Week 1 (08-08-25)

The team members working on this project are the following:

1. Anandhakrishnan V
2. Anjana Krishna
3. Anand J Nair
4. Ann Merin Rejo
5. Harigovind K S

PROBLEM STATEMENT : Develop a system to detect hotspots in coal mines by generating a detailed thermal map of the area and automating sprinklers to mitigate potential hazards effectively.

WHAT WAS DONE TODAY(07-08-25)

Discussed with Dr.Arun and Daniel sir at STEAG Rajagiri, got an overview of the project and how to segment it into different phases.

The first day was mostly spent researching the problem statement. We talked in detail about the problem at hand with experts at STEAG Rajagiri, during the process we cleared all our doubts and gained valuable insights on how the industry actually works as well the reason why thermal detection and prevention at a coal field is a challenging task.

After our discussion we set out to find possible solutions and technologies we can adapt for solving this problem. After careful consideration we came to the conclusion that the best approach is to use a thermal camera as the primary sensor for reading temperature data and accompanying the thermal camera with other sensors to make sure that we get accurate reading.

Since infrastructure and cost is a main issue, for now we can only think about drones, all of us are on the consensus that we should avoid and replace drones if possible due to its poor battery life and flight time. Since we don’t really have a plausible solution at hand right now, we’ve decided to take the drone as a default and think about other means to replace it later. Some other options we considered were gantry (3d printer style) top down system, spider-cams, rovers. The former two of the three required better infrastructure and thus would cost more money while the latter was slow and had limited coverage.

The next topic we researched was about mapping. For now we've decided to use a grid based mapping with gps and image stitching to get the hotspots.

A microprocessor like Raspberry Pi was chosen for processing the data captured by the thermal camera as well as other sensors and sending it to a base station. We intend to replace the Raspberry pi with a custom made board for better efficiency and performance.

By the end of the day we've got a solid foundation about the problem statement as well as the general route we want to take to finish this project. The plans and components are subjected to change in the future.

Week 2 (22/08/25)

The objective for the day was to get ourselves familiarized with the thermal camera and the software required.

Thermal Camera Setup on Windows (Camera: TOPDON TC001)
Our first task for the day was to read the user manual for the thermal camera and find out all its features and learn how to use it properly. After reading the document and manual we installed the driver software provided by the company on Windows OS for learning how to use it. With the help of the software we were able to get output from the camera, read the temperature, take pictures, videos etc. We also learned about different colour mapping modes as well as the resolution of the camera. We were already planning to use Linux OS for our project, but this step was crucial because we were able to identify the key features of the thermal camera by using the official software provided by the company. This step helped us to identify the key “Problem” of ambient temperature and distance which we’ll mention in detail later.

Configuring Linux for Future Deployment
The next step we did was configuring Linux OS, we had already installed the OS on a bootable USB drive to use it for the day but at the project center we faced some issues with booting Linux up. After some time and with a bit of tinkering the issue was swiftly solved. We then used the GitHub repo used by our seniors who were working on this project to interface with the camera.

GitHub link : https://github.com/leswright1977/PyThermalCamera

The user had mentioned in the page that the python script is a rough patch for taking output from the thermal camera and there could be errors. Just like the description, after using the code we were faced with errors. It took us some time to get the code in a working condition. The working code is pasted below for future use, there will be iterations to the code below in the future.

