THERMAL IMAGING CAMERA WITH RASPBERRY PI 3
by Vito Vincenzo Covella

Hardware used:

- A Raspberry Pi (Any Raspberry Pi should be fine; however since the OpenCV code used for the project performed much slower on the Raspberry Pi Zero W, I opted for the more powerful Raspberry Pi 3 B+).
- An MLX90640 32x24 pixel thermal camera (I bought it from here https://shop.pimoroni.com/products/mlx90640-thermal-camera-breakout); sensors with higher resolution are too expensive for my budget, on the other end the cheapest solution available in the market is a 8x8 sensor, which is a very low resolution. The MLX90640 was a good middleground.
- A picamera (I used the Kuman 5MP SC15, bought from here. https://www.amazon.it/Raspberry-Fotocamera-Notturna-Infrarossi-Supports/dp/B01ICNT3HC, we are not gonna need the two infrared LEDs).
- Some tactile buttons, used to take screenshots and switch on and off the device.
- An SPI LCD screen with ILI9341 (https://www.amazon.it/gp/product/B0749NHRV4)
- Two 3.7V rechargeable batteries, for example two samsung INR 18650 25r 3,7v 20a 2500mah batteries (https://www.ebay.it/itm/2x-Samsung-Li-Ionen-INR-18650-25R-3-7V-20A-2500mAh-Lithium-Akku-Akkubox)
- A bms protection and charging board (https://www.ebay.it/itm/2S-5A-8-4V-Li-ion-Lithium-18650-Battery-BMS-Packs-PCB-Protection-Board)
- a mini360 buck converter to allow us to step down from 8.4V to 5V (https://www.ebay.it/itm/360-DC-DC-Regolabile-Modulo-Converter-Alimentatore-3V-5V-16V-Power-MP2307-Chip/173967505621)
- A 9V 3A power supply (like this one https://www.amazon.it/gp/product/B01GRYFI6S/)
- A power plug
- A toggle switch
- A 3D printer in order to print the case of the device.
- Various jumper cables

Preliminary operations

Before actually diving into the details of the project, I suggest you to prepare your sd card and operating system. First of all I used Raspbian Stretch instead of Buster,

because the driver used to make the SPI LCD work properly doesn't seem to be compatible with Buster. Moreover, since I went for the headless installation process and used the Lite version, I installed the desktop manager through `sudo apt-get install raspberrypi-ui-mods` and other tools I would need throughout the making of the project, such as vim, git, python3's pip3. Moreover the picamera must be enabled from the menu accessible by typing `sudo raspi-config`. If you are going with the headless approach, I advise you to enable the VNC server: you're gonna need it to test some things with the capabilities offered by the desktop manager. Finally you must modify /etc/fstab in order to mount /tmp on RAM, because the code we're gonna use continuously writes thermal data on /tmp/heatmap.csv to make it accessible by other programs. Once you've done it, your /etc/fstab should look similar to this

```
proc            /proc           proc    defaults          0       0
PARTUUID=18c32f6c-01  /boot           vfat    defaults          0       2
PARTUUID=18c32f6c-02  /               ext4    defaults,noatime  0       1
tmpfs           /tmp      tmpfs defaults,noatime,nosuid,nodev,noexec,mode=1777 0  0
```

Installing the MLX90640 and making it work

Grab the code from https://github.com/pimoroni/mlx90640-library by typing `git clone https://github.com/pimoroni/mlx90640-library.git` then compile it following the instruction on the page keeping in mind we're gonna use the I2C mode. Install all the libraries needed for the code, as mentioned on the github page.
For some reasons if you execute make `make I2C_MODE=LINUX` you could get some error about the BCM2835 Library; you can avoid it by executing

```
make bcm2835
make clean
make I2C_MODE=LINUX
```

What we really need at the end of the process is the libMLX90640_API.a, which will be linked with the driver that creates the heatmap.csv (more on that later).
Now you can connect the MLX90640 to the Raspberry I2C pin and power pins. I used the following wiring scheme (where the number of the pin is the actual physical pin numbering system, not the GPIO/BCM system)

3V	pin 1
SDA	pin 3
SCL	pin 5
GND	pin 9

Now, to test that everything works fine, if you type from mlx90640-library directory `examples/sdlscale` you should see the sensor working in an SDL window.

