w, fire_h) in Fires:

         if fire_width_in_frame !=0:
            Distance = Distance_finder(Focal_length_found,
Known_width,fire_width_in_frame)
            Distance = round(Distance,2)
            # Drwaing Text on the screen
            Distance_level= int(Distance)
            cv2.line(frame, (50,33), (130, 33), (BLACK), 15)
            cv2.putText(frame, f"Robot State", (50,35),
fonts,0.4, (YELLOW),1)
            # cv2.line(frame, (50,65), (170, 65), (BLACK),
15)

            # Direction Decider Condition

               # cv2.line(frame, (50,65), (170, 65), (BLACK),
15)
               # cv2.putText(frame, f"Truing = False",
(50,70), fonts,0.4, (WHITE),1)
            if FC_X<Left_Bound:
               # Writing The motor Speed
               Motor1_Speed=Truing_Speed
               Motor2_Speed=Truing_Speed
               print("Left Movement")
               # Direction of movement
               Direction=3
               cv2.line(frame, (50,65), (170, 65), (BLACK),
15)
               cv2.putText(frame, f"Move Left {FC_X}",
(50,70), fonts,0.4, (YELLOW),1)

            elif FC_X>RightBound:
               # Writing The motor Speed
               Motor1_Speed=Truing_Speed
               Motor2_Speed=Truing_Speed
               print("Right Movement")
               # Direction of movement
               Direction=4
               cv2.line(frame, (50,65), (170, 65), (BLACK),
15)
               cv2.putText(frame, f"Move Right {FC_X}",
(50,70), fonts,0.4, (GREEN),1)

            elif Distance >30 and Distance<=1000:
               # Writing The motor Speed
               Motor1_Speed=net_Speed
```

```
            Motor2_Speed=net_Speed
            # Direction of movement
            Direction=1
            cv2.line(frame, (50,55), (200, 55), (BLACK),
15)
            cv2.putText(frame, f"Forward Movement",
(50,58), fonts,0.4, (PERPEL),1)
            print("Move Forward")

         #elif Distance >5 and Distance<=15 :
          #   # Writing The motor Speed
           #  Motor1_Speed=net_Speed
            # Motor2_Speed=net_Speed
            ## Direction of movement
            #Direction=2
            #print("Move Backward")
            #cv2.line(frame, (50,55), (200, 55), (BLACK),
15)
            #cv2.putText(frame, f"Backward Movement",
(50,58), fonts,0.4, (PERPEL),1)
         elif Distance >15 and Distance<=30:
            # Writing The motor Speed
            Motor1_Speed=net_Speed
            Motor2_Speed=net_Speed
            # Direction of movement
            Direction=5
            print("stop and pump water")
            cv2.line(frame, (50,55), (200, 55), (BLACK),
15)
            cv2.putText(frame, f"stop and pump water",
(50,58), fonts,0.4, (PERPEL),1)

         else:
            # Writing The motor Speed
            Motor1_Speed=0
            Motor2_Speed=0
            # Direction of movement
            Direction=0
            cv2.line(frame, (50,55), (200, 55), (BLACK),
15)
            cv2.putText(frame, f"No Movement and rotate
servo", (50,58), fonts,0.4, (PERPEL),1)

         cv2.putText(frame, f"Distance {Distance} Inch",
(fire_x-6,fire_y-6), fonts,0.6, (BLACK),2)
         data = f"{Direction}" #A233B233D2
         print(data)
         # Sending data to Arduino
         Arduino.write(data.encode()) #Encoding the data
into Byte fromat and then sending it to the arduino
         time.sleep(0.002) # Providing time to Arduino to
Receive data.
         Arduino.flushInput() #Flushing out the Input.

   cv2.line(frame, (Left_Bound, 80), (Left_Bound, 480-80),
(YELLOW), 2)
   cv2.line(frame, (RightBound, 80),(RightBound, 480-80),
(YELLOW), 2)
   cv2.imshow("frame", frame )

   if cv2.waitKey(1)==ord("q"):
      break

cap.release()
cv2.destroyAllWindows()
```