Thermal Mapping Module for Supporting Wildfire Suppression

Miguel Alexandre Gonçalves Lourenço – Nº 61251

Free Option - Advanced Topics in Electronics

2024, March

Wildfires are among the natural disasters with the greatest impact, causing enormous devastation to humans and the environment. The incomplete extinguishment of wildfires represents a challenge, as they have the potential to reignite, delaying their total resolution or even leading to new outbreaks. As such, early identification of reignition hotspots can prevent wildfires from perpetuating their destruction, hindering effective wildfire suppression. This essay proposes a Thermal Mapping Module to detect hotspots that could potentially lead to new wildfire outbreaks.

Background and Motivation

Wildfires threaten ecosystems and communities, and the reignition of these fires from unburned fuel can hinder efforts for complete extinguishment and even result in new outbreaks. Incomplete combustion in the initial wildfire may leave behind dry and flammable vegetation, serving as potential ignition sources for subsequent fires (Sullivan, 2017). In addition, smouldering fires, often found in peaty or organic soils, may remain undetected for long periods. When conditions become favourable again, these smouldering wildfires can grow, increasing the complexity of wildfire management (Zhang et al., 2024). As such, their rapid detection can speed up their extinguishment, thus minimising the impact of these phenomena in various domains (social, economic, environmental, etc.).

This essay proposes a Thermal Mapping Module (TMM) based on the ESP32 development board and the MLX90640 thermal infrared (IR) array sensor to detect hotspots or smouldering fires. This device is designed to enhance the efficiency of terrain scouting and swiftly identify potential danger zones using IR scanning. Simplifying this process means less attention needs to be given to each specific area or measurement, thus expediting the aftermath patrols, and ensuring no potential reignition sources go undetected and are fully extinguished.

Wildfire management and suppression techniques have benefited significantly from technological advances that have improved the initial response to wildfires. These advances have resulted in faster suppression of wildfires in their early stages. However, the remaining wildfires, although less frequent, are characterised by their larger scale and greater complexity, requiring a more comprehensive and interdisciplinary approach to their suppression (Beighley & Hyde, 2018). Given these challenges, there is growing recognition of the need to consider the concept of 'smart firefighting', which refers to integrating advanced technologies, data analytics, and innovative strategies to increase the effectiveness and efficiency of firefighting operations (Wu et al., 2017). By combining traditional firefighting expertise with modern technologies, 'smart firefighting' seeks to improve situational awareness and the overall decision-making process during these operations.

The rapid development of technology has led to the widespread integration of thermal imaging, particularly in Unmanned Aerial Vehicles (UAVs) and satellites (Plucinski, 2019). Despite significant advancements in thermal imaging technology in UAVs and satellites that can carry more capable sensors, operate in complex terrain or with dense vegetation, and have extended coverage over long distances (especially in the case of satellites), there can still be limitations when it comes to resolution in some applications. Particularly during post-wildfire terrain scouting, their thermal imaging limitations become apparent as subtle signs of smouldering or potential hotspots require a level of detail that can exceed the current capabilities of these systems. By equipping firefighters with higher-resolution technologies capable of capturing the thermal signature in greater detail, we can extend their knowledge and experience with technological tools that empower and increase their efficiency in identifying potential danger zones using IR scanning (Barrado et al., 2010; VanderMeer, Ault, & Hvenegaard, 2011)

Design of the Thermal Mapping Module

The TMM proposed in this essay is used to detect hotspots or smouldering fires. This device leverages an MLX90640 thermal IR array sensor whose data is processed by an ESP32 development board (which shows it on an ILI9341 2.8″ TFT display to facilitate interpretation by the device operator). Its data is transmitted via Bluetooth Low Energy (BLE) to a client application running on a smartphone. The use of this communication protocol allows the smartphone to be connected simultaneously to the TMM and the internet, thus facilitating the transmission of its data to a server where it can be accessed by other peers. The TMM diagram can be found in Figure 1 and its firmware is in Appendix A.

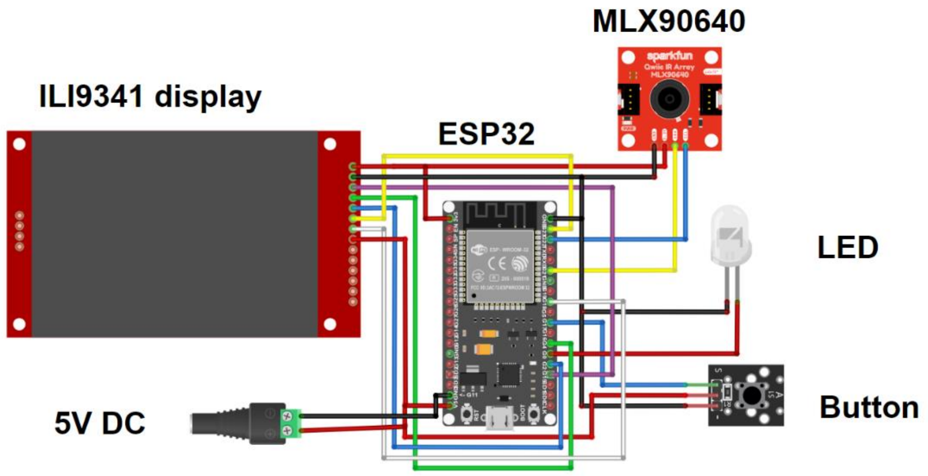


Figure 1 - The TMM diagram.