As mentioned before, what we really need from the compilation process is the libMLX90640_API.a. This file must be linked with the mlx90640_driver obtained by compiling the .cpp retrievable from https://github.com/leswright1977/mlx90640_python. leswright1977's idea was to overlap the output of the picamera with the thermal image, in order to see temperature data and the borders of the objects in the image.

The instruction to do what we have described are these:

```
git clone https://github.com/leswright1977/mlx90640_python.git
cp mlx90640_python/mlx90640_driver.cpp
mlx90640-library/mlx90640_driver.cpp
cp mlx90640_python/thermalcam.py mlx90640-library/thermal.py
cd mlx90640-library
g++ -c mlx90640_driver.cpp -o mlx90640_driver.o
g++ mlx90640_driver.o libMLX90640_API.a -o mlx90640_driver
```

If you want to detect temperatures higher than 99.99 °C, comment line 115 of mlx90640_driver.cpp before compiling it.

The thermal.py code must be modified in order to make it work fullscreen and stretch the image to occupy all the 320x240 pixels that will be available on the screen.
The final file should look like this:

```
import cv2
import time
import numpy
import random
import math
import traceback
import io

#a hack to wake our bus if it hangs....
import subprocess
p = subprocess.run(['i2cdetect', '-y','1', '0x33', '0x33'])
########################################

from picamera.array import PiRGBArray
```

```
from picamera import PiCamera
from picamera.exc import PiCameraRuntimeError
from picamera.exc import PiCameraValueError
import numpy as np
import sys

sys.stdout = open("/home/pi/thermaloutput.log", "w")
sys.stderr = open("/home/pi/thermalerr.log", "w")

time.sleep(7)
camera = PiCamera()
camera.resolution = (288, 368) #start with a slightly larger image so
we can crop and align later!
camera.framerate = 20
#PiCamera.CAPTURE_TIMEOUT = 60
rawCapture = PiRGBArray(camera, size=(288, 368))

# allow the camera to warmup
time.sleep(0.1)

nmin = 0
nmax = 255
alpha1 = 0.5
alpha2 = 0.5

prevData = []
temp = 0
for frame in camera.capture_continuous(rawCapture, format="rgb",
use_video_port=True):
   # Capture frame-by-frame
   frame = frame.array
   #frame = cv2.flip( frame, 0 ) #flip if neccesary
   #frame = cv2.flip(frame, 1)

   #crop and align visible image...
       #crop image y start yend, xstart xend; original value
[5:325,10:250]
   frame = frame[5:325, 10:250]
```

```
   heatmap = np.zeros((32,24,3), np.int32) #create the blank image to
work from

   data = np.fromfile('/tmp/heatmap.csv', dtype=float, count=-1,
sep=',') #get the data
   if np.array_equal(data,prevData):
       print('Data stall...Probing i2c')
       p = subprocess.run(['i2cdetect', '-y','1', '0x33', '0x33'])

   prevData = data

   index = 0
   #add to the image
   if len(data) == 768:
       for y in range (0,32):
           for x in range (0,24):
               val = (data[index]*10)-100
               if math.isnan(val):
                   val = 0
               if val > 255:
                   val=255
               #print(index)
               #print(data)

               heatmap[y,x] = (val,val,val)

               if(y == 16) and (x == 12):
                   temp = data[index]
               index+=1
       heatmap = cv2.flip(heatmap, -1 ) #flip heatmap to match image
       prev_heatmap = heatmap #save the heatmap in case we get a data
miss

   else:
       print("Data miss...Loading previous thermal image")
   try:
       heatmap = prev_heatmap
   except:
       print("Previous heatmap does not exist!")
```

```
   heatmap = cv2.normalize(heatmap,None,nmin,nmax,cv2.NORM_MINMAX,
cv2.CV_8U)
   heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
   heatmap =
cv2.resize(heatmap,(240,320),interpolation=cv2.INTER_CUBIC)
   heatmap = cv2.flip(heatmap, 0)



   # Display the resulting frame
   window_name = 'Thermal'
   cv2.namedWindow(window_name,cv2.WND_PROP_FULLSCREEN)
   cv2.moveWindow(window_name, 0, 0)
   cv2.setWindowProperty(window_name, cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)

   #Sharpen the image up so we can see edges under the heatmap
   kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
   frame = cv2.filter2D(frame, -1, kernel)

   frame = cv2.addWeighted(frame,alpha1,heatmap,alpha2,0) #combine the
images

   cv2.line(frame,(120,150),(120,170),(0,0,0),1) #vline
   cv2.line(frame,(110,160),(130,160),(0,0,0),1) #hline

   cv2.putText(frame,'Temp: '+str(temp), (10, 15),\
   cv2.FONT_HERSHEY_SIMPLEX, 0.6,(0, 255, 255), 1, cv2.LINE_AA)
#    M = cv2.getRotationMatrix2D((160,120), 90, 1)
#    rotated90 = cv2.warpAffine(frame, M, (240, 320))

#    cv2.imshow('Thermal', rotated90)
   #custom added
   frame = cv2.resize(frame, (420, 320), interpolation=cv2.INTER_CUBIC)
   cv2.imshow('Thermal',frame)

   res = cv2.waitKey(1)
   #print(res)

   if res == 113: #q
       break
```

```
    if res == 97: #a
        nmin += 10
        print(nmin)
    if res == 122: #z
        nmin -= 10
        print(nmin)
    if res == 115: #s
        nmax += 10
        print(nmax)
    if res == 120: #x
        nmax -= 10
        print(nmax)
    if res == 100: #d
        alpha1 += 0.1
        alpha2 -= 0.1
    if res == 99: #c
        alpha1 -= 0.1
        alpha2 += 0.1

    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)



print("Exiting application")
cv2.destroyAllWindows()
```