PyThermalCamera Script:

```
#!/usr/bin/env python3
'''
Les Wright 21 June 2023
https://youtube.com/leslaboratory
A Python program to read, parse and display thermal data from the Topdon TC001
Thermal camera!
'''
print('Les Wright 21 June 2023')
print('https://youtube.com/leslaboratory')
print('A Python program to read, parse and display thermal data from the Topdon TC001
Thermal camera!')
print('')
print('Tested on Debian all features are working correctly')
print('This will work on the Pi However a number of workarounds are implemented!')
print('Seemingly there are bugs in the compiled version of cv2 that ships with the Pi!')
print('')
print('Key Bindings:')
print('')
print('a z: Increase/Decrease Blur')
print('s x: Floating High and Low Temp Label Threshold')
print('d c: Change Interpolated scale Note: This will not change the window size on the
Pi')
print('f v: Contrast')
print('q w: Fullscreen Windowed (note going back to windowed does not seem to work
on the Pi!)')
print('r t: Record and Stop')
print('p : Snapshot')
print('m : Cycle through ColorMaps')
print('h : Toggle HUD')

import cv2
import numpy as np
import argparse
import time
import io
```

```
#We need to know if we are running on the Pi, because openCV behaves a little oddly on all the builds!
#https://raspberrypi.stackexchange.com/questions/5100/detect-that-a-python-program-is-running-on-the-pi
def is_raspberrypi():
    try:
        with io.open('/sys/firmware/devicetree/base/model', 'r') as m:
            if 'raspberry pi' in m.read().lower(): return True
    except Exception: pass
    return False

isPi = is_raspberrypi()

parser = argparse.ArgumentParser()
parser.add_argument("--device", type=int, default=0, help="Video Device number e.g. 0, use v4l2-ctl --list-devices")
args = parser.parse_args()

if args.device:
        dev = args.device
else:
        dev = 0

#init video
cap = cv2.VideoCapture('/dev/video'+str(dev), cv2.CAP_V4L)
#cap = cv2.VideoCapture(0)
#pull in the video but do NOT automatically convert to RGB, else it breaks the temperature data!
#https://stackoverflow.com/questions/63108721/opencv-setting-videocap-property-to-cap-prop-convert-rgb-generates-weird-boolean
if isPi == True:
        cap.set(cv2.CAP_PROP_CONVERT_RGB, 0.0)
else:
        cap.set(cv2.CAP_PROP_CONVERT_RGB, 0.0)

#256x192 General settings
width = 256 #Sensor width
height = 192 #sensor height
scale = 3 #scale multiplier
newWidth = width*scale
```

```
newHeight = height*scale
alpha = 1.0 # Contrast control (1.0-3.0)
colormap = 0
font=cv2.FONT_HERSHEY_SIMPLEX
dispFullscreen = False
cv2.namedWindow('Thermal',cv2.WINDOW_GUI_NORMAL)
cv2.resizeWindow('Thermal', newWidth,newHeight)
rad = 0 #blur radius
threshold = 2
hud = True
recording = False
elapsed = "00:00:00"
snaptime = "None"

def rec():
        now = time.strftime("%Y%m%d--%H%M%S")
        #do NOT use mp4 here, it is flakey!
        videoOut = cv2.VideoWriter(now+'output.avi', cv2.VideoWriter_fourcc(*'XVID'),25,
(newWidth,newHeight))
        return(videoOut)

def snapshot(heatmap):
        #I would put colons in here, but it Win throws a fit if you try and open them!
        now = time.strftime("%Y%m%d-%H%M%S")
        snaptime = time.strftime("%H:%M:%S")
        cv2.imwrite("TC001"+now+".png", heatmap)
        return snaptime


while(cap.isOpened()):
        # Capture frame-by-frame
        ret, frame = cap.read()
        if ret == True:
                imdata,thdata = np.array_split(frame, 2)
                #now parse the data from the bottom frame and convert to temp!

#https://www.eevblog.com/forum/thermal-imaging/infiray-and-their-p2-pro-discussion/20
0/
                #Huge props to LeoDJ for figuring out how the data is stored and how to
compute temp from it.
```

```
#grab data from the center pixel...
hi = thdata[96][128][0]
lo = thdata[96][128][1]
#print(hi,lo)
lo = lo*256
rawtemp = hi+lo
#print(rawtemp)
temp = (rawtemp/64)-273.15
temp = round(temp,2)
#print(temp)
#break

#find the max temperature in the frame
lomax = thdata[...,1].max()
posmax = thdata[...,1].argmax()
#since argmax returns a linear index, convert back to row and col
mcol,mrow = divmod(posmax,width)
himax = thdata[mcol][mrow][0]
lomax=lomax*256
maxtemp = himax+lomax
maxtemp = (maxtemp/64)-273.15
maxtemp = round(maxtemp,2)


#find the lowest temperature in the frame
lomin = thdata[...,1].min()
posmin = thdata[...,1].argmin()
#since argmax returns a linear index, convert back to row and col
lcol,lrow = divmod(posmin,width)
himin = thdata[lcol][lrow][0]
lomin=lomin*256
mintemp = himin+lomin
mintemp = (mintemp/64)-273.15
mintemp = round(mintemp,2)

#find the average temperature in the frame
loavg = thdata[...,1].mean()
hiavg = thdata[...,0].mean()
loavg=loavg*256
avgtemp = loavg+hiavg
```

```
                avgtemp = (avgtemp/64)-273.15
                avgtemp = round(avgtemp,2)



                # Convert the real image to RGB
                bgr = cv2.cvtColor(imdata,  cv2.COLOR_YUV2BGR_YUYV)
                #Contrast
                bgr = cv2.convertScaleAbs(bgr, alpha=alpha)#Contrast
                #bicubic interpolate, upscale and blur
                bgr =
cv2.resize(bgr,(newWidth,newHeight),interpolation=cv2.INTER_CUBIC)#Scale up!
                if rad>0:
                        bgr = cv2.blur(bgr,(rad,rad))

                #apply colormap
                if colormap == 0:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_JET)
                        cmapText = 'Jet'
                if colormap == 1:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_HOT)
                        cmapText = 'Hot'
                if colormap == 2:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_MAGMA)
                        cmapText = 'Magma'
                if colormap == 3:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_INFERNO)
                        cmapText = 'Inferno'
                if colormap == 4:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_PLASMA)
                        cmapText = 'Plasma'
                if colormap == 5:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_BONE)
                        cmapText = 'Bone'
                if colormap == 6:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_SPRING)
                        cmapText = 'Spring'
                if colormap == 7:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_AUTUMN)
                        cmapText = 'Autumn'
```

```
                if colormap == 8:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_VIRIDIS)
                        cmapText = 'Viridis'
                if colormap == 9:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_PARULA)
                        cmapText = 'Parula'
