Early Wildfire Detection System

By

Edward Hernandez, Kirsten Lau, Thomas Lubet

Senior Project

DEPARTMENT OF ELECTRICAL ENGINEERING

California Polytechnic State University, San Luis Obispo

San Luis Obispo

Table of Contents

List of Figures

Acknowledgements

Thank you to all Electrical Engineering Professors for the past years of teaching at Cal Poly, and a special thanks to our advisor, John Saghri.

Abstract

Fires ravaged over 4 million acres and thousands of structures across California during 2020[1]. California is prone to fires due to dry flatland and scarce rainfall every year. Fires cause damage to not only natural habitats, but also communities near the fire. Rebuilding taxes the local communities as well as state and federal resources. Stopping fires before they start is crucial to protecting ecosystems as well as lives. Currently, manned towers are the system in place today to detect fires. This project aims to detect early signs of fire through visual and infrared image capture in an automated process.

Introduction

Problem

Wildfires are the cause of much destruction to people, wildlife, and property. The most effective way of mitigating fire damage is through early detection. Fires consist of the smoke plume, heat plume, and fire core. The fire core is the source of the fire and the flames. The heat plume is the heated air surrounding the fire core. The smoke plume rises above the fire core and heat plume, and is cold but visible on the visible light spectrum. To automate the detection of fire, the best way is to utilize the visible and infrared light spectrum.

Our Approach

The visible light spectrum can detect signs of smoke, but not the heat from the fire. Using only the visible spectrum gives rise to more problems. Clouds or movement from trees could trigger false positive fire detection. Since fires are serious, there would need to be a manual check of each of these detections. If there are many false detections, it wastes time and resources to check each detection. The infrared light spectrum can detect the heat from the fire core and the heat plume. But at farther distances, the fire would have to be very large for the infrared spectrum to detect. This would delay the extinguishing of the fire, and by then the fire could be raging out of control.

This detection system aims to use both infrared and visible spectrums of light. The infrared spectrum can show the fire core and heat plume. The visible spectrum shows the smoke plume. By using both spectrums, there are more opportunities to detect the early signs of a fire. This paper documents the different aspects of the detection system prototype.