Feel free to hack the code inside the two for loops, used to get the heatmap value from the array “data”, in order to adjust it to your purposes. Right now, if for example there’s a flame on the scene, the code emphasizes in red all the temperatures above a certain threshold (35.5 °C), which is useful if you want to see in red the hot air around the flame. If instead you visually want less sensibility to temperatures higher than 35.5 °C and more precision (for the example above, the flame in red and everything else in blue, light blue or yellow), use `val = data[index]` and remove the check `if val > 255: val = 255`. Whatever you decide to do, if you leave the rest unchanged, the text visualized in the upper part of the screen will always show the correct temperature detected at the center of the image, where the cross is located.

The key modifications applied to the original code are the added lines to make it fullscreen, the lines to resize the output image and the instructions to flip the heatmap image, more specifically

```
window_name = 'Thermal'
cv2.namedWindow(window_name,cv2.WND_PROP_FULLSCREEN)
cv2.moveWindow(window_name, 0, 0)
cv2.setWindowProperty(window_name, cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)
frame = cv2.resize(frame, (420, 320), interpolation=cv2.INTER_CUBIC)
```

and

```
heatmap = cv2.flip(heatmap, 0)
```

There are also some instruction to redirect stdout and stderr to specific files for debugging purposes, feel free to remove them if you don't need them.
Moreover there's a 7 seconds delay to warmup the picamera and reduce the frequency of crashes during the firsts seconds (more on the paragraphs "Thermal cam monitor" and "Conclusions and further developments").
Finally to correctly detect temperatures below 0 °C there are other modifications to be done: the heatmap variable must contain data of type np.int32 instead of np.uint8 (`heatmap = np.zeros((32,24,3), np.int32)`) and the normalize function must return data of type cv2.CV_8U to make cv2.applyColorMap work correctly (`heatmap = cv2.normalize(heatmap,None,nmin,nmax,cv2.NORM_MINMAX, cv2.CV_8U)`).

To make the python file work, you should install opencv-python and other libraries, with the following command: `pip3 install opencv-python; sudo apt-get install -y libcblas-dev libhdf5-dev libhdf5-serial-dev libatlas-base-dev libjasper-dev  libqtgui4  libqt4-test`.
Now in one terminal tab (on a VNC client or with hdmi screen, keyboard and mouse) execute mlx90640_driver by typing `./mlx90640_driver` and on the other tab give the command `python3 thermal.py`. If you've done everything right, you should see a fullscreen video with thermal images. Remember to put the two cameras (the picamera and the sensor) as close to each other as possible.