                if colormap == 10:
                        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_RAINBOW)
                        heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)
                        cmapText = 'Inv Rainbow'

                #print(heatmap.shape)

                # draw crosshairs
                cv2.line(heatmap,(int(newWidth/2),int(newHeight/2)+20),\
                (int(newWidth/2),int(newHeight/2)-20),(255,255,255),2) #vline
                cv2.line(heatmap,(int(newWidth/2)+20,int(newHeight/2)),\
                (int(newWidth/2)-20,int(newHeight/2)),(255,255,255),2) #hline

                cv2.line(heatmap,(int(newWidth/2),int(newHeight/2)+20),\
                (int(newWidth/2),int(newHeight/2)-20),(0,0,0),1) #vline
                cv2.line(heatmap,(int(newWidth/2)+20,int(newHeight/2)),\
                (int(newWidth/2)-20,int(newHeight/2)),(0,0,0),1) #hline
                #show temp
                cv2.putText(heatmap,str(temp)+' C', (int(newWidth/2)+10,
int(newHeight/2)-10),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 0, 0), 2, cv2.LINE_AA)
                cv2.putText(heatmap,str(temp)+' C', (int(newWidth/2)+10,
int(newHeight/2)-10),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 255, 255), 1, cv2.LINE_AA)

                if hud==True:
                        # display black box for our data
                        cv2.rectangle(heatmap, (0, 0),(160, 120), (0,0,0), -1)
                        # put text in the box
                        cv2.putText(heatmap,'Avg Temp: '+str(avgtemp)+' C', (10, 14),\
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1,
cv2.LINE_AA)
```

```
                cv2.putText(heatmap,'Label Threshold: '+str(threshold)+' C', (10,
28),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1,
cv2.LINE_AA)

                cv2.putText(heatmap,'Colormap: '+cmapText, (10, 42),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1,
cv2.LINE_AA)

                cv2.putText(heatmap,'Blur: '+str(rad)+' ', (10, 56),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1,
cv2.LINE_AA)

                cv2.putText(heatmap,'Scaling: '+str(scale)+' ', (10, 70),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1,
cv2.LINE_AA)

                cv2.putText(heatmap,'Contrast: '+str(alpha)+' ', (10, 84),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1,
cv2.LINE_AA)



                cv2.putText(heatmap,'Snapshot: '+snaptime+' ', (10, 98),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1,
cv2.LINE_AA)

                if recording == False:
                        cv2.putText(heatmap,'Recording: '+elapsed, (10, 112),\
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(200, 200, 200), 1,
cv2.LINE_AA)
                if recording == True:
                        cv2.putText(heatmap,'Recording: '+elapsed, (10, 112),\
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(40, 40, 255), 1,
cv2.LINE_AA)

                #Yeah, this looks like we can probably do this next bit more efficiently!
                #display floating max temp
                if maxtemp > avgtemp+threshold:
                        cv2.circle(heatmap, (mrow*scale, mcol*scale), 5, (0,0,0), 2)
                        cv2.circle(heatmap, (mrow*scale, mcol*scale), 5, (0,0,255), -1)
```

```
                        cv2.putText(heatmap,str(maxtemp)+' C', ((mrow*scale)+10,
(mcol*scale)+5),\
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0,0,0), 2, cv2.LINE_AA)
                        cv2.putText(heatmap,str(maxtemp)+' C', ((mrow*scale)+10,
(mcol*scale)+5),\
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 255, 255), 1,
cv2.LINE_AA)

                #display floating min temp
                if mintemp < avgtemp-threshold:
                        cv2.circle(heatmap, (lrow*scale, lcol*scale), 5, (0,0,0), 2)
                        cv2.circle(heatmap, (lrow*scale, lcol*scale), 5, (255,0,0), -1)
                        cv2.putText(heatmap,str(mintemp)+' C', ((lrow*scale)+10,
(lcol*scale)+5),\
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0,0,0), 2, cv2.LINE_AA)
                        cv2.putText(heatmap,str(mintemp)+' C', ((lrow*scale)+10,
(lcol*scale)+5),\
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 255, 255), 1,
cv2.LINE_AA)

                #display image
                cv2.imshow('Thermal',heatmap)

                if recording == True:
                        elapsed = (time.time() - start)
                        elapsed = time.strftime("%H:%M:%S", time.gmtime(elapsed))
                        #print(elapsed)
                        videoOut.write(heatmap)

                keyPress = cv2.waitKey(1)
                if keyPress == ord('a'): #Increase blur radius
                        rad += 1
                if keyPress == ord('z'): #Decrease blur radius
                        rad -= 1
                        if rad <= 0:
                                rad = 0

                if keyPress == ord('s'): #Increase threshold
                        threshold += 1
                if keyPress == ord('x'): #Decrease threashold
```

```
                threshold -= 1
                if threshold <= 0:
                        threshold = 0

        if keyPress == ord('d'): #Increase scale
                scale += 1
                if scale >=5:
                        scale = 5
                newWidth = width*scale
                newHeight = height*scale
                if dispFullscreen == False and isPi == False:
                        cv2.resizeWindow('Thermal', newWidth,newHeight)
        if keyPress == ord('c'): #Decrease scale
                scale -= 1
                if scale <= 1:
                        scale = 1
                newWidth = width*scale
                newHeight = height*scale
                if dispFullscreen == False and isPi == False:
                        cv2.resizeWindow('Thermal', newWidth,newHeight)

        if keyPress == ord('q'): #enable fullscreen
                dispFullscreen = True
                cv2.namedWindow('Thermal',cv2.WND_PROP_FULLSCREEN)

cv2.setWindowProperty('Thermal',cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FUL
LSCREEN)
        if keyPress == ord('w'): #disable fullscreen
                dispFullscreen = False
                cv2.namedWindow('Thermal',cv2.WINDOW_GUI_NORMAL)

cv2.setWindowProperty('Thermal',cv2.WND_PROP_AUTOSIZE,cv2.WINDOW_GUI_N
ORMAL)
                cv2.resizeWindow('Thermal', newWidth,newHeight)

        if keyPress == ord('f'): #contrast+
                alpha += 0.1
                alpha = round(alpha,1)#fix round error
                if alpha >= 3.0:
                        alpha=3.0
```