Software

Flowchart

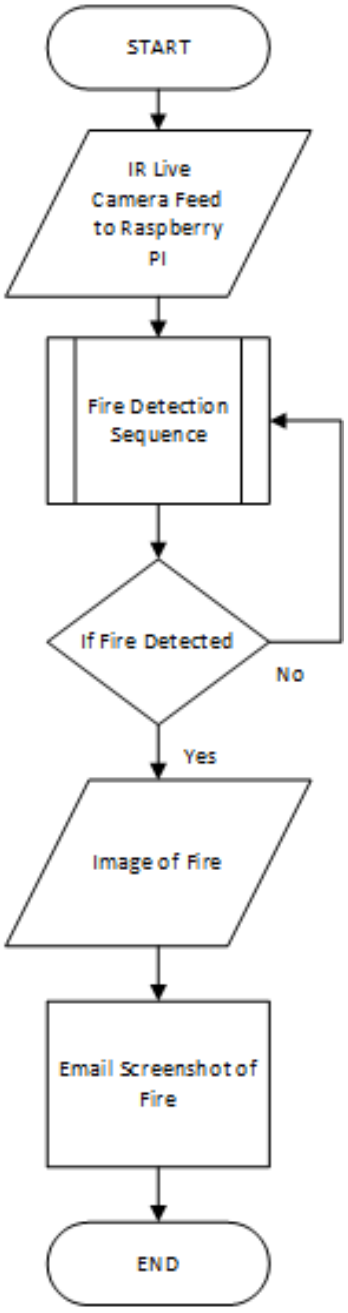

Figure 1. Fire detection algorithm flowchart

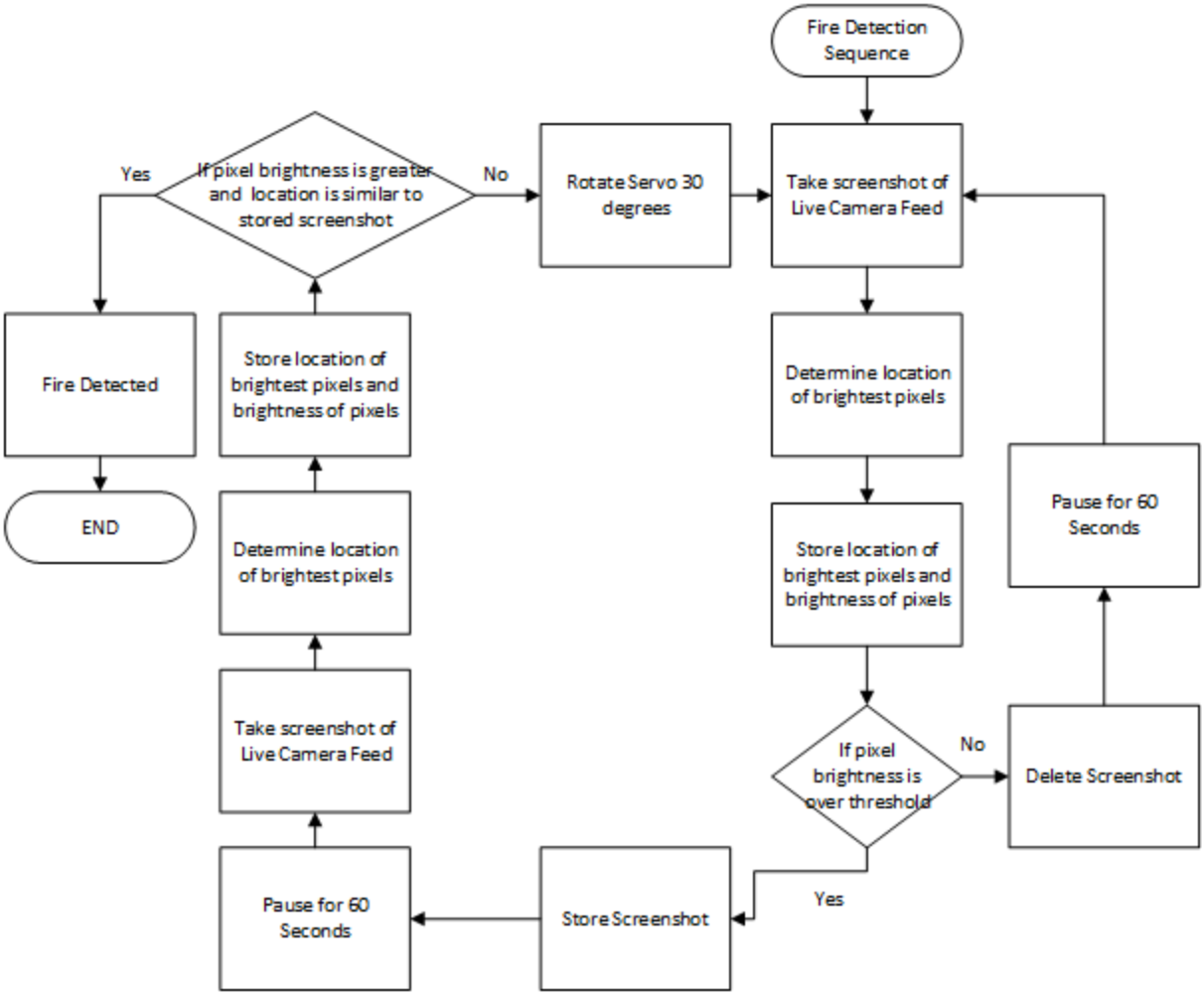

Figure 2. Fire detection sequence flowchart

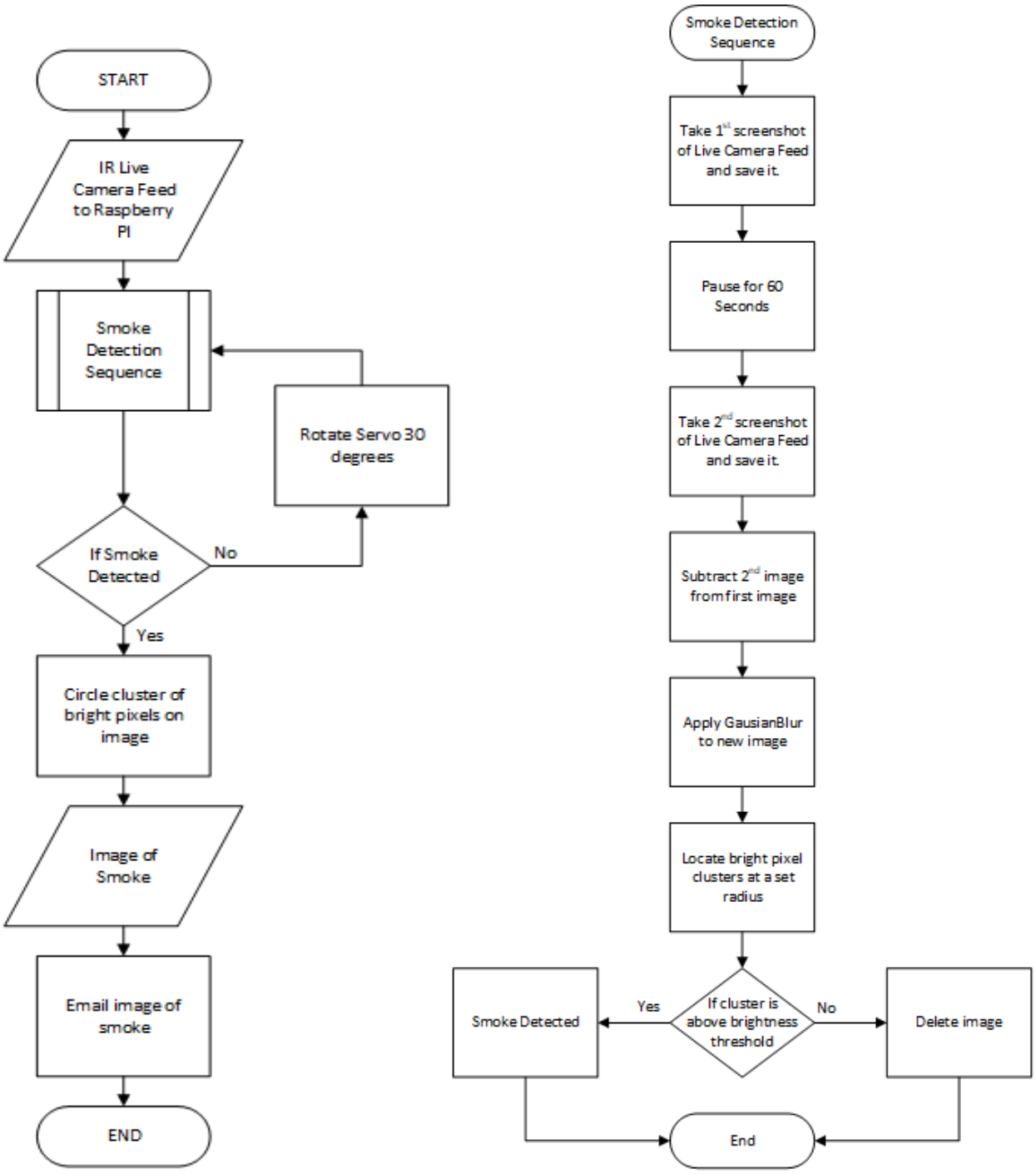

Figure 3a and 3b. Smoke detection algorithm and Smoke detection sequence flowchart