Installing the SPI LCD screen

If you've got the same screen listed in the hardware section, this is the wiring for it (again, the pin number is the physical one, not the BCM numbering system):
MISO -> 21
LED -> 11
SCK -> 23
MOSI -> 19
DC -> 15
RESET -> 13

CS -> 24
GND -> GND
VCC -> 5V
We are not gonna use the touchscreen and SD cart slot of the screen. The driver we will use mirrors what gets sent to the hdmi port. For this reason, we have to tweak the /boot/config.txt by adding the following lines and then reboot the system:

```
display_rotate=1
hdmi_group=2
hdmi_mode=87
hdmi_cvt=240 320 60 1 0 0 0
hdmi_force_hotplug=1
```

After rebooting, you must download the SPI driver here https://github.com/juj/fbcp-ili9341 by cloning the repository with `git clone https://github.com/juj/fbcp-ili9341.git`, then follow the instruction to compile it by using the proper parameters for you LCD screen. If you have the same LCD I bought, give the following commands:

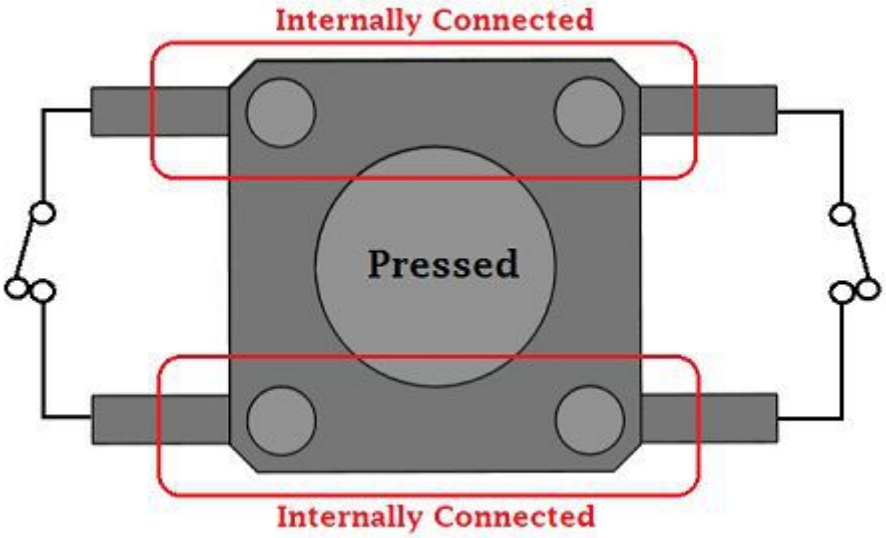
```
sudo apt-get install cmake
cd fbcp-ili9341
mkdir build
cd build
cmake -DILI9341=ON -DGPIO_TFT_DATA_CONTROL=22
-DGPIO_TFT_RESET_PIN=27 -DGPIO_TFT_BACKLIGHT=17
-DSPI_BUS_CLOCK_DIVISOR=6 -DARMV8A=ON -DBACKLIGHT_CONTROL=ON
-DSTATISTICS=0 ..
make -j
```

To execute the driver (which must be up and running all the time for the LCD screen to work properly), give `sudo ./fbcp-ili9341` on the terminal. I will tell you how to autostart the driver in the appropriate section.

Taking screenshot with a tactile button

I wanted to allow the user of the device to take screenshots of the thermal image when pressing a button and saving the screenshot in ~/shots. In order to do this I

decided to use a tactile button, which looks like this:  . The schematics for this button are the following ones:



This means that to make it work properly all we need is to connect one pin to a GPIO pin of the Raspberry Pi (I chose pin 32, a.k.a. GPIO12) and the other to GND. We can leave untouched the other two pins of the button.
Then you must write the code to take the screenshots, but before doing that, make sure to have scrot installed on your system by typing `sudo apt-get install scrot`. Also install RPi.GPIO with `pip3 install --user RPi.GPIO`. Finally, create the shots directory in your home folder with

```
cd ~
mkdir shots
```

and the python file that will have the code with

```
cd ~
mkdir piscreenshot
cd piscreenshot
vim pyscreenshot.py
```