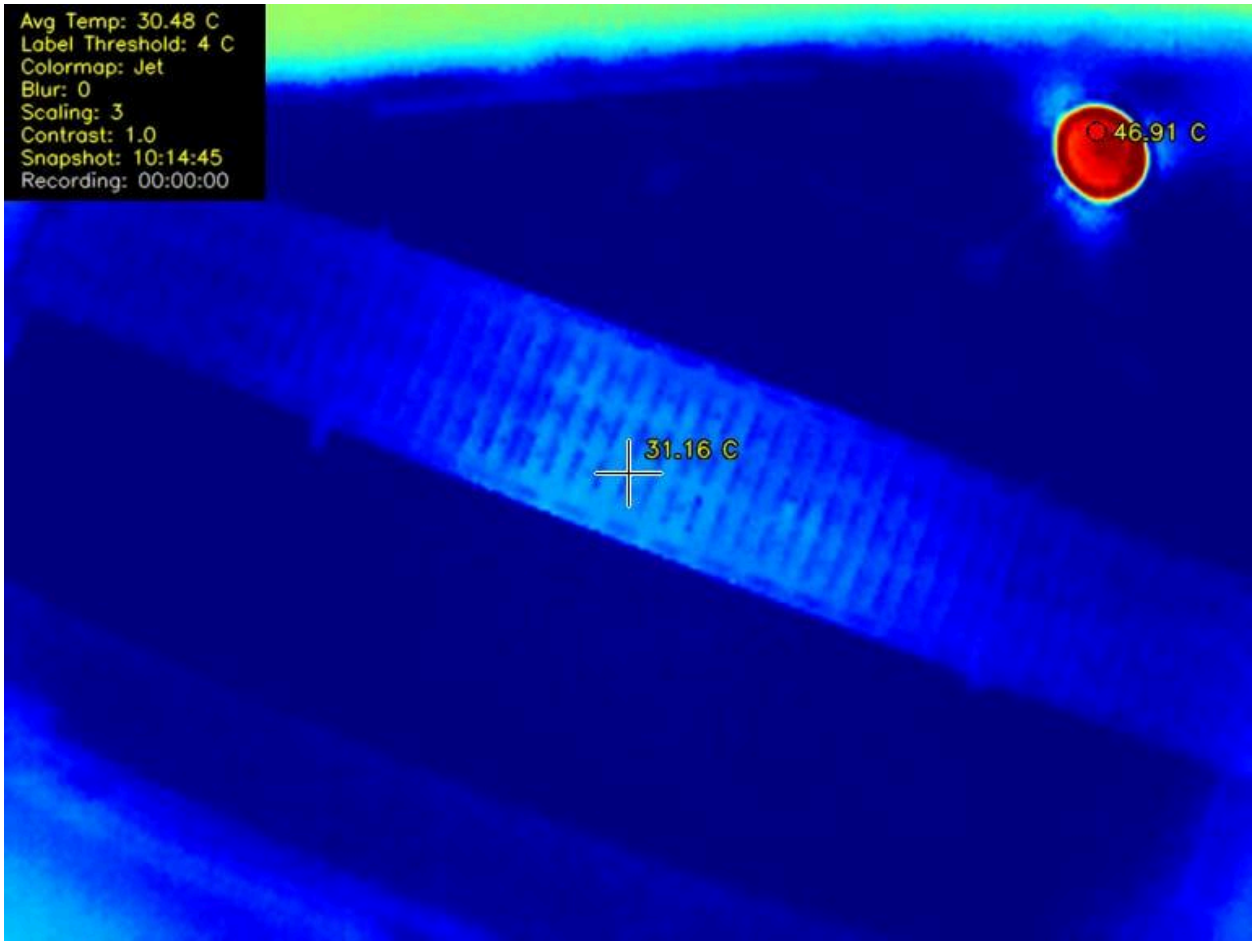
```
if keyPress == ord('v'): #contrast-
        alpha -= 0.1
        alpha = round(alpha,1)#fix round error
        if alpha<=0:
                alpha = 0.0


if keyPress == ord('h'):
        if hud==True:
                hud=False
        elif hud==False:
                hud=True

if keyPress == ord('m'): #m to cycle through color maps
        colormap += 1
        if colormap == 11:
                colormap = 0

if keyPress == ord('r') and recording == False: #r to start reording
        videoOut = rec()
        recording = True
        start = time.time()
if keyPress == ord('t'): #f to finish reording
        recording = False
        elapsed = "00:00:00"

if keyPress == ord('p'): #f to finish reording
        snaptime = snapshot(heatmap)

if keyPress == ord('q'):
        break
        capture.release()
        cv2.destroyAllWindows()
```

This code, after error correction enabled us to see the output from the Thermal camera.

Image Processing and Threshold Detection

After getting the live output from the thermal camera, we started to look for ways to capture the image and videos. The python script provided a manual way to take pictures

and videos, since it was our first time working with the camera as well as the code, we were only able to track the highest temperature spot on the live feed and could not finish automating the snapshot feature. An example of one of such images taken is shown below.



The crosshair shows where the camera is pointing to and the temperature, meanwhile on the top right corner a red dot shows the object with the highest temperature in frame.

Research on Gas Emission Detection

Parallel to camera testing, we also conducted some research on gases released before/during coal combustion. We were able to gather information from past incidents of coal combustion and could come to the conclusion that some gases are released before combustions like carbon monoxide. From what we found, we decided to add gas sensors to aid in detecting coal combustion. This added sensor would serve as a secondary safety measure in addition to the thermal camera. Coal fields are huge and we don't know if we'd be able to detect a noticeable change in the gas concentration, this needs to be further researched and tested.

Key Problems Identified

After familiarizing and testing with thermal camera, we've noticed some important problems and things that we have to consider.

1. **Ambient Temperature Dependence**
   While testing the thermal camera with the proprietary windows software, we've realized that for accurate measurement of the temperature, we need to give the ambient temperature. The camera software takes in the ambient temperature and modifies/corrects the temperature reading to match the actual temperature, without the ambient temperature value, the readings obtained will be incorrect.

2. **Distance Parameter**
   The camera actually requires a distance parameter to actually calculate the correct temperature. This distance is crucial for correctly calculating the spot size (the actual area represented by a pixel in the image). The distance to the object is also used for compensation for atmospheric dissipation of IR radiation over longer ranges.

3. **Reference for testing**
   For cross checking and verifying our temperature reading obtained from the thermal camera, we either need a device that can accurately measure the temperature of an object to get a reference or an object with known heat dissipation to act as a reference. Without any of the above two, there's no way to verify our temperature readings.

4. **Reflection of IR Radiation**
   During testing, we observed that IR can be reflected by certain surfaces, This was a crucial finding as this leads to the risk of the camera showing a temperature that represents not just the surface itself, but also the reflected IR radiation from the surrounding as well as the sun. Currently this needs to be further investigated and find a solution as soon as possible.

Our primary focus now is to solve the problem and tweak the software to get as accurate of a reading as possible.

Week 3 (12/09/25)

The objective for this day was to get the raw data of a thermal image and store the raw data as a separate file and then make a website to share and view the raw thermal data collected.