Our Approach

There are two different algorithms used in our prototype. One for each spectrum utilized to detect fire. This is due to how the camera we designed our prototype around works. The camera cannot change between light spectrums via software code. The camera switches between

spectrums via a button on the camera. For our purposes, we are demonstrating the feasibility of our design.

INFRARED FIRE DETECTION SYSTEM

The Raspberry Pi takes in live infrared camera footage of the surrounding area. It then screenshots the footage for image processing. The Raspberry Pi uses OpenCV functions to determine the location of the brightest pixels. Then it saves both the location and the brightness of the pixels. The Raspberry Pi then compares the brightness of the pixels to determine if they are above a certain threshold. If the pixels are above the threshold, the Raspberry Pi stores the image. If they are below, the Raspberry Pi discards the image. After 60 seconds, the Raspberry Pi repeats the process of capturing the live feed and finding and comparing the pixel brightness to get a second image. The Raspberry Pi compares the location and brightness of the bright pixels of the first image to the second. If the location is similar and the brightness is greater, the Raspberry Pi marks this as a positive detection. On a detection, it circles the cluster that passes over the threshold and has the highest brightness. "Fire Detected" is also added to the corner of the image. The second image is then saved into a file path as well as emailed to an email that was set up to receive the images. The output of this algorithm is the image of the fire, and the email with the image.

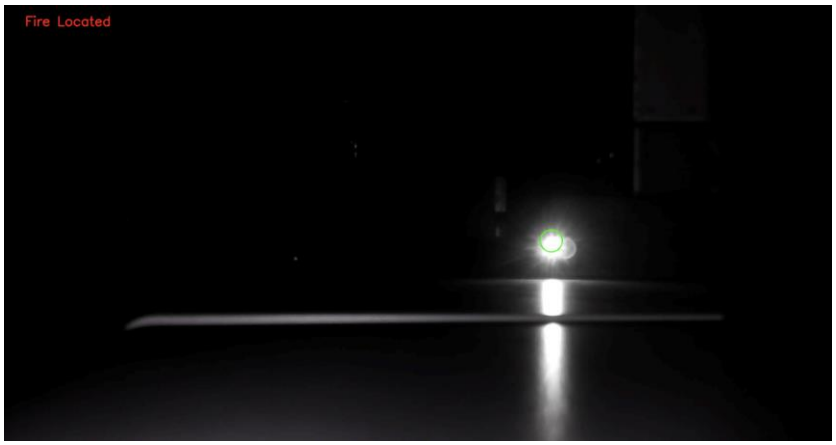

Figure 4. Annotated infrared capture of a fire output by the system.

VISUAL SMOKE DETECTION

The Raspberry Pi takes in a visual light spectrum camera feed to detect smoke. Like with the fire detection algorithm, the Raspberry Pi takes a screenshot for image processing. It will

then take a second screenshot 60 seconds later. Using OpenCV functions, the first image then subtracts the second image to create a new image. This is image differencing. Gaussian blur is then applied to the new image to smooth the image. Then the Raspberry Pi locates the brightest pixel clusters. If the cluster brightness surpasses a set threshold, the Raspberry Pi marks this as a positive detection. On detection, it circles the cluster that passes over the threshold and has the highest brightness. "Smoke Detected" is also added to the corner of the image. The image is then saved into a file path as well as emailed to an email that was set up to receive the images.

Figure 5a and 5b. Captures from the camera live feed of no smoke and with smoke.

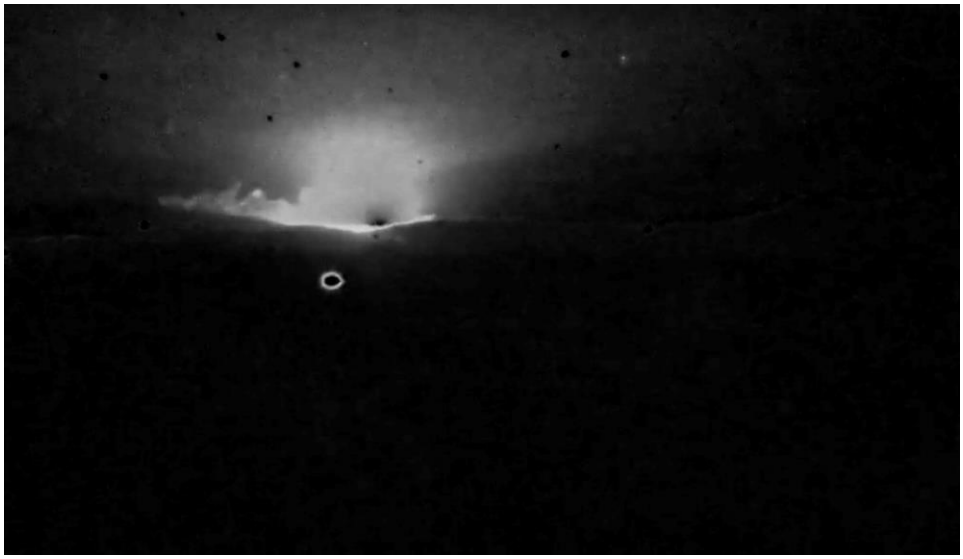

Figure 6. Resulting differenced image of the captures from the camera live feed of no smoke and with smoke.