Thermal sensor

The MLX90640 IR sensor (Melexis, 2018) used is available in a breakout board and is equipped with a 31x24 array of thermopile sensors, in practice creating a low-resolution thermal imaging camera. This specific sensor features a 55ºx35º field of view (FOV) sensor with a temperature measurement range of -40ºC-300ºC. It provides its data via the I2C interface, thus allowing it to be connected to a microcontroller. In this case, the ESP32 development board was chosen because it has the processing capacity and RAM required to handle the MLX90640 sensor's data, as well as having a built-in BLE interface that enables its data to be transmitted to a smartphone (Espressif Systems, 2024). The firmware was programmed through the Arduino IDE software that contains a "setup" function for variable and settings initialization and a "loop" function where the MLX90640 sensor is continuously read.

The added ILI9341 2.8" TFT display is essential for the operator to visually interpret the thermal data. It connects to the ESP32 development board through the SPI interface and draws a new thermal image at a frequency of 16Hz (which is the maximum refresh rate that the MLX90640 sensor could operate). The display's layout can be seen in Figure 2. The colour scale at the right shows a colour gradient from the minimum temperature measurement (stored in variable T_min) to the maximum temperature measurement (stored in variable T_max) and allows for a quick analysis of the value measurements. At the bottom centre, the temperature value of the middle pixel is displayed (represented by T+).

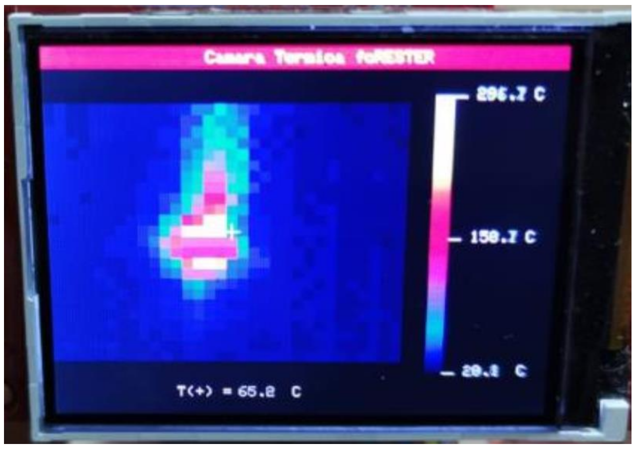


Figure 2 - The TMM display.

The "loop" function runs through every single value stored in the sensor's array (in the 'mlx90640To' variable) to determine the highest and lowest temperature value read. These values are essential to the TMM as they set the scale for the colour grading displayed on the screen. Then, the program assigns a colour to a group of pixels from the "getColour" function, which classifies the colour as one of six tiers using a value calculated at the beginning of the display's drawing function. This value is a measurement of where the real temperature value fits within the boundaries set in the previous function. The "drawline" function serves to set the pixels between two coordinates to a specific colour. The "fillRect" function receives the coordinates of a pair of pixels and creates a rectangle of a set colour with those two points as opposite corners. Lastly, the "setCursor" and "print" functions allow for text to be displayed without needing to draw out each letter.

The detection of hotspots makes use of the Normal distribution (see Figure 3), more specifically of its empirical rule that says that approximately 99.7% of the values in a normally distributed dataset lie within 3 standard deviations of the mean - meaning within the range of -3 times the standard deviation to +3 times the standard deviation. This implies that most observations in a Normal distribution are clustered around the mean, with fewer occurring as we move away from it. Temperature values outside this range are considered outliers. To implement it, the average temperature is first calculated and then the standard deviation. When there is no hotspot, the values obtained do not vary much from the average, so the standard deviation is low. If the standard deviation is very high, this means that the values are quite dispersed and far from the average, which allows us to conclude that there is a hotspot. The result is a value of '1' if a hotspot is detected, or '0' if it is not detected.

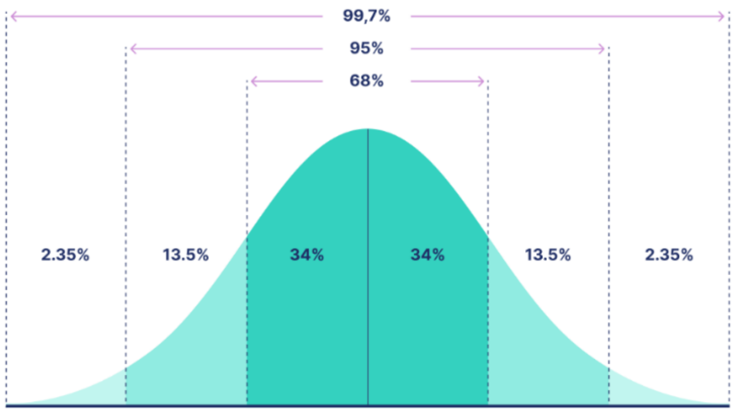


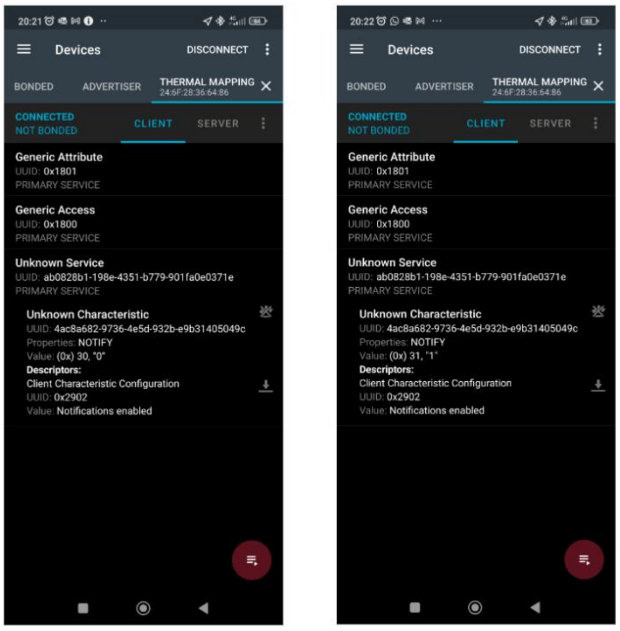
Figure 3 – The Normal distribution graph (Bhandari, 2023).