Raw Data Collection

One of the objectives for our project is to collect thermal data and send it to the base station for viewing and analysis. For that purpose just sending a thermal image alone won't be enough, a captured thermal image will only contain visualization of temperature corresponding to the colour scheme selected by us, we can't get temperature reading at points from the images. So in order for us to get a dynamic thermal image where we can point the mouse pointer and get the temperature reading of the corresponding point, we have to export the raw thermal data of each pixel. We have implemented this feature by making a tweak to the PyThermalCamera code so that it'll export the temperature reading represented by each pixel as a numpy array. The numpy array contains the temperature value of each pixel read by the thermal camera. The advantage of this approach is that we can import this numpy array into another program to recreate the thermal image but this time we can see the temperature reading of each and every pixel by just hovering the mouse pointer. This approach makes it easier for temperature interpretation.

Modified Code for Exporting Numpy file

```
#!/usr/bin/env python3
'''
Les Wright 21 June 2023
[https://youtube.com/leslaboratory](https://youtube.com/leslaboratory)
A Python program to read, parse and display thermal data from the Topdon TC001 Thermal camera!
'''
print('Les Wright 21 June 2023')
print('[https://youtube.com/leslaboratory](https://youtube.com/leslaboratory)')
print('A Python program to read, parse and display thermal data from the Topdon TC001 Thermal camera!')
print('')
print('Tested on Debian all features are working correctly')
print('This will work on the Pi However a number of workarounds are implemented!')
print('Seemingly there are bugs in the compiled version of cv2 that ships with the Pi!')
print('')
print('Key Bindings:')
print('')
print('a z: Increase/Decrease Blur')
```

```
print('s x: Floating High and Low Temp Label Threshold')
print('d c: Change Interpolated scale Note: This will not change the window size on the Pi')
print('f v: Contrast')
print('q w: Fullscreen Windowed (note going back to windowed does not seem to work on the Pi!)')
print('r t: Record and Stop')
print('p : Snapshot')
print('m : Cycle through ColorMaps')
print('h : Toggle HUD')

import cv2
import numpy as np
import argparse
import time
import io

#We need to know if we are running on the Pi, because openCV behaves a little oddly on all the builds!
#[https://raspberrypi.stackexchange.com/questions/5100/detect-that-a-python-program-is-running-on-the-pi](https://raspberrypi.stackexchange.com/questions/5100/detect-that-a-python-program-is-running-on-the-pi)
def is_raspberrypi():
    try:
        with io.open('/sys/firmware/devicetree/base/model', 'r') as m:
            if 'raspberry pi' in m.read().lower(): return True
    except Exception: pass
    return False

isPi = is_raspberrypi()

parser = argparse.ArgumentParser()
parser.add_argument("--device", type=int, default=0, help="Video Device number e.g. 0, use v4l2-ctl --list-devices")
args = parser.parse_args()

if args.device:
        dev = args.device
else:
        dev = 0

#init video
cap = cv2.VideoCapture('/dev/video'+str(dev), cv2.CAP_V4L)
#cap = cv2.VideoCapture(0)
#pull in the video but do NOT automatically convert to RGB, else it breaks the temperature data!
#[https://stackoverflow.com/questions/63108721/opencv-setting-videocap-property-to-cap-prop-convert-rgb-generates-weird-boolean](https://stackoverflow.com/questions/63108721/opencv-setting-videocap-property-to-cap-prop-convert-rgb-generates-weird-boolean)
```

```
if isPi == True:
	cap.set(cv2.CAP_PROP_CONVERT_RGB, 0.0)
else:
	cap.set(cv2.CAP_PROP_CONVERT_RGB, 0.0)

#256x192 General settings
# --- FIX: The original hardcoded values caused the error, so we'll remove them. ---
# width = 256 #Sensor width
# height = 192 #sensor height
scale = 3 #scale multiplier
# newWidth = width*scale
# newHeight = height*scale
alpha = 1.0 # Contrast control (1.0-3.0)
colormap = 0
font=cv2.FONT_HERSHEY_SIMPLEX
dispFullscreen = False
cv2.namedWindow('Thermal',cv2.WINDOW_GUI_NORMAL)
# cv2.resizeWindow('Thermal', newWidth,newHeight) # This will be set after the first frame
rad = 0 #blur radius
threshold = 2
hud = True
recording = False
elapsed = "00:00:00"
snaptime = "None"
first_frame = True # A new flag to handle window resize on the first frame

def rec(newWidth, newHeight):
	now = time.strftime("%Y%m%d--%H%M%S")
	#do NOT use mp4 here, it is flakey!
	videoOut = cv2.VideoWriter(now+'output.avi', cv2.VideoWriter_fourcc(*'XVID'),25,
(newWidth,newHeight))
	return(videoOut)

def snapshot(heatmap):
	#I would put colons in here, but it Win throws a fit if you try and open them!
	now = time.strftime("%Y%m%d-%H%M%S")
	snaptime = time.strftime("%H:%M:%S")
	cv2.imwrite("TC001"+now+".png", heatmap)
	return snaptime


while(cap.isOpened()):
	# Capture frame-by-frame
	ret, frame = cap.read()
	if ret == True:
		imdata,thdata = np.array_split(frame, 2)
```

```
    # --- FIX: Dynamically get the dimensions from the captured data. ---
    # This prevents the IndexError.
            height, width, channels = thdata.shape
            if first_frame:
                newWidth = width*scale
                newHeight = height*scale
                cv2.resizeWindow('Thermal', newWidth, newHeight)
                first_frame = False

            #now parse the data from the bottom frame and convert to temp!

#[https://www.eevblog.com/forum/thermal-imaging/infiray-and-their-p2-pro-discussion/200/](https://www.eevblog.com/forum/thermal-imaging/infiray-and-their-p2-pro-discussion/200/)
            #Huge props to LeoDJ for figuring out how the data is stored and how to compute temp from it.

            # --- FIX: Loop over the entire array to create the rawtemps matrix ---
            rawtemps = np.zeros((height, width), dtype=np.float32)
            for r in range(height):
                for c in range(width):
                    hi = thdata[r][c][0]
                    lo = thdata[r][c][1]
                    val = hi+lo*256
                    rawtemps[r,c] = (val/64)-273.15

            # --- NEW: Retrieve and print the temperature at a specific pixel ---
            # Let's check the temperature at a specific pixel, e.g., (100, 100).
            # We must make sure the indices are within the valid range.
            pixel_row = 100
            pixel_col = 100
            if 0 <= pixel_row < height and 0 <= pixel_col < width:
                pixel_temp = rawtemps[pixel_row, pixel_col]
                print(f"Temperature at pixel ({pixel_row}, {pixel_col}): {pixel_temp:.2f}°C")

            #grab data from the center pixel...
    # --- FIX: Use the dynamic height and width ---
            hi = thdata[int(height/2)][int(width/2)][0]
            lo = thdata[int(height/2)][int(width/2)][1]
            #print(hi,lo)
            lo = lo*256
            rawtemp = hi+lo
            #print(rawtemp)
            temp = (rawtemp/64)-273.15
            temp = round(temp,2)
            #print(temp)
```

```
                #break

                #find the max temperature in the frame
                lomax = thdata[...,1].max()
                posmax = thdata[...,1].argmax()
                #since argmax returns a linear index, convert back to row and col
        # --- FIX: Use the dynamic 'width' for divmod calculation ---
                mcol,mrow = divmod(posmax,width)
                himax = thdata[mcol][mrow][0]
                lomax=lomax*256
                maxtemp = himax+lomax
                maxtemp = (maxtemp/64)-273.15
                maxtemp = round(maxtemp,2)


                #find the lowest temperature in the frame
                lomin = thdata[...,1].min()
                posmin = thdata[...,1].argmin()
                #since argmax returns a linear index, convert back to row and col
        # --- FIX: Use the dynamic 'width' for divmod calculation ---
                lcol,lrow = divmod(posmin,width)
                himin = thdata[lcol][lrow][0]
                lomin=lomin*256
                mintemp = himin+lomin
                mintemp = (mintemp/64)-273.15
                mintemp = round(mintemp,2)

                #find the average temperature in the frame
                loavg = thdata[...,1].mean()
                hiavg = thdata[...,0].mean()
                loavg=loavg*256
                avgtemp = loavg+hiavg
                avgtemp = (avgtemp/64)-273.15
                avgtemp = round(avgtemp,2)



                # Convert the real image to RGB
                bgr = cv2.cvtColor(imdata,  cv2.COLOR_YUV2BGR_YUYV)
                #Contrast
                bgr = cv2.convertScaleAbs(bgr, alpha=alpha)#Contrast
                #bicubic interpolate, upscale and blur
                bgr = cv2.resize(bgr,(newWidth,newHeight),interpolation=cv2.INTER_CUBIC)#Scale
up!
                if rad>0:
                        bgr = cv2.blur(bgr,(rad,rad))
```

```
#apply colormap
if colormap == 0:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_JET)
        cmapText = 'Jet'
if colormap == 1:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_HOT)
        cmapText = 'Hot'
if colormap == 2:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_MAGMA)
        cmapText = 'Magma'
if colormap == 3:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_INFERNO)
        cmapText = 'Inferno'
if colormap == 4:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_PLASMA)
        cmapText = 'Plasma'
if colormap == 5:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_BONE)
        cmapText = 'Bone'
if colormap == 6:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_SPRING)
        cmapText = 'Spring'
if colormap == 7:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_AUTUMN)
        cmapText = 'Autumn'
if colormap == 8:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_VIRIDIS)
        cmapText = 'Viridis'
if colormap == 9:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_PARULA)
        cmapText = 'Parula'
if colormap == 10:
        heatmap = cv2.applyColorMap(bgr, cv2.COLORMAP_RAINBOW)
        heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)
        cmapText = 'Inv Rainbow'

#print(heatmap.shape)

# draw crosshairs
cv2.line(heatmap,(int(newWidth/2),int(newHeight/2)+20),\
(int(newWidth/2),int(newHeight/2)-20),(255,255,255),2) #vline
cv2.line(heatmap,(int(newWidth/2)+20,int(newHeight/2)),\
(int(newWidth/2)-20,int(newHeight/2)),(255,255,255),2) #hline

cv2.line(heatmap,(int(newWidth/2),int(newHeight/2)+20),\
```