Figure 7. Annotated capture of a positive detection of smoke output by the system.

Verdict

The design and implementation of the autonomous wildfire detection system used both infrared and visual light spectrums. By taking the live camera feed and taking screenshots, the process of analysis is easier. The processed image is then sent in an email upon fire detection. The delay between each capture is easy to change within the software because of this method. Each image is also saved to keep a record of the surrounding area. This record of images does mean that storage will be a problem for the detection system. But there is enough storage to have the system running for a long time before it becomes an issue. If this system is operating on a constant basis, the required power is intensive.

There are other issues with this approach. If the camera is not stable, the results for smoke detection will not be accurate. Any shakiness will show up in the image differencing and may cause a false positive. Some shakiness may also be due to the Servo motor that rotates the camera. The Servo allows for a 360-degree rotation of the camera so that it surveys the surrounding area. This full-circle motion also allows for the detection system to be on at all times. But the motion is not smooth, as the motor turns the camera 30 degrees at a time.

Hardware

Image Data Flowchart

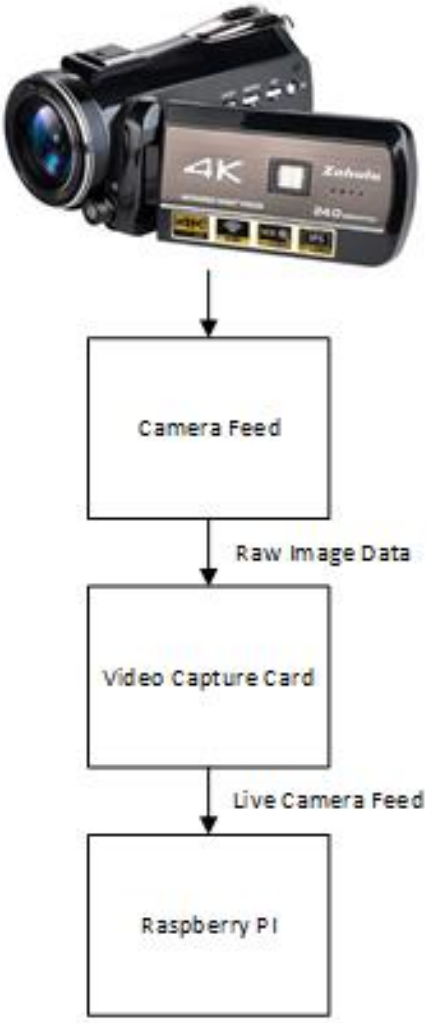

Figure 8. Video data flowchart from the camera to the Raspberry Pi.

Camera

The prototype this paper outlines uses the Zohulu 4K Wifi Full Spectrum Camcorder. We chose this camera for both the infrared and visual spectrum capabilities it provides. In this prototype, we use both spectrums. The visual spectrum for the detection of smoke and visual signs of fire. IR capabilities for significant heat signatures that may be behind foliage. Because the camera had to detect objects very far away, it needed a very high resolution to provide clear images. The 24MP met that need. Finally, the difficulty of outputting the video feed to another

device influenced our choice. The camera has a built-in HDMI port that, along with a video card, makes obtaining the feed easy and compact.

Figure 9. Zohulu 4K Wifi Full Spectrum Camcorder used in the prototype.

Raspberry PI

We chose the Raspberry Pi because of its flexibility and power for such a small device. It contains a microprocessor with 8 gigabytes of RAM used for processing image data in a small amount of time. There are several ports for other devices, including USB, HDMI, and GPIO pins, to control motors or other equipment. The Raspberry Pi is also very popular for prototyping. Thus, it has a large amount of online documentation and support to reference for every aspect of the project. The Raspberry Pi only needs 5V to run, so it does not use large amounts of power. Adding more storage or other modules to the Raspberry Pi can increase functionality and tailor it to the project.

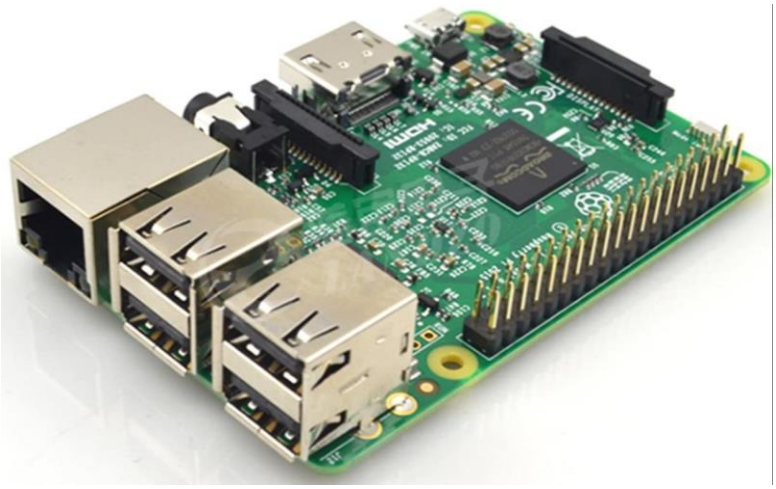

Figure10. Raspberry Pi 3 used as the microprocessor in this prototype.