Communication

The TMM communicates with external equipment (such as a smartphone) via BLE. The only data transmitted is whether a hotspot has been detected (represented by the value '1') or not detected (represented by the value '0'). Only this value will be transmitted via BLE, as it is neither feasible (nor important) to transmit the entire 31x24 array of temperature measurements. This evaluation will take place continuously upon new thermal data from the MLX90640 sensor (converted into thermal 'images' shown on the display), however, the transmission will only happen once the user presses the button, and the LED briefly flashes to indicate the transmission occurred.

Initially, it was necessary to define some specific variables related to the Bluetooth GATT (Generic Attribute Profile) specification, namely a custom Service and a custom Characteristic for storing the detection data of a hotspot. Next, an interrupt was created to execute the "sendDataBLE" method when the user pressed the button. The empirical rule to detect the hotspots will be executed and the BLE Characteristic value will be set to '1' (hotspot detected) or '0' (hotspot not detected) in 'notify' mode. To test the communication with the smartphone, the 'nRF Connect' mobile application was used in which the user can connect to the 'Thermal Mapping' device and access the custom Service and custom Characteristic associated with the thermal data. From there, the 'notify' option can be selected to receive values continuously (screenshots of the app can be found in Figure 4).

In this way, the TMM can effectively be extended with the communication capabilities of smartphones through their BLE connection, in which the smartphone works as a bridge between this the TMM and the Internet (i.e. to communicate to a server). The 'nRF Connect' application was used to test the BLE communication, but it's possible to develop a new application that leverages the smartphone's BLE technology to interact with the TMM.



20:21

Devices DISCONNECT

BONDED ADVERTISER THERMAL MAPPING 24:6F:28:36:64:86

CONNECTED NOT BONDED CLIENT SERVER

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Unknown Service
UUID: ab0828b1-198e-4351-b779-901fa0e0371e
PRIMARY SERVICE

Unknown Characteristic
UUID: 4ac8a682-9736-4e5d-932b-e9b31405049c
Properties: NOTIFY
Value: (0x) 30, "0"
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled

a)

20:22

Devices DISCONNECT

BONDED ADVERTISER THERMAL MAPPING 24:6F:28:36:64:86

CONNECTED NOT BONDED CLIENT SERVER

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Unknown Service
UUID: ab0828b1-198e-4351-b779-901fa0e0371e
PRIMARY SERVICE

Unknown Characteristic
UUID: 4ac8a682-9736-4e5d-932b-e9b31405049c
Properties: NOTIFY
Value: (0x) 31, "1"
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled

b)

Figure 4 - The data visualization from the TMM in the 'nRF Connect' app: a) displays the value '0' (no hotspot detected); b) displays the value '1' (hotspot detected).

Results and Discussion

The evaluation process had 2 phases: (1) determining the distance at which a more realistic flame would be reliably detected (and how an environment with high thermal "noise" would affect the TMM performance); (2) assessing the reliability of the hotspot detection algorithm. The first phase of tests was all performed using wood as fuel. Considering how the temperature of a wood-fueled flame far surpasses the maximum temperature the IR sensor can register, and that the fires used in field tests would be significantly larger, the tests were done at distances between 20 and 50 meters. For additional data points, some tests at 20 meters were made from a point around 5 meters below the flames and with a somewhat obstructed view.

Phase 1 of the Evaluation Process

The first set of tests was conducted on a partly cloudy day, using a barbecue pit containing various dried branches as fuel. The first test was conducted at 1,5 meters (for control purposes), which was then followed by various measurements made from a lower point at 20 meters. As can be observed in Figure 5, the results at 20 meters showed a few pixels far hotter than their surroundings (represented by the white squares seen in the display). Although satisfactory, these results raised the question of whether the impact of a high ambient temperature would dimmish the performance of the TMM, thus a second set of tests was performed.

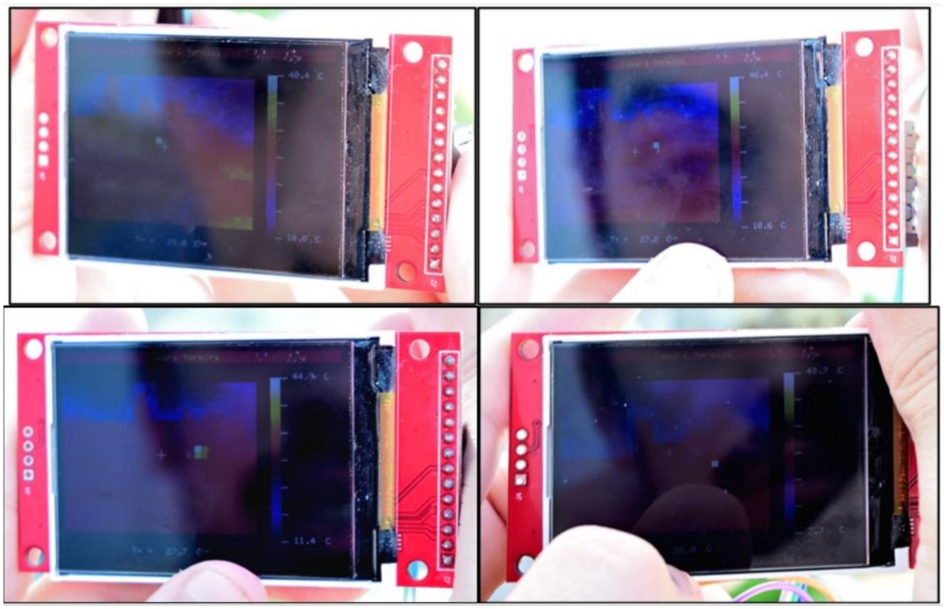


Figure 5 - Measurements at 20 m. Top left- Embers; Others - Flames burning.