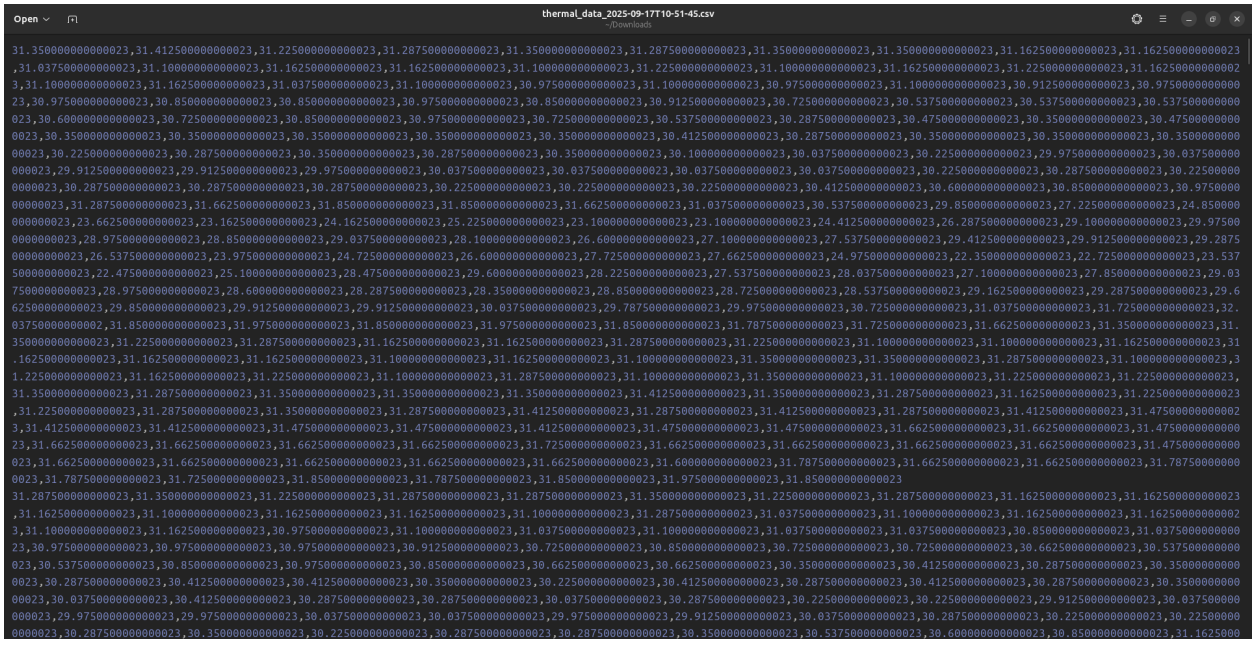
```
(int(newWidth/2),int(newHeight/2)-20),(0,0,0),1) #vline
cv2.line(heatmap,(int(newWidth/2)+20,int(newHeight/2)),\
(int(newWidth/2)-20,int(newHeight/2)),(0,0,0),1) #hline
#show temp
cv2.putText(heatmap,str(temp)+' C', (int(newWidth/2)+10, int(newHeight/2)-10),\
cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(heatmap,str(temp)+' C', (int(newWidth/2)+10, int(newHeight/2)-10),\
cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 255, 255), 1, cv2.LINE_AA)

if hud==True:
        # display black box for our data
        cv2.rectangle(heatmap, (0, 0),(160, 120), (0,0,0), -1)
        # put text in the box
        cv2.putText(heatmap,'Avg Temp: '+str(avgtemp)+' C', (10, 14),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1, cv2.LINE_AA)

        cv2.putText(heatmap,'Label Threshold: '+str(threshold)+' C', (10, 28),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1, cv2.LINE_AA)

        cv2.putText(heatmap,'Colormap: '+cmapText, (10, 42),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1, cv2.LINE_AA)

        cv2.putText(heatmap,'Blur: '+str(rad)+' ', (10, 56),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1, cv2.LINE_AA)

        cv2.putText(heatmap,'Scaling: '+str(scale)+' ', (10, 70),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1, cv2.LINE_AA)

        cv2.putText(heatmap,'Contrast: '+str(alpha)+' ', (10, 84),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1, cv2.LINE_AA)


        cv2.putText(heatmap,'Snapshot: '+snaptime+' ', (10, 98),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 255, 255), 1, cv2.LINE_AA)

        if recording == False:
                cv2.putText(heatmap,'Recording: '+elapsed, (10, 112),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(200, 200, 200), 1,
cv2.LINE_AA)
        if recording == True:
                cv2.putText(heatmap,'Recording: '+elapsed, (10, 112),\
                cv2.FONT_HERSHEY_SIMPLEX, 0.4,(40, 40, 255), 1, cv2.LINE_AA)

#Yeah, this looks like we can probably do this next bit more efficiently!
#display floating max temp
if maxtemp > avgtemp+threshold:
```

```
        cv2.circle(heatmap, (mrow*scale, mcol*scale), 5, (0,0,0), 2)
        cv2.circle(heatmap, (mrow*scale, mcol*scale), 5, (0,0,255), -1)
        cv2.putText(heatmap,str(maxtemp)+' C', ((mrow*scale)+10, (mcol*scale)+5),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0,0,0), 2, cv2.LINE_AA)
        cv2.putText(heatmap,str(maxtemp)+' C', ((mrow*scale)+10, (mcol*scale)+5),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 255, 255), 1, cv2.LINE_AA)

#display floating min temp
if mintemp < avgtemp-threshold:
        cv2.circle(heatmap, (lrow*scale, lcol*scale), 5, (0,0,0), 2)
        cv2.circle(heatmap, (lrow*scale, lcol*scale), 5, (255,0,0), -1)
        cv2.putText(heatmap,str(mintemp)+' C', ((lrow*scale)+10, (lcol*scale)+5),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0,0,0), 2, cv2.LINE_AA)
        cv2.putText(heatmap,str(mintemp)+' C', ((lrow*scale)+10, (lcol*scale)+5),\
        cv2.FONT_HERSHEY_SIMPLEX, 0.45,(0, 255, 255), 1, cv2.LINE_AA)

#display image
cv2.imshow('Thermal',heatmap)

if recording == True:
        elapsed = (time.time() - start)
        elapsed = time.strftime("%H:%M:%S", time.gmtime(elapsed))
        #print(elapsed)
        videoOut.write(heatmap)

keyPress = cv2.waitKey(1)
if keyPress == ord('a'): #Increase blur radius
        rad += 1
if keyPress == ord('z'): #Decrease blur radius
        rad -= 1
        if rad <= 0:
                rad = 0

if keyPress == ord('s'): #Increase threshold
        threshold += 1
if keyPress == ord('x'): #Decrease threashold
        threshold -= 1
        if threshold <= 0:
                threshold = 0

if keyPress == ord('d'): #Increase scale
        scale += 1
        if scale >=5:
                scale = 5
        newWidth = width*scale
        newHeight = height*scale
```

```
                if dispFullscreen == False and isPi == False:
                        cv2.resizeWindow('Thermal', newWidth,newHeight)
        if keyPress == ord('c'): #Decrease scale
                scale -= 1
                if scale <= 1:
                        scale = 1
                newWidth = width*scale
                newHeight = height*scale
                if dispFullscreen == False and isPi == False:
                        cv2.resizeWindow('Thermal', newWidth,newHeight)

        if keyPress == ord('q'): #enable fullscreen
                dispFullscreen = True
                cv2.namedWindow('Thermal',cv2.WND_PROP_FULLSCREEN)

cv2.setWindowProperty('Thermal',cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)
        if keyPress == ord('w'): #disable fullscreen
                dispFullscreen = False
                cv2.namedWindow('Thermal',cv2.WINDOW_GUI_NORMAL)

cv2.setWindowProperty('Thermal',cv2.WND_PROP_AUTOSIZE,cv2.WINDOW_GUI_NORMAL)
                cv2.resizeWindow('Thermal', newWidth,newHeight)

        if keyPress == ord('f'): #contrast+
                alpha += 0.1
                alpha = round(alpha,1)#fix round error
                if alpha >= 3.0:
                        alpha=3.0
        if keyPress == ord('v'): #contrast-
                alpha -= 0.1
                alpha = round(alpha,1)#fix round error
                if alpha<=0:
                        alpha = 0.0


        if keyPress == ord('h'):
                if hud==True:
                        hud=False
                elif hud==False:
                        hud=True

        if keyPress == ord('m'): #m to cycle through color maps
                colormap += 1
                if colormap == 11:
                        colormap = 0
```

```
if keyPress == ord('r') and recording == False: #r to start reording
        videoOut = rec()
        recording = True
        start = time.time()
if keyPress == ord('t'): #f to finish reording
        recording = False
        elapsed = "00:00:00"

if keyPress == ord('p'): #f to finish recording
        snaptime = snapshot(heatmap)

if keyPress == ord('q'):
        break
        capture.release()
        cv2.destroyAllWindows()
```