Servo Motor and Mounting

A servo motor controls the angle of the camera. It rotates about 30° to catch different angles of a location and is not fixed on a single point. The Raspberry Pi controls the servo and sends the command to rotate every 30 seconds. The servo is reliable to keep the camera rotating for long periods of time.

Each device sits on a 3D printed base that attaches to a camera tripod. The servo is attached to the tripod with the rotating base on top of the servo. The baseplate had grooves for straps to hold down the camera and holes for screws to hold down the Raspberry Pi.

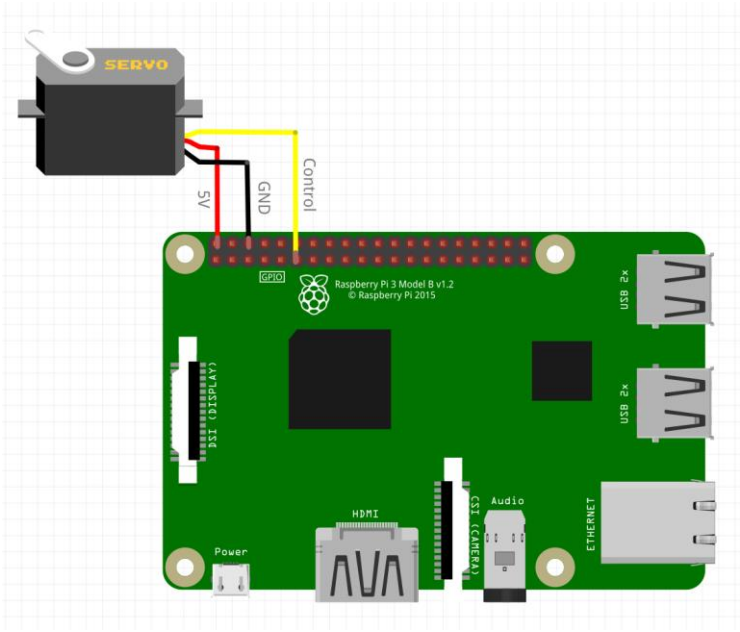

Figure 11. Servo connection to Raspberry PI via GPIO pins.

Results

Conclusion

Transporting the camera feed into the Raspberry Pi was the most important achievement because there would be no way of identifying any signs of a fire without it. It does not read continuous images but rather takes screenshots at certain intervals. This method decreases the processing power demand needed to completely scan the images. The second achievement is the rotating base for the camera. This allows the camera to capture images all around the tripod. Although the base can rotate by itself, the wires connecting to the servo limited the completely free range of the servo. To circumvent the issue, for the servo to go back to its starting position, it must rotate backward 330° so the wires do not tangle up. The next achievement is the fire detection through IR coming from the camera. This was the easiest to set up because we could find a brighter area in the screen capture while also not finding false positives if there were more than one entity in question. While testing with several bright spots, the fire was positively identified every time. The final achievement was the visual identification of smoke. It was the trickiest to test because of the image collection, but it still works as intended.

Next Step

The next steps for the autonomous fire detection system are necessary if this system is to go out in the field. First, the system needs to not rely on wired power if it is to sit on fire-prone hills and mountainous areas. The least power needed would be 5V at 3A to power the Raspberry PI. The camera has its own battery that needs the occasional charge so it does not need constant power. Next, the fire detection algorithm and smoke detection algorithm need to work in tandem for accurate fire detection. This means that we need a camera that can change from infrared to visible light spectrum via software. This way, the two algorithms can combine into one and then send only one email. Next, programming the Servo motor to move in a way that does not shake the camera. This will help in the detection of smoke as any movement will show in the image differencing. The most important step is testing the detection system once the previous modifications are in place. Both fire and smoke detection need further testing in both controlled and real-world situations. Controlled, for testing the full extent of the detection system's capabilities. And real-world situations to see how well the detection works as well as see if there are other issues that need addressing. Once the testing is complete, designing and building a

better way to mount the camera, Raspberry PI, and Servo motor to the top of the tripod is next. Finally, the housing. The housing will allow the system to be weatherproof and stay outdoors and be effective. These modifications and tests will show us how effective our autonomous fire detection system is at detecting fires. How effective the system is will determine if a camera-based fire detection system is a viable solution to fire prevention.

Appendix A - Senior Project Design Analysis

Summary of Functional Requirements

The Early Wildfire Detection System has several useful capabilities and functions. For this to be an effective system it will have to first be able to communicate with the appropriate authorities and other cameras about positions of suspicious anomalies detected by the camera. The communication function of the EWFDS is via a cellular connection through the raspberry pi that can reach cell towers 20 miles away. Detection brings another capability of the camera system which includes taking in visual data through two cameras, one camera is to be a vision camera to see smoke and the other camera will be an IR camera to detect heat anomalies. The microcontroller Raspberry Pi receives the processed data to understand the state of the surroundings of the camera system. This brings the next capabilities of seeing the surrounding of the camera, ideally, up to a mile in clear conditions with 360 degrees of view by mounting the cameras on a 360-degree servo/motor. Next, when a camera detects a fire or smoke it must be able to contact the appropriate authorities in a timely fashion. Timely fashion meaning under 2 minutes from detection and confirmation of an anomaly the authorities are contacted and aware of the specific position.

Primary Constraints

The significant challenges we faced revolved around power, communication, location, and performance. The EWFDS uses AC power, power poses a challenge because of any disturbances that might cause outages in our device, and we want to keep the device from failing or turning off at any moment. We wanted to integrate a processor that communicates in a variety of ways because the remote locations for the device do not always have a strong cellular signal available. The locations of the devices might be very harsh in weather and with the temperature importance on electrical equipment, we wanted to make sure that the device was safe from extreme temperatures. The performance was difficult to quantify because we wanted a 4k resolution, a low-powered camera that would take pristine images and video for processing.