The second set of tests was performed during clear sunny days with very high temperatures, as well as sandy ground which got very hot and provided a good source of thermal noise. The target flame was made in a barbecue pit, and dried wood was used as fuel. The measurements were made at distances between 26 and 50 meters. Taking the ambient temperature into account would always make the target considerably hotter than the air around it, the unlit source was measured at 1.5 meters for control purposes. As seen in Figure 6, the pit is considerably colder than the ground beneath it, thus it has no risk of false positives.

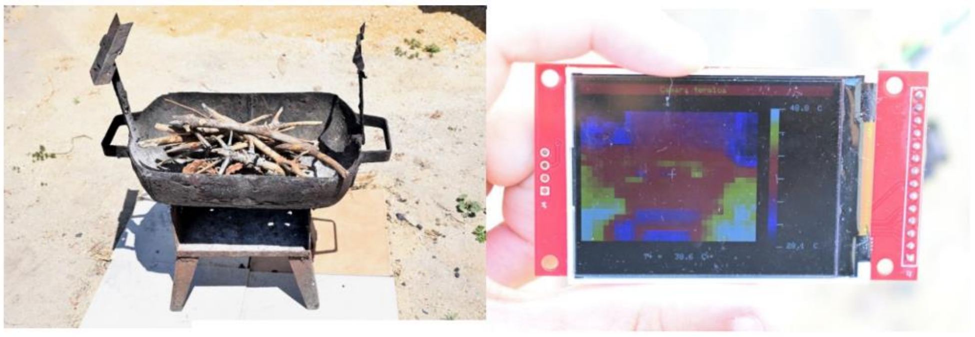


Figure 6 - Unlit source and measurement.

After this first control check, the flame was ignited for a second control test, in which the sensor captured temperatures close to its maximum range of 300ºC, at which point the measurements became somewhat unreliable. However, for this project, the detection of very high temperatures is far more important than the precision of the temperature measurements.

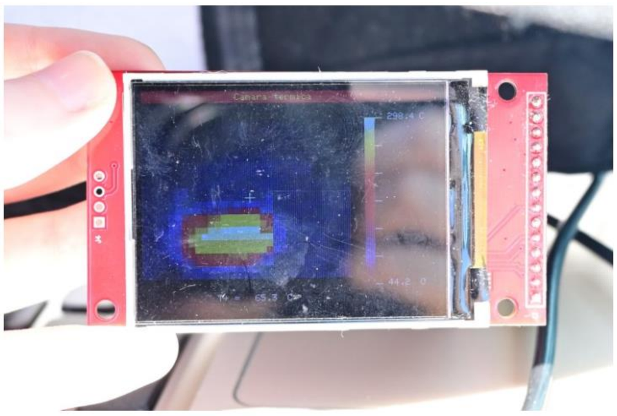


Figure 7 - Flame temperature measurement.

Following this second control measurement, the long-distance tests were performed. The first one was at 30 meters, and the testing environment is represented in Figure 8, with the target flame in the upper centre of the image.

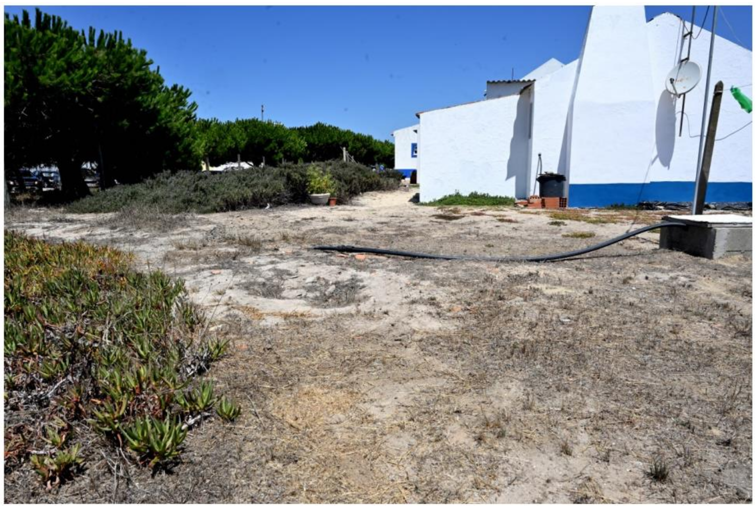


Figure 8 - Testing environment.

As can be seen in Figure 8, the object is difficult to perceive by the naked eye, thus serving as a good study of how well small flames or reignition sources would be detected using the device. This series of tests showed good results, as the TMM was able to reliably detect the flames consistently. However, at this range, the high noise levels from the ground made it somewhat difficult to detect the exterior of the barbecue pit reliably when the blaze did not crest above it.

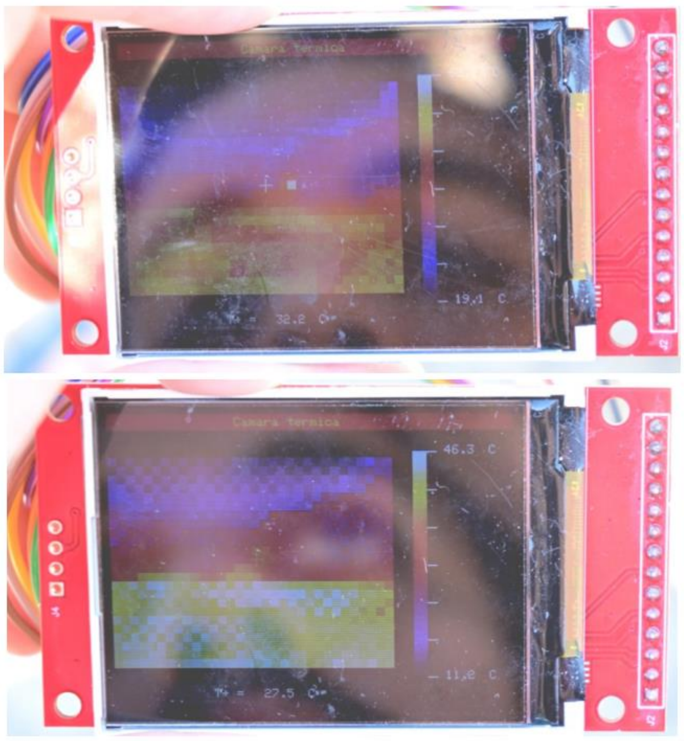


Figure 9 - Measurements with and without flames above the rim of the pit.