Numpy array example



Raw Thermal Data Viewer

Just getting the numpy file containing the thermal data alone is not enough. We need a way to display the raw data in a meaningful manner, so we have created a simple website page that can take the numpy file and recreate it as a thermal image with all the necessary data. Currently we have the options to view the temperature corresponding to each pixel along with some basic functionalities like exporting the data in various

formats. The main benefit of making a website is that we can host website then upload the captured data directly to the website and we can view it from anywhere.

Code for Thermal data Viewer

```
import numpy as np
import json

# Load the temperature data
data = np.load("temperature_array_20250912-145735.npy")
data = np.flipud(data)  # Flip vertically

print(f"Creating enhanced thermal viewer...")
print(f"Data shape: {data.shape}")
print(f"Temperature range: {np.min(data):.2f}°C to {np.max(data):.2f}°C")

# Convert data to JSON for JavaScript
data_json = json.dumps(data.tolist())
min_temp = float(np.min(data))
max_temp = float(np.max(data))
mean_temp = float(np.mean(data))

# Create enhanced HTML with all features
enhanced_html = f'''<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Enhanced Thermal Camera Viewer</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Inter:wght@400;600;700&display=swap');

        * {{
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }}

        body {{
            font-family: 'Inter', 'Roboto', Arial, sans-serif;
            background: linear-gradient(120deg, #232526 0%, #414345 100%);
            color: #e0e0e0;
```

```
    min-height: 100vh;
}}

.header {{
    background: rgba(40, 48, 72, 0.85);
    color: #fff;
    padding: 32px 0 18px 0;
    text-align: center;
    border-radius: 0 0 24px 24px;
    box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
    backdrop-filter: blur(8px);
    font-weight: 700;
    letter-spacing: 1px;
}}

.container {{
    max-width: 1100px;
    margin: 0 auto;
    padding: 32px 16px;
}}

.controls, .stats, .plot-container {{
    background: rgba(44, 54, 80, 0.85);
    border-radius: 18px;
    box-shadow: 0 4px 24px rgba(31, 38, 135, 0.17);
    backdrop-filter: blur(6px);
    margin-bottom: 24px;
    padding: 24px;
}}

.controls {{
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 24px;
}}

.control-group {{
    display: flex;
    flex-direction: column;
}}

.control-group label {{
    font-weight: 600;
    margin-bottom: 7px;
    color: #b0b8d1;
    letter-spacing: 0.5px;
```

```
}}

.control-group select, .control-group input {{
    padding: 10px;
    border: 2px solid #353b48;
    border-radius: 8px;
    font-size: 15px;
    background: rgba(60, 70, 100, 0.7);
    color: #e0e0e0;
    transition: border-color 0.2s;
}}

.control-group select:focus, .control-group input:focus {{
    border-color: #6a82fb;
    outline: none;
}}

.stats {{
    background: white;
    padding: 15px;
    border-radius: 10px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
    margin-bottom: 20px;
}}

.stats-grid {{
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(130px, 1fr));
    gap: 18px;
    text-align: center;
}}

.stat-item {{
    padding: 14px 0;
    background: rgba(60, 70, 100, 0.7);
    border-radius: 8px;
    border-left: 4px solid #6a82fb;
    box-shadow: 0 2px 8px rgba(31, 38, 135, 0.07);
}}

.stat-value {{
    font-size: 20px;
    font-weight: 700;
    color: #6a82fb;
    letter-spacing: 0.5px;
}}
```

```
.stat-label {{
    font-size: 13px;
    color: #b0b8d1;
    margin-top: 4px;
    font-weight: 500;
}}

.plot-container {{
    padding: 28px 18px;
}}

#thermal-plot {{
    width: 100%;
    height: 600px;
    border-radius: 12px;
    box-shadow: 0 2px 16px rgba(31, 38, 135, 0.13);
    background: #232526;
}}

.export-buttons {{
    margin-top: 28px;
    text-align: center;
}}

.btn {{
    background: linear-gradient(135deg, #6a82fb 0%, #36454F 100%);
    color: white;
    padding: 12px 26px;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    margin: 0 12px 12px 0;
    font-size: 15px;
    font-weight: 600;
    box-shadow: 0 2px 8px rgba(31, 38, 135, 0.13);
    transition: transform 0.2s, box-shadow 0.2s;
}}

.btn:hover {{
    transform: translateY(-2px) scale(1.04);
    box-shadow: 0 4px 16px rgba(31, 38, 135, 0.23);
}}

.temperature-info {{
    position: fixed;
```

```
            top: 24px;
            right: 24px;
            background: rgba(44, 54, 80, 0.95);
            color: #fff;
            padding: 14px 18px;
            border-radius: 10px;
            font-family: 'Inter', monospace;
            font-size: 15px;
            display: none;
            z-index: 1000;
            box-shadow: 0 2px 12px rgba(31, 38, 135, 0.17);
            backdrop-filter: blur(4px);
        }}

        @media (max-width: 900px) {{
            .container {{
                padding: 12px;
            }}
            .controls {{
                grid-template-columns: 1fr;
            }}
            #thermal-plot {{
                height: 400px;
            }}
        }}
    </style>
</head>
<body>
    <div class="header">
        <h1>🌡️ Enhanced Thermal Camera Viewer</h1>
        <p>Interactive visualization with advanced controls</p>
    </div>

    <div class="container">
        <div class="stats">
            <div class="stats-grid">
                <div class="stat-item">
                    <div class="stat-value">{data.shape[1]} × {data.shape[0]}</div>
                    <div class="stat-label">Resolution</div>
                </div>
                <div class="stat-item">
                    <div class="stat-value">{min_temp:.1f}°C</div>
                    <div class="stat-label">Min Temperature</div>
                </div>
                <div class="stat-item">
                    <div class="stat-value">{max_temp:.1f}°C</div>
```

```
                <div class="stat-label">Max Temperature</div>
            </div>
            <div class="stat-item">
                <div class="stat-value">{mean_temp:.1f}°C</div>
                <div class="stat-label">Average</div>
            </div>
            <div class="stat-item">
                <div class="stat-value" id="current-temp">--</div>
                <div class="stat-label">Cursor Position</div>
            </div>
        </div>
    </div>

    <div class="controls">
        <div class="control-group">
            <label for="colorscale">Color Scale:</label>
            <select id="colorscale" onchange="updatePlot()">
                <option value="Jet">Jet (Classic)</option>
                <option value="Viridis">Viridis (Perceptual)</option>
                <option value="Hot">Hot (Red-Yellow)</option>
                <option value="Cool">Cool (Blue-Cyan)</option>
                <option value="Rainbow">Rainbow</option>
                <option value="Turbo">Turbo</option>
                <option value="Plasma">Plasma</option>
                <option value="Inferno">Inferno</option>
                <option value="RdYlBu">Red-Yellow-Blue</option>
                <option value="Spectral">Spectral</option>
            </select>
        </div>

        <div class="control-group">
            <label for="min-temp">Min Temperature (°C):</label>
            <input type="number" id="min-temp" value="{min_temp:.1f}" step="0.1"
onchange="updatePlot()">
        </div>

        <div class="control-group">
            <label for="max-temp">Max Temperature (°C):</label>
            <input type="number" id="max-temp" value="{max_temp:.1f}" step="0.1"
onchange="updatePlot()">
        </div>

        <div class="control-group">
            <label for="smoothing">Smoothing:</label>
            <select id="smoothing" onchange="updatePlot()">
                <option value="false">None</option>
```

```
                <option value="best">Best</option>
                <option value="fast">Fast</option>
            </select>
        </div>
    </div>

    <div class="plot-container">
        <div id="thermal-plot"></div>
        <div class="export-buttons">
            <button class="btn" onclick="exportImage('png')">📸 Export PNG</button>
            <button class="btn" onclick="exportImage('svg')">🎨 Export SVG</button>
            <button class="btn" onclick="exportImage('pdf')">📄 Export PDF</button>
            <button class="btn" onclick="exportData()">📊 Export Data</button>
            <button class="btn" onclick="resetView()">🔄 Reset View</button>
        </div>
    </div>
</div>

<div class="temperature-info" id="temp-info">
    Temperature: --°C<br>
    Position: (--,--)
</div>

<script>
    // Thermal data
    const thermalData = {data_json};
    const originalMinTemp = {min_temp};
    const originalMaxTemp = {max_temp};

    let currentPlot = null;

    // Initialize the plot
    function initPlot() {{
        updatePlot();
    }}

    // Update plot with current settings
    function updatePlot() {{
        const colorscale = document.getElementById('colorscale').value;
        const minTemp = parseFloat(document.getElementById('min-temp').value);
        const maxTemp = parseFloat(document.getElementById('max-temp').value);
        const smoothing = document.getElementById('smoothing').value === 'true';

        const trace = {{
            z: thermalData,
            type: 'heatmap',
```

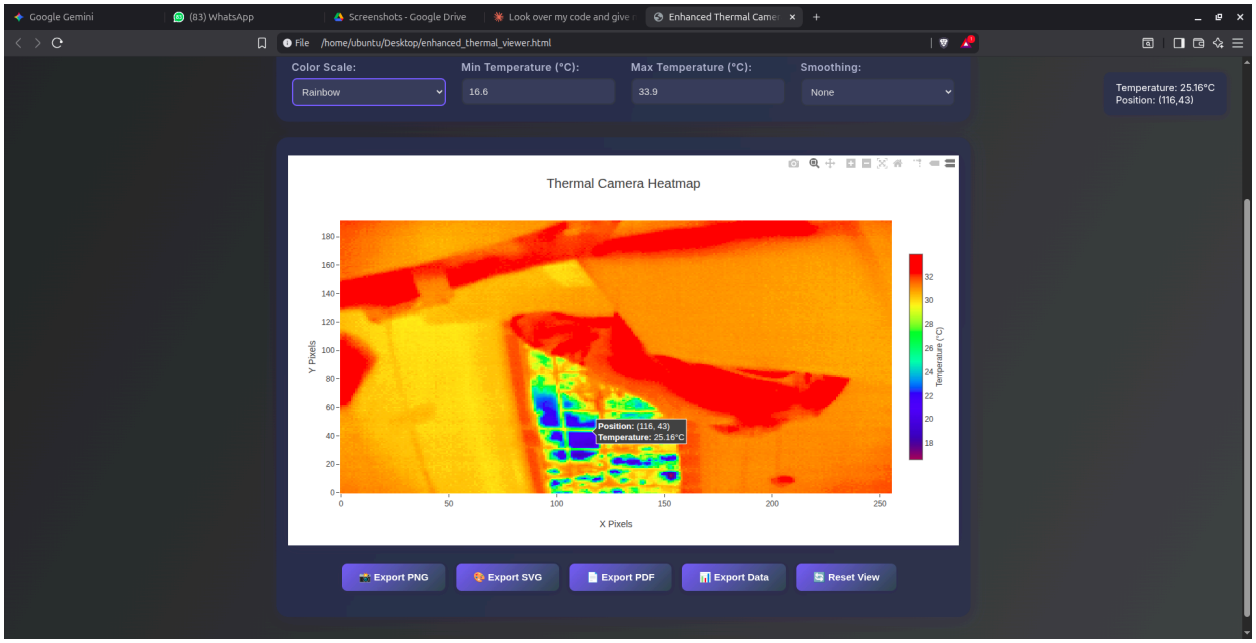
```
    colorscale: colorscale,
    showscale: true,
    zmin: minTemp,
    zmax: maxTemp,
    colorbar: {{
        title: {{
            text: 'Temperature (°C)',
            side: 'right'
        }},
        thickness: 20,
        len: 0.8
    }},
    hovertemplate: '<b>Position:</b> (%{{x}}, %{{y}})<br>' +
                   '<b>Temperature:</b> %{{z:.2f}}°C<br>' +
                   '<extra></extra>',
    connectgaps: smoothing
}};

const layout = {{
    title: {{
        text: 'Thermal Camera Heatmap',
        font: {{ size: 20 }}
    }},
    xaxis: {{
        title: 'X Pixels',
        showgrid: true,
        zeroline: false
    }},
    yaxis: {{
        title: 'Y Pixels',
        showgrid: true,
        zeroline: false
    }},
    plot_bgcolor: 'white',
    paper_bgcolor: 'white',
    font: {{ family: 'Segoe UI, Arial, sans-serif' }}
}};

const config = {{
    displayModeBar: true,
    displaylogo: false,
    modeBarButtonsToAdd: ['pan2d', 'zoom2d', 'autoScale2d'],
    modeBarButtonsToRemove: ['lasso2d', 'select2d'],
    responsive: true
}};
```