Economic

Preventing the spread of wildfires will protect the surrounding area; which includes homes, businesses, as well as natural resources from destruction. This saves millions of dollars

on suppressing fires as well as billions of dollars in property damage [2]. Not to mention, the damage done to local ecosystems and natural resources, which can take decades to recover from [3].

If manufactured on a commercial basis

In the manufacturing of this product, the estimations on devices sold per year, cost per device, the price for each device, profit per year, and cost-efficiency. An estimated thousand devices are sold in a year due to the range of the system and how much area needs surveillance. In addition, using the cost for parts and estimating the cost for labor, an estimated $850 per unit for manufacturing each device. Likewise, the estimated cost for purchase price for each device factors in the cost of designing and writing the code for the device and the manufacturing cost per device; and is an estimated $1100 to buy each device. Now to estimate the profit margin for the EWFDS, the price sold subtracted from the manufacturing cost per unit and multiplied by the number of units sold, comes out to $250,000. Estimated utility cost ranges from 68.4 to 79.2 kWh and labor for device installation. Utility price for each kWh will depend on location and distributor price.

Environmental

This project aims to detect wildfires as they are starting, and notify local authorities who can prevent the spread of the wildfire. Preventing the spread of a wildfire will help protect the surrounding ecosystems which can take several decades to recover from a bad fire. With the ongoing drought in California, no time for the ecosystem to recover before the next dry season. This drought means an increasing amount of dry brush that can catch fire. With this project, wildfires can be detected early and prevented quickly, thus allowing more time for the surrounding areas to recover and grow. The EWFDS does use many natural resources to produce the cameras, servos, and mobile phone used as well as create more of the associated pollution caused in manufacturing these products. But the EWFDS aims to reduce the loss of natural resources from wildfires as well as from rebuilding after a fire, not to mention preventing the air pollution from the smoke and ashes.

Manufacturability

This project designs a functioning wildfire detection system that will automatically detect fires and smoke. If/when this product is placed for use in the field, the manufactured product should be reliable and resilient to the environment surrounding it. One area of concern in the manufacturing of this product is the servo motors and their reliability. When exposed to the elements and looking at all the parts going into this project the most delicate part would be the servos. Two steps to minimize the damage to the servos include, making sure that the enclosure also insulates and protects the servo from the environment. The problem with expanding the enclosure is that it can make the enclosure more complicated and harder to manufacture via 3D printing as there is a limit to how big the 3D printer can print. The second way to minimize damage to the servo is to pay more for a higher quality servo, but the problem is that a more expensive servo will cause the cost of the project to go up and with the same budget stress increases and forces cheaper components in other areas.

Sustainability

Cal Poly defines sustainability as "the ability of the Natural and Social Systems to survive and thrive together to meet Current and Future Needs"[4]. The EWFDS will have to cover the three spheres of sustainability: Environmental, Economic, and Social. The EWFDS does not use any natural resources in the build, but all the parts can be repurposed for other functions, such as the raspberry pi and the two camera systems. Economic impacts include a decrease in damage to urban areas done by wildfires and profit margins in the production of this product. With the EWFDS in place, socially, the quality of life will increase due to a safer living environment.

Ethical

The device could invade the privacy of others. The purpose of the device is to capture video to then process the information which would mean to store the video data. The device could record someone while in their home if placed in a commercial area prone to fire, or into someone's car. This is concerning, as the IEEE code of ethics 7.8.I.1 [5] states that the safety and welfare of the public, as well as the privacy of others, is to be protected. The NSPE code of ethics also states: "Engineers shall not reveal facts, data, or information without the prior consent of the client or employer except as authorized or required by law or this Code"[2].

Health and Safety

Few associated health and safety concerns arise with using this project. One concern is installing the device. The device may be difficult to transport and install at remote locations on hillsides, or atop utility poles and towers. But using this project will help in preventing wildfires, which can help decrease air and water pollution due to smoke and ash.

Social and Political

This project impacts those who live in areas heavily impacted by wildfires. Those who live and work in these areas are notified of a wildfire as early as possible so that they can protect their property and/or safely evacuate by local authorities. Also, knowledge of this camera system will positively impact local and surrounding communities by making them feel safer with an extra pair of eyes watching out for them. The EWFDS is not intended as a commercially available product but could have a positive impact on stakeholders if there becomes a demand for the product in the market. While not commercially available, local governments in fire-prone areas would be able to purchase EWFDS to better protect the public. As a result, the EWFDS would be subject to local rules and regulations as well as public opinion.

Development

Digital image processing handling manages control of digital images through a digital computer. Generally, image processing contributes to building up a computer system that can perform processing on an image. Digital image processing includes robust intelligent systems, robots are in automation made altogether by looking into researchers in different parts of the world. It incorporates advancements in different digital image processing applications [6]. The part of the project that we construct contains the learning we have not completed yet. No experimentation to reflect on yet.