As seen in Figure 9, an open flame registers up to 63ºC, while just the metal exterior registers around 46ºC and is virtually indistinguishable from the ground. After these measurements were taken, the outside of the pit's temperature was measured at around 215ºC. After the first set of tests allowed for an estimation of the upper limit for reliable measurements, a second set of tests was performed at 50 meters, as this value was observed as the maximum distance at which the TMM would be used. These tests were somewhat positive, as it was possible to faintly detect the flame, but this was only possible due to prior knowledge of where the target was.

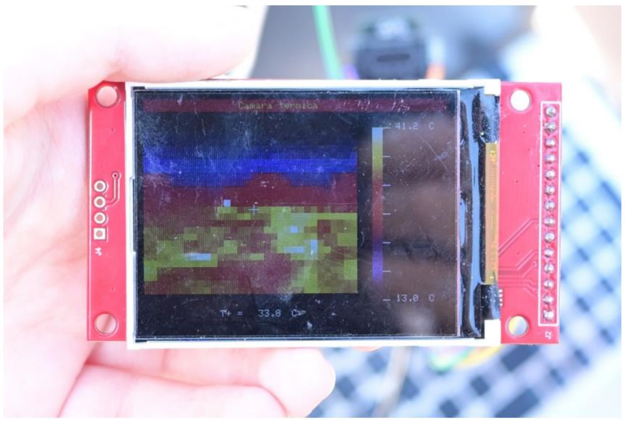


Figure 10 - Flame detected in the centre-left of the screen.

As the environmental conditions did not allow for testing at this range while there was less thermal noise, it is not possible to assess the reliability of the TMM when under less extreme conditions however, one can estimate that a hotspot could be reliably detected at the maximum proposed range when the ambient temperature surrounding it is less than 30ºC. The last set of tests was done at 26 meters to see if vegetation would obscure hotspots. The results were positive since despite the occlusion by the vegetation, the TMM was able to measure the temperature. These tests helped reinforce the idea that the TMM can be reliable when used in high-vegetation areas, as the overall thermal noise around potential hotspots will be less intense.

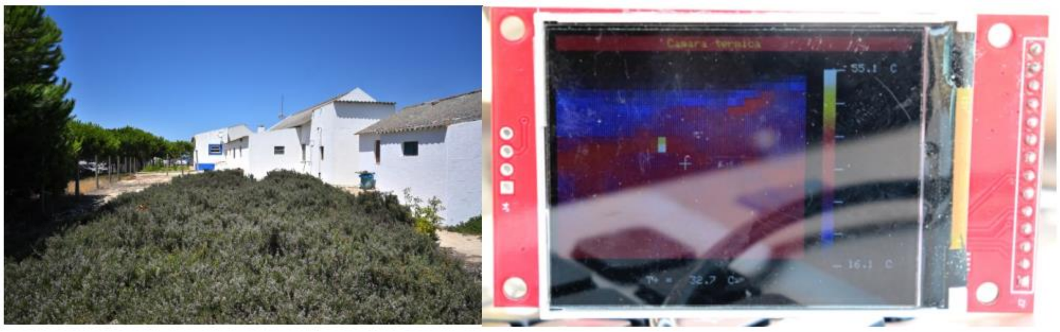


Figure 11 - Environment and measurements at 26 meters.

Phase 2 of the Evaluation Process

The Phase 2 of the Evaluation Process was meant to test the reliability of the hotspot detection algorithm. The detection of temperature outliers (hotspots) uses the empirical rule of the Normal distribution, where approximately 99.7% of the values in a normally distributed dataset lie within three standard deviations of the mean (meaning within the range of -3 times the standard deviation to +3 times the standard deviation). Every data point outside this range is considered an outlier. The results are summarized in Table 1.

*Table 1 - Test results outside (*minimum standard deviation reduced to 2.5).*

Dist. (m)	Hotspot	stDev	Mean Temp. (ºC)	Max Temp. (ºC)	Min Temp. (ºC)
0	DETECTED	60,33	57,83º	300,99º	19,87º
5 m	DETECTED	11,85	15,90º	208,66º	9,93º
10 m	DETECTED	5,92	11,50º	141,41º	1,30º
15 m	DETECTED	4,25	11,40º	84,07º	0,13º
20 m	DETECTED	4,31	10,56º	57,74º	-2,72º
22 m	DETECTED*	2,64	11,71º	62,36º	3,61º
± 23 m	DETECTED*	2,66	12,72º	57,72º	3,16º
+ 23 m	NOT DETECTED	1,92	12,62º	26,03º	5,36º

Conclusion

A few conclusions can be drawn from the 2 phases of the evaluation process. Phase 1 was divided into 2 parts: (1) Part 1 was meant to determine the distance at which a flame would be reliably detected; (2) Part 2 was meant to know how an environment with high thermal "noise" would affect the TMM performance. Phase 2 was meant to evaluate the reliability of the hotspot detection algorithm.