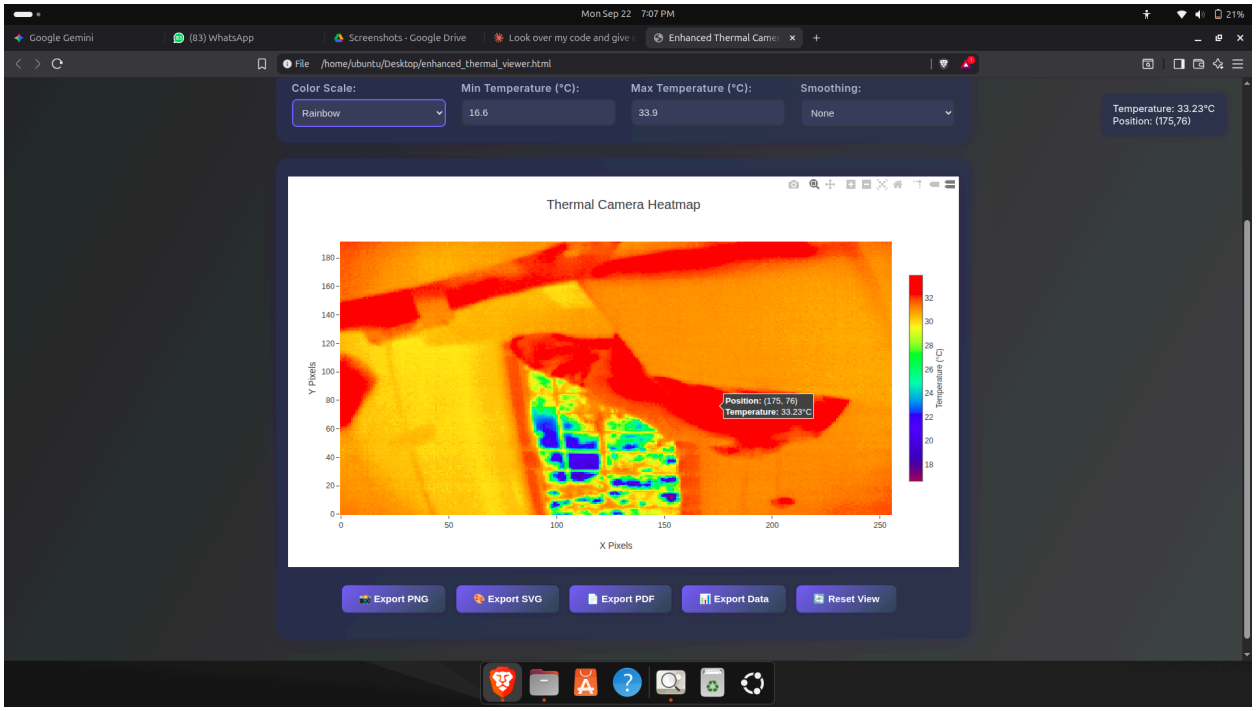
```
    Plotly.newPlot('thermal-plot', [trace], layout, config);

    // Add hover event to show temperature info
    document.getElementById('thermal-plot').on('plotly_hover', function(data) {{
        const point = data.points[0];
        const temp = point.z.toFixed(2);
        const x = point.x;
        const y = point.y;

        document.getElementById('current-temp').textContent = temp + '°C';

        const tempInfo = document.getElementById('temp-info');
        tempInfo.innerHTML = `Temperature: ${{temp}}°C<br>Position: (${{x}},${{y}})`;
        tempInfo.style.display = 'block';
    }});

    document.getElementById('thermal-plot').on('plotly_unhover', function() {{
        document.getElementById('temp-info').style.display = 'none';
        document.getElementById('current-temp').textContent = '--';
    }});
}}

// Export functions
function exportImage(format) {{
    const filename = `thermal_image_${{new Date().toISOString().slice(0,19).replace(/:/g, '-')}}`;

    Plotly.downloadImage('thermal-plot', {{
        format: format,
        width: 1200,
        height: 800,
        filename: filename
    }});
}}

function exportData() {{
    const csvContent = thermalData.map(row => row.join(',')).join('\\n');
    const blob = new Blob([csvContent], {{ type: 'text/csv' }});
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `thermal_data_${{new Date().toISOString().slice(0,19).replace(/:/g, '-')}}.csv`;
    a.click();
}}

function resetView() {{
    document.getElementById('colorscale').value = 'Jet';
```

```
            document.getElementById('min-temp').value = originalMinTemp.toFixed(1);
            document.getElementById('max-temp').value = originalMaxTemp.toFixed(1);
            document.getElementById('smoothing').value = 'false';
            updatePlot();
        }}

        // Keyboard shortcuts
        document.addEventListener('keydown', function(e) {{
            if (e.ctrlKey || e.metaKey) {{
                switch(e.key) {{
                    case 's':
                        e.preventDefault();
                        exportImage('png');
                        break;
                    case 'r':
                        e.preventDefault();
                        resetView();
                        break;
                }}
            }}
        }});

        // Initialize on page load
        window.addEventListener('load', function() {{
            initPlot();
            console.log('Enhanced thermal viewer loaded successfully!');
            console.log('Keyboard shortcuts: Ctrl+S (save), Ctrl+R (reset)');
        }});
    </script>
</body>
</html>'''

# Save the enhanced HTML file
with open("enhanced_thermal_viewer.html", "w", encoding='utf-8') as f:
    f.write(enhanced_html)

print("Enhanced thermal viewer created: enhanced_thermal_viewer.html")
print("\n Features included:")
print("  • Multiple colorscale options (Jet, Viridis, Hot, etc.)")
print("  • Temperature range controls (min/max sliders)")
print("  • Real-time temperature display on hover")
print("  • Export capabilities (PNG, SVG, PDF)")
print("  • Data export to CSV")
print("  • Responsive design for mobile/desktop")
print("  • Keyboard shortcuts (Ctrl+S to save, Ctrl+R to reset)")
print("  • Statistical information display")
```

```
print("  • Zoom and pan controls")
print("  • Modern, professional UI")
print("\n Open 'enhanced_thermal_viewer.html' in your browser to use!")
```

Thermal Data Viewer Images





Key Problems Identified

Stitching and Mapping: Right now the numpy file only contains the data of a single shot. But we can't get the temperature reading of the entire field in a single frame so we would have to use other mapping and stitching techniques to merge together multiple images to get a coherent single output. But the problem with that is that thermal images tend to have relatively low details and a lot of mapping techniques that we have researched works on the basis of finding common points and textures from overlapping images to stitch together, which thermal images lack. The work around that we've thought deals with the aforementioned numpy array. Rather than trying to stitch together thermal images in png format, try to stitch together the numpy arrays containing the thermal data, but we have not concluded the feasibility of this approach and also each time we take a picture and save the numpy array, there are slight variations in the values of each pixels so finding commons values and merging the array would be a difficult job.