Appendix ## Python Code

Functions Code

```
def fireCheck (maxLoc, maxVal):
    import pyautogui
    import os, os.path
    import cv2
    import numpy as np
    import argparse
    import time

    #set variables
    #radius stays the same
    radius = 25
    omaxLoc = maxLoc
    #new threshold value has to be greater or equal to past
    thrVal = maxVal

    #delay between second screenshot that compares
    print('wait started')
    time.sleep(1)
    print('wait ended')
    path2 = '/home/pi/Desktop/WildfireProject/Screenshot'
    index = len(os.listdir(path2))
    path =
r'/home/pi/Desktop/WildfireProject/Screenshot/screenshot%d.png'
% (index)
    im = pyautogui.screenshot(region = (5,70, 1915, 1000))

    #saves new screenshot
    im.save(path)
```

```
    #converts image so openCV tools can read it
    image = cv2.imread(path)
    imageGray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    imageGray = cv2.GaussianBlur(imageGray, (radius, radius), 0)
    (nminVal, nmaxVal, nminLoc, nmaxLoc) =
cv2.minMaxLoc(imageGray)

    #set new bounds so that if the fire got bigger thus moving
the center it will still pass
    #convert to numpy arrays so can use multiplication to set
bounds
    lower = np.array(nmaxLoc) * 0.75
    higher = np.array(nmaxLoc) * 1.25
    middle = np.array(nmaxLoc)

    #makes sure that this 'fire' is within a respectable bounds
from the previous one
    if nmaxVal >= thrVal:
        if lower[0] <= middle[0] <= higher[0]:
            if lower[1] <= middle[1] <= higher[1]:
                return(True)
            else:
                return(False)
        else:
            return(False)
    else:
        return(False)


def sendImage(path):
```

```
    #This Code Block is with reference to
https://realpython.com/python-send-email/
    #There is not much that can be changed from original code
    import email
    import smtplib
    import ssl

    from email import encoders
    from email.mime.base import MIMEBase
    from email.mime.multipart import MIMEMultipart
    from email.mime.text import MIMEText
    subject = "FIRE HAS BEEN DETECTED"
    body = "Fire Detection system :"
    sender_email = "cppthrowaway123@gmail.com"
    receiver_email = "tlubet.sf@gmail.com"
    password = input("Email Password")

    # Create a multipart message and set headers
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = subject
    message["Bcc"] = receiver_email  # Recommended for mass
emails

    # Add body to email
    message.attach(MIMEText(body, "plain"))

    filename = path

    # Open PDF file in binary mode
```

```
    with open(filename, "rb") as attachment:
        # Add file as application/octet-stream
        # Email client can usually download this automatically as
attachment
        part = MIMEBase("application", "octet-stream")
        part.set_payload(attachment.read())

    # Encode file in ASCII characters to send by email
    encoders.encode_base64(part)

    # Add header as key/value pair to attachment part
    part.add_header(
        "Content-Disposition",
        f"attachment; filename= {filename}",
    )

    # Add attachment to message and convert message to string
    message.attach(part)
    text = message.as_string()

    # Log in to server using secure context and send email
    context = ssl.create_default_context()
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context)
as server:
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, text)
    return()

def servo():
    import RPi.GPIO as GPIO
    import time
```

```
#ebables GPIO pins
GPIO.setmode(GPIO.BOARD)

# Set pin 11 as an output
GPIO.setup(11,GPIO.OUT)
servo1 = GPIO.PWM(11,50)

servo1.start(0)
time.sleep(1)
print('Camera is about to move')

servo1.ChangeDutyCycle(7.5)
time.sleep(0.1)
servo1.ChangeDutyCycle(0)
print('Camera is is in new position')

servo1.stop()
GPIO.cleanup()
```

Wildfire Detection

```
import pyautogui
import os, os.path
import cv2
import numpy as np
import argparse
import time
from function import *

i = 1
while i <= 2:
   #ap = argparse.ArgumentParser()
   #ap.add_argument('-i', '--index')
   #The following line of code can open the video input when pasted in another command window
   #ffplay -f v4l2 -input_format mjpeg -video_size 1920x1080 -framerate 30 -i /dev/video0

   #path of images that have bright enough pixels that could be fires
   path4 = '/home/pi/Desktop/WildfireProject/Fire'

   #index so a history of images can be accumulated to adjust code values for future
   index4 = len(os.listdir(path4))

   #file path of images that have been confirmed to be fires
   path3 = r'/home/pi/Desktop/WildfireProject/Fire/FireProof%d.png' % (index4)

   #This is the file path for all screenshots
```

```
    path2 = '/home/pi/Desktop/WildfireProject/Screenshot'
    #index of number of screenshots in path
    index = len(os.listdir(path2))

    path =
'/home/pi/Desktop/WildfireProject/Screenshot/screenshot%d.png' %
(index)

    #takes screenshot and saves it to screenshot filepath
    #                (left, top, right, bottom)
    im = pyautogui.screenshot(region = (5,70, 1915, 1000))
    im.save(path)

    print(f'Screenshot saved to path: {path}')

    files = len(os.listdir(path2))
    print('There are %d files in the path' % (files))

    #predetermined values that set the radius of bright pixel
clusters and threshold brightness values
    radius = 25
    thrVal = 161

    #allows for openCV functions to read saved screenshot images
    image = cv2.imread(path)
    #Grayscales image incase of png compression issues
    imageGray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #Applies gaussian smoothing so small bright pixels will now
skew results
    imageGray = cv2.GaussianBlur(imageGray, (radius, radius), 0)
    #finds maxValues and Location on image
```

```
    (minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(imageGray)

    #Checks to see if image max values is bright enough to
warrant a second check
    if maxVal >= thrVal:
        #calls for a function that double checks for a fire grown
after a set amount of time
        fire = fireCheck(maxLoc, maxVal)
        if fire == True:
            #uses opencv tools to circle fire spot and write in
top corner that fire was
            cv2.circle(image, maxLoc, radius, (0, 255, 0), 2)
            cv2.putText(image, 'Fire Detected', (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
            cv2.imwrite(path3, image)
            #emails image of fire to appropriate authorities (in
this case from one gmail to another)
            sendImage(path3)
            #cv2.imshow('Fire?', image)
            #cv2.waitKey(0)
        else:
            cv2.putText(image, 'No Fire Detected', (50,50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
    else:
        cv2.putText(image, 'No Fire Detected', (50,50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)

    print('maxVal', maxVal)
    print('minVal', minVal)
    print('maxLoc', maxLoc)
    print('minLoc', minLoc)
```

```
    servo()

print('exited while loop')
```