The multiple tests carried out in Phase 1 aimed to test the behaviour of the TMM in various circumstances, such as the influence of weather conditions on the detection of hotspots and their occlusion by vegetation. It was found that it was possible to detect hotspots at 20 m on a partly cloudy day in a sandy field. Tests were then carried out between 26 and 50 m, where it was possible to detect hotspots on clear sunny days (where there could be more thermal 'noise' disturbing the test results). The last set of tests was conducted at 26 m and verified the detection of hotspots even when vegetation occluded. The results of Phase 2 can be seen in Table 1, which shows that hotspots were detected up to a maximum distance of around 23 m. From this distance, the standard deviation in the pixel temperature measurements of the MLX90640 was so small that the developed algorithm was unable to detect hot spots.

Overall, the evaluation process showed that the TMM is ineffective in calculating real temperatures as the distance increases, however, this is not relevant as it does not influence the ability to detect hotspots at short distances. The greatest influence of this factor has to do with the performance of the hotspot detection algorithm, since as the distance increases, temperature measurements that are much higher than their surroundings become incrementally more difficult to read, thus the standard deviation becomes smaller until it is no longer possible to distinguish them from adjacent pixels. In any case, the detection of abnormally hotter pixels (which could be hotspots) can be achieved by the assessment of a human operator, although this is only possible up to a certain distance, beyond which the MLX90640 sensor cannot measure abnormally high temperatures. Therefore, it was found that the MLX90640 sensor may not have an acceptable performance for detecting hotspots at greater distances than those tested (which ranged from 20 to 50 m). On the other hand, the low refresh rate of only 16 Hz of the ILI9341 display hinders the correct verification of temperatures by the operator. The following future work in this solution is recommended:

- ➢ New firmware implementation with optimizations such as: (1) The ESP32 has a dual-core processor but only core 1 is used (with core 0 being occasionally used to read the button and transmit data via BLE). Reformulating the firmware so that the interaction with the ILI9341 display takes place on core 0 could theoretically increase the refresh rate from 16 Hz;
- ➢ Consider using a sensor other than the MLX90640;
- ➢ Consider using a microcontroller other than ESP32;
- ➢ Improving the connection of components would remove errors caused by connection interruptions.

References

Barrado, C., Messeguer, R., Lopez, J., Pastor, E., Santamaria, E., & Royo, P. (2010). Wildfire monitoring using a mixed air-ground mobile network. IEEE Pervasive Computing, 9(4), 24-32. https://doi.org/10.1109/MPRV.2010.54

Beighley, M., & Hyde, A. C. (2018). Portugal Wildfire Management in a New Era. Assessing Fire Risks, Resources and Reforms. Retrieved from https://www.isa.ulisboa.pt/files/cef/pub/articles/2018-04/2018_Portugal_Wildfire_Management_in_a_New_Era_Engish.pdf

Bhandari, P. (2023). Normal distribution: Examples, formulas, & uses. Scribbr. Retrieved from https://www.scribbr.com/statistics/normal-distribution/

Espressif Systems. (2024). ESP32 series datasheet. Retrieved from https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Melexis. (2018). MLX90640 32x24 IR array. Retrieved from https://cdn.sparkfun.com/assets/3/1/c/6/f/MLX90640-Datasheet.pdf

Plucinski, M. P. (2019). Fighting Flames and Forging Firelines: Wildfire Suppression Effectiveness at the Fire Edge. Current Forestry Reports, 5(1), 1-19. https://doi.org/10.1007/s40725-019-00084-5

Sullivan, A. L. (2017). Inside the Inferno: Fundamental Processes of Wildland Fire Behaviour. Current Forestry Reports, 3(2), 132-149. https://doi.org/10.1007/s40725-017-0057-0

VanderMeer, S., Ault, R., & Hvenegaard, S. (2011). An evaluation of handheld infrared cameras for ground initial attack crew use during wildfire mop-up operations. FPInnovations Technical Report for Wildland Fire Operations Research Group. Retrieved from https://library.fpinnovations.ca/media/FOP/8722.PDF

Wu, X., Dunne, R., Zhang, Q., & Shi, W. (2017). Edge computing enabled smart firefighting: Opportunities and challenges. In Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies (pp. 11). Association for Computing Machinery. https://doi.org/10.1145/3132465.3132475

Zhang, Y., Shu, Y., Qin, Y., Chen, Y., Lin, S., Huang, X., & Zhou, M. (2024). Resurfacing of underground peat fire: smouldering transition to flaming wildfire on litter surface. International Journal of Wildland Fire, 33(2). https://doi.org/10.1071/WF23128