The code used to take screenshots must be able to handle the debouncing problem. Moreover in order not to “peg” the CPU, we are going to check for an interrupt instead of relying on polling. The final code looks like this:

```
import RPi.GPIO as GPIO
import threading
```

```
import os
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
   time_stamp = time.time()
   GPIO.wait_for_edge(12, GPIO.RISING)
   time_now = time.time()
   if (time_now - time_stamp) >= 0.3:
       os.system("scrot '%Y-%m-%d-%s_$wx$h.png' -e 'mv $f ~/shots/'")
       #print("button pressed")
```

You can test the code by typing `python3 pyscreenshot.py` and then pressing the button to take the screenshots. This is another application that must autostart at the booting of the device, we will see how to do that in another section of this document.

Thermal cam monitor

Since the instruction `camera.capture_continuous(rawCapture, format="rgb", use_video_port=True)` sometimes makes the application crash during the first seconds (I've checked the error log and tried to fix it, without success; however putting a 7 seconds delay makes the crashes occur less frequently), we are going to code a monitor to make it restart in case of accidental crashes. This monitor launches the application thermal.py and waits until it terminates; when this happens, it restarts thermal.py.

```
from subprocess import Popen
import sys

sys.stdout = open("/home/pi/thermalwatchout.log", "w")
sys.stderr = open("/home/pi/thermalwatcherr.log", "w")

while True:
   p = Popen("/usr/bin/python3 " +
"/home/pi/mlx90640-library/thermal.py", shell=True)
   p.wait()
```

Let's save this file as thermalwatcher.py.

Shutdown button

I wanted the raspberry pi to shutdown properly. For this reason I coded another python script that listens to a button press in order to power off the device. This time I wanted the button to be held down for 3 seconds before powering off the device, in order to prevent accidental shutdowns. This is the python script I used, inspired by https://github.com/TonyLHansen/raspberry-pi-safe-off-switch:

```
#!/usr/bin/env python3
from gpiozero import Button
from signal import pause
import os, sys

offGPIO = 16
holdTime = 3

# the function called to shut down the RPI
def shutdown():
   os.system("sudo shutdown -h now")

btn = Button(offGPIO, hold_time=holdTime)
btn.when_held = shutdown
pause()    # handle the button presses in the background
```

I saved this file as pyshutdown.py inside /home/pi/pishutdown folder.

Autostarting applications

I won't use the touchscreen, so in order for the device to operate properly, I decided to make some application autostart at boot time. There are two different applications: the ones that don't need LXDE up and running and the ones that do need the functionality of the desktop manager. For this reason some application will be autostarted by modding rc.local, others by modding /etc/xdg/lxsession/LXDE-pi/autostart (or ~/.config/lxsession/LXDE-pi/autostart if you have this file on your system).
Let's start by installing unclutter with `sudo apt-get install unclutter`, since we need it to make the mouse cursor disappear after booting up. Then disable the screensaver from the screensaver configuration of the kiosk desktop manager.
After that, open /etc/rc.local with `sudo vim /etc/rc.local` and add the following lines before the line `exit 0`:

```
sudo /home/pi/fbcp-ili9341/build/fbcp-ili9341 &
/home/pi/mlx90640-library/mlx90640_driver &
```

```
/usr/bin/python3 /home/pi/pishutdown/pyshutdown.py &
```

These lines are needed in order to autostart the driver of the LCD screen, the MLX90640 logger that continuously writes on /tmp/heatmap.csv and the script that handles the shutdown button.