Smoke Detection

```
import cv2
import time
import os, os.path
import numpy as np
import pyautogui
from function import *

#Smoke Detection

       #/home/pi/Desktop/WildfireProject/Smoke
path0 = '/home/pi/Desktop/WildfireProject/Smoke/'
index = len(os.listdir(path0))
path1 = "/home/pi/Desktop/WildfireProject/Smoke/zzz%d.png" %
(index)
img1 = pyautogui.screenshot(region = (5,70, 1915, 1000))
img1.save(path1)

time.sleep(1)

index = len(os.listdir(path0))
path2 = '/home/pi/Desktop/WildfireProject/Smoke/zzz%d.png' %
(index)
img2 = pyautogui.screenshot(region = (5,70, 1915, 1000))
img2.save(path2)


index = len(os.listdir(path0))
path3 =
'/home/pi/Desktop/WildfireProject/Smoke/smokeProof%d.png' %
(index)
```

```
#for testing uncomment next 2 lines and comment out all the
image gathering above except path0, path3 and index above path3
# path1 = "/home/pi/Desktop/WildfireProject/Smoke/zzz.png"
# path2 = "/home/pi/Desktop/WildfireProject/Smoke/zzz3.png"

img1 = cv2.imread(path1)
img2 = cv2.imread(path2)

gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
#cv2.imshow('test', gray1)

gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

original = cv2.imread(path1)
copy = cv2.imread(path2)
difference = cv2.subtract(gray2, gray1)
score = cv2.absdiff(gray1, gray2)
print('Score:', score)
#cv2.imshow('diff', difference)
#cv2.imshow('Original', difference)
radius = 45
thrVal = 161
imageBlur = cv2.GaussianBlur(difference, (radius, radius), 0)
(minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(imageBlur)
if maxVal >= thrVal:
   cv2.circle(img2, maxLoc, radius, (0, 255, 0), 2)
   cv2.putText(img2, 'Smoke Detected', (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
   cv2.imwrite(path3, img2)
   sendImage(path3)
```

```
else:
    print('no smoke')
    print('maxVal', maxVal)
    print('minVal', minVal)
    print('maxLoc', maxLoc)
    print('minLoc', minLoc)

#cv2.waitKey(0)
#cv2.destroyAllWindows()
```

References

[1] B. Sridhar, “Applications of Digital Image Processing In Real Time World,” *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, vol. 8, no. 12.

[2] I. Arslan and A. Z. Alkar, "Real-time wildfire smoke detection on moving camera," 2014 22nd Signal Processing and Communications Applications Conference (SIU), Trabzon, 2014, pp. 1203-1206, doi: 10.1109/SIU.2014.6830451.

[3] California Department of Forestry and Fire Protection (CAL FIRE), “2020 Incident Archive,” Cal Fire Department of Forestry and Fire Protection. [Online]. Available: https://www.fire.ca.gov/incidents/2020/. [Accessed: 25-Feb-2021].

[4] K. Pierre-louis and J. Schwartz, “Why Does California Have So Many Wildfires?,” The New York Times, 21-Aug-2020. [Online]. Available: https://www.nytimes.com/article/why-does-california-have-wildfires.html. [Accessed: 25-Feb-2021].

[5] “Code of Ethics,” Code of Ethics | National Society of Professional Engineers, Jul-2019. [Online]. Available: https://www.nspe.org/resources/ethics/code-ethics. [Accessed: 14-Mar-2021].

[6] Z. Jian, J. Guo, X. Xu, B. Wang, Y. Yi, and H. Ye, "Risk Analysis of Wild Fire on Electric Grid Transmission Lines Considering Cascading Failure," 2019 IEEE 3rd Conference on Energy Internet and Energy System Integration (EI2), Changsha, China, 2019, pp. 872-876, doi: 10.1109/EI247390.2019.9061890.

[7] “4K Wifi Full Spectrum Camcorders, Ultra HD Infrared Night Vision Paranormal Investigation Video Camera with 60fps 24MP 30X Digital Zoom - Ghost Hunting Camera (with 2 batteries, 32GB SD card included),” *Amazon*. [Online]. Available: https://www.amazon.com/gp/product/B07BD3KJLV/ref=ox_sc_act_title_2?smid=A2LTZZ2EKFL29G&psc=1. [Accessed: 05-Apr-2020].

[8] “OpenCV Python Image Smoothing - Gaussian Blur,” *TutorialKart*, 30-Nov-2020. [Online]. Available: https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/. [Accessed: 20-Aug-2021].

[9] “OpenCV modules,” *OpenCV*. [Online]. Available: https://docs.opencv.org/3.4/index.html. [Accessed: 16-Jul-2021].

[10] J. de Langen, “Sending Emails with Python,” *Real Python*, 27-Feb-2021. [Online]. Available: https://realpython.com/python-send-email/. [Accessed: 30-Jul-2021].