Appendix A

Thermal Mapping Module firmware

```
#include "MLX90640_API.h"
#include "MLX90640_I2C_Driver.h"
#include <Wire.h>
#include <SPI.h>
#include <TFT_eSPI.h> // Hardware-specific library
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

#define TFT_CS   15
#define TFT_DC    2
#define TFT_MOSI 13
#define TFT_CLK  14
#define TFT_RST  26
#define TFT_MISO 12
#define TFT_LED  27
#define SERVICE_UUID "ab0828b1-198e-4351-b779-901fa0e0371e"
#define HOTSPOT_CHARACTERISTIC_UUID "4ac8a682-9736-4e5d-932b-e9b31405049c"
#define TA_SHIFT 8 //Default shift for MLX90640 in open air

BLECharacteristic *hotspotCharacteristic; // This BLE Characteristic will
receive data if YES or NO hotspot
bool deviceConnected = false; // If the BLE module has an active connection
boolean pressed = false;

TFT_eSPI tft = TFT_eSPI(); //(changed)

const byte MLX90640_address = 0x33; //Default 7-bit unshifted address of the
MLX90640
static float mlx90640To[768];
paramsMLX90640 mlx90640;

int xPos, yPos;
int R_colour, G_colour, B_colour;
int i, j;
float T_max, T_min, T_medium;
float T_center;
float stDev, meanTemperature;

class ServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};

void sendDataBLEInterrupt() {
  xTaskCreate(
               sendDataBLE,      /* Task function. */
               "SendDataBLE",    /* String with name of task. */
               5000,            /* Stack size in bytes. */
               NULL,            /* Parameter passed as input of the task */
               2,               /* Priority of the task. */
               NULL);           /* Task handle. */
}

// **************************************
// *************** SETUP ***************
// **************************************
void setup()
   {
    Serial.begin(115200);
    Wire.begin();
    Wire.setClock(5000000); //speed at 5Mhz (max)

    pinMode(17, INPUT);
    pinMode(0, OUTPUT);
    attachInterrupt(digitalPinToInterrupt(17), sendDataBLEInterrupt, CHANGE);

    if (isConnected() == false)
       {
        Serial.println("MLX90640 not detected at default I2C address. Please
check wiring. Freezing.");
        while (1);
       }

    Serial.println("MLX90640 online!");

    //Get device parameters - We only have to do this once
    int status;
    uint16_t eeMLX90640[832];

    status = MLX90640_DumpEE(MLX90640_address, eeMLX90640);

    if (status != 0)
       Serial.println("Failed to load system parameters");

    status = MLX90640_ExtractParameters(eeMLX90640, &mlx90640);

    if (status != 0)
       {
        Serial.println("Parameter extraction failed");
        Serial.print(" status = ");
        Serial.println(status);
       }

    //Once params are extracted, we can release eeMLX90640 array
    MLX90640_I2CWrite(0x33, 0x800D, 6401);    // writes the value 1901 (HEX) =
6401 (DEC) in the register at position 0x800D to enable reading out the
temperatures
    MLX90640_SetRefreshRate(MLX90640_address, 0x05); //setting to 16Hz

    pinMode(TFT_LED, OUTPUT);
    digitalWrite(TFT_LED, HIGH);
    tft.begin();
    tft.setRotation(1);
    tft.fillScreen(ILI9341_BLACK);
    tft.fillRect(0, 0, 319, 13, TFT_RED);
    tft.setCursor(100, 3);
    tft.setTextSize(1);
    tft.setTextColor(ILI9341_YELLOW);
    tft.print("Camara Termica foRESTER");
    tft.drawLine(250, 210 - 0, 258, 210 - 0, tft.color565(255, 255, 255));
    tft.drawLine(250, 210 - 90, 258, 210 - 90, tft.color565(255, 255, 255));
    tft.drawLine(250, 210 - 180, 258, 210 - 180, tft.color565(255, 255, 255));
    tft.setCursor(80, 220);
    tft.setTextColor(ILI9341_WHITE, tft.color565(0, 0, 0));
    tft.print("T(+) = ");

    // drawing the colour-scale
    // ========================

    for (i = 0; i < 181; i++)
       {
        getColour(i);
        tft.drawLine(240, 210 - i, 250, 210 - i, tft.color565(R_colour,
G_colour, B_colour));
       }


  ///////BLE SETUP//////
  BLEDevice::init("Thermal Mapping"); // nome do dispositivo bluetooth
  // Create the BLE Server
  BLEServer *server = BLEDevice::createServer(); //cria um BLE server
  server->setCallbacks(new ServerCallbacks()); //seta o callback do server
  // Create the BLE Service
  BLEService *service = server->createService(SERVICE_UUID);
  // Create a BLE Characteristic to show the detection (or not) of hotspot(s)
  hotspotCharacteristic =
service->createCharacteristic(HOTSPOT_CHARACTERISTIC_UUID,BLECharacteristic::PRO
PERTY_NOTIFY);
  hotspotCharacteristic->addDescriptor(new BLE2902());

  service->start();
  // Start advertising (descoberta do ESP32)
  server->getAdvertising()->start();
  }

// *********************************
// ************** LOOP **************
// *********************************
void loop()
   {
    for (byte x = 0 ; x < 2 ; x++) //Read both subpages
       {
        uint16_t mlx90640Frame[834];
        int status = MLX90640_GetFrameData(MLX90640_address, mlx90640Frame);
        if (status < 0)
           {
            Serial.print("GetFrame Error: ");
            Serial.println(status);
           }

        float vdd = MLX90640_GetVdd(mlx90640Frame, &mlx90640);
        float Ta = MLX90640_GetTa(mlx90640Frame, &mlx90640);

        float tr = Ta - TA_SHIFT; //Reflected temperature based on the sensor
ambient temperature
        float emissivity = 0.95;

        MLX90640_CalculateTo(mlx90640Frame, &mlx90640, emissivity, tr,
mlx90640To);
       }


    // determine T_min and T_max and eliminate error pixels
    // ====================================================
    mlx90640To[1*32 + 21] = 0.5 * (mlx90640To[1*32 + 20] + mlx90640To[1*32 +
22]);    // eliminate the error-pixels
    mlx90640To[4*32 + 30] = 0.5 * (mlx90640To[4*32 + 29] + mlx90640To[4*32 +
31]);    // eliminate the error-pixels

    T_min = mlx90640To[0];
    T_max = mlx90640To[0];

    for (i = 1; i < 768; i++)
       {
        if((mlx90640To[i] > -41) && (mlx90640To[i] < 301))
           {
            if(mlx90640To[i] < T_min)
               {
                T_min = mlx90640To[i];
               }

            if(mlx90640To[i] > T_max)
               {
                T_max = mlx90640To[i];
               }
           }
        else if(i > 0)   // temperature out of range
           {
            mlx90640To[i] = mlx90640To[i-1];
           }
        else
           {
            mlx90640To[i] = mlx90640To[i+1];
           }
       }

    // determine T_center
    // ==================
    T_center = mlx90640To[11* 32 + 15];
    T_medium = ((T_max + T_min)/2);

    float temperatureSum = 0.0;

    // STEP 1a, FIND THE MEAN: SUM ALL TEMPERATURES
    for (i = 1; i < 768; i++)
    {
      temperatureSum += mlx90640To[i];
    }

    // STEP 1b, FIND THE MEAN: DIVIDE BY THE # OF TEMPERATURE VALUES
    meanTemperature = temperatureSum / float(768);
    float sqDevSum = 0.0;

    // STEP 2, sum the squares of the differences from the mean
    for (i = 1; i < 768; i++)
    {
      sqDevSum += pow((meanTemperature - float(mlx90640To[i])), 2);
    }

    // STEP 3, TAKE THE SQUARE ROOT OF THAT TO CALCULATE THE STANDARD DEVIATION
    stDev = sqrt(sqDevSum/float(768));

    // drawing the picture
    // ===================
    for (i = 0 ; i < 24 ; i++)
    {
      for (j = 0; j < 32; j++)
      {
        mlx90640To[i*32 + j] = 180.0 * (mlx90640To[i*32 + j] - T_min) / (T_max - 
T_min);
        getColour(mlx90640To[i*32 + j]);
        tft.fillRect(217 - j * 7, 35 + i * 7, 7, 7, tft.color565(R_colour, 
G_colour, B_colour));
      }
    }

    tft.drawLine(217 - 15*7 + 3.5 - 5, 11*7 + 35 + 3.5, 217 - 15*7 + 3.5 + 5, 
11*7 + 35 + 3.5, tft.color565(255, 255, 255));
    tft.drawLine(217 - 15*7 + 3.5, 11*7 + 35 + 3.5 - 5, 217 - 15*7 + 3.5, 11*7 + 
35 + 3.5 + 5,  tft.color565(255, 255, 255));
    tft.fillRect(260, 25, 37, 10, tft.color565(0, 0, 0));
    tft.fillRect(260, 205, 37, 10, tft.color565(0, 0, 0));
    tft.fillRect(115, 220, 37, 10, tft.color565(0, 0, 0));
    tft.setTextColor(ILI9341_WHITE, tft.color565(0, 0, 0));
    tft.setCursor(265, 25);
    tft.print(T_max, 1);
    tft.setCursor(265, 205);
    tft.print(T_min, 1);
    tft.setCursor(120, 220);
    tft.print(T_center, 1);
    tft.setCursor(265, 115);
    tft.print(T_medium, 1);
    tft.setCursor(300, 25);
    tft.print("C");
    tft.setCursor(300, 205);
    tft.print("C");
    tft.setCursor(300, 115);
    tft.print("C");
    tft.setCursor(155, 220);
    tft.print("C");

    delay(10);
}

// ===============================
// ===== determine the colour ====
// ===============================
void getColour(int j)
   {
    if (j >= 0 && j < 30)
       {
        R_colour = 0;
        G_colour = 0;
        B_colour = 20 + (120.0/30.0) * j;
       }

    if (j >= 30 && j < 60)
       {
        R_colour = (120.0 / 30) * (j - 30.0);
        G_colour = 0;
        B_colour = 140 - (60.0/30.0) * (j - 30.0);
       }

    if (j >= 60 && j < 90)
       {
        R_colour = 120 + (135.0/30.0) * (j - 60.0);
        G_colour = 0;
        B_colour = 80 - (70.0/30.0) * (j - 60.0);
       }

    if (j >= 90 && j < 120)
       {
        R_colour = 255;
        G_colour = 0 + (60.0/30.0) * (j - 90.0);
        B_colour = 10 - (10.0/30.0) * (j - 90.0);
       }

    if (j >= 120 && j < 150)
       {
        R_colour = 255;
        G_colour = 60 + (175.0/30.0) * (j - 120.0);
        B_colour = 0;
       }

    if (j >= 150 && j <= 180)
```

```
       {
        R_colour = 255;
        G_colour = 235 + (20.0/30.0) * (j - 150.0);
        B_colour = 0 + 255.0/30.0 * (j - 150.0);
       }

   }

// Returns true if the MLX90640 is detected on the I2C bus
boolean isConnected()
   {
    Wire.beginTransmission((uint8_t)MLX90640_address);
    Serial.print("looking for i2c");
    if (Wire.endTransmission() != 0)
       return (false); //Sensor did not ACK

    return (true);
   }

// Sends hotspot data to client through Bluetooth BLE
void sendDataBLE(void * pvParameters ){
  // Debounce the button by reading its state multiple times
  int buttonState = digitalRead(17);
  // Handle voltage bouncing
  delay(10);
  if (buttonState == digitalRead(17)) {
    // The button state hasn't changed, so it's a valid press
    // Logic to fire once and not while the button is pressed
    if (!pressed && buttonState == HIGH) {
      pressed = true;
      String isHotspot = "-1";
      int str_len = isHotspot.length() + 3;
      char char_array[str_len];

      // To detect an hotspot - check the T_max and the standard deviation, if
both have abnormal values - hotspot detected
      if (T_max > 40.0 && stDev > 3.0) {
        Serial.println("HOTSPOT DETECTED!");
        isHotspot = "1";
      } else {
        Serial.println("NO HOTSPOT DETECTED!");
        isHotspot = "0";
      }

      // Send the data through Bluetooth BLE
      isHotspot.toCharArray(char_array, str_len);
      hotspotCharacteristic->setValue(char_array); //seta o valor que a
caracteristica notificará (enviar)
      hotspotCharacteristic->notify(); // Envia o valor para o smartphone

      // Blink LED to signal BLE data send
      digitalWrite(0, HIGH);   // turn the LED on
      delay(200);
      digitalWrite(0, LOW);    // turn the LED off
    } else if (pressed){
      digitalWrite(0, LOW);    // turn the LED off
      pressed = false;
    }
  }

  vTaskDelete(NULL);
}
```