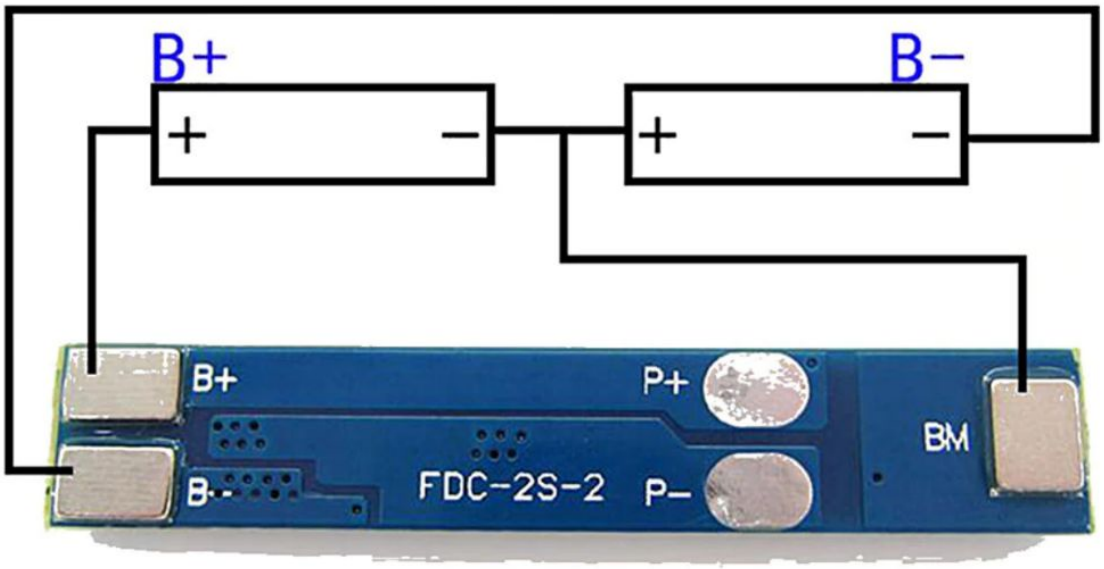
Finally add the following line to your /etc/xdg/lxsession/LXDE-pi/autostart (or ~/.config/lxsession/LXDE-pi/autostart if you have this file on your system):

```
@xset -dpms
@xset s off
@unclutter -idle 0
@/usr/bin/python3 /home/pi/mlx90640-library/thermalwatcher.py
@/usr/bin/python3 /home/pi/piscreenshot/pyscreenshot.py
```

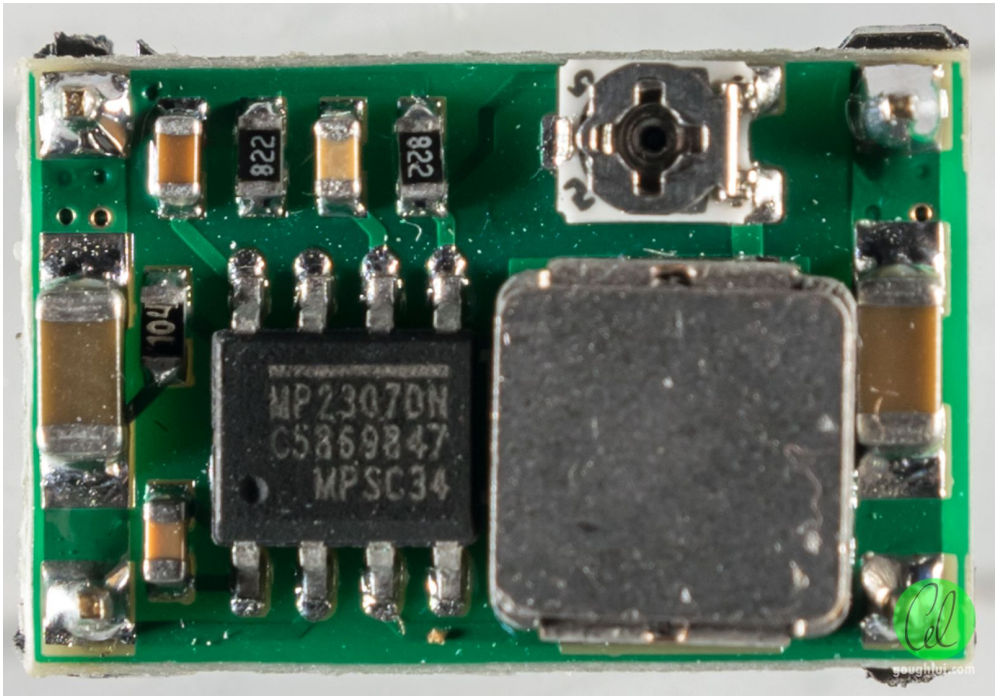
These lines will make sure to autostart the python application that will visualize the thermal video feed and the python application that will take screenshots when the tactile button will be pressed.

Charging circuit and power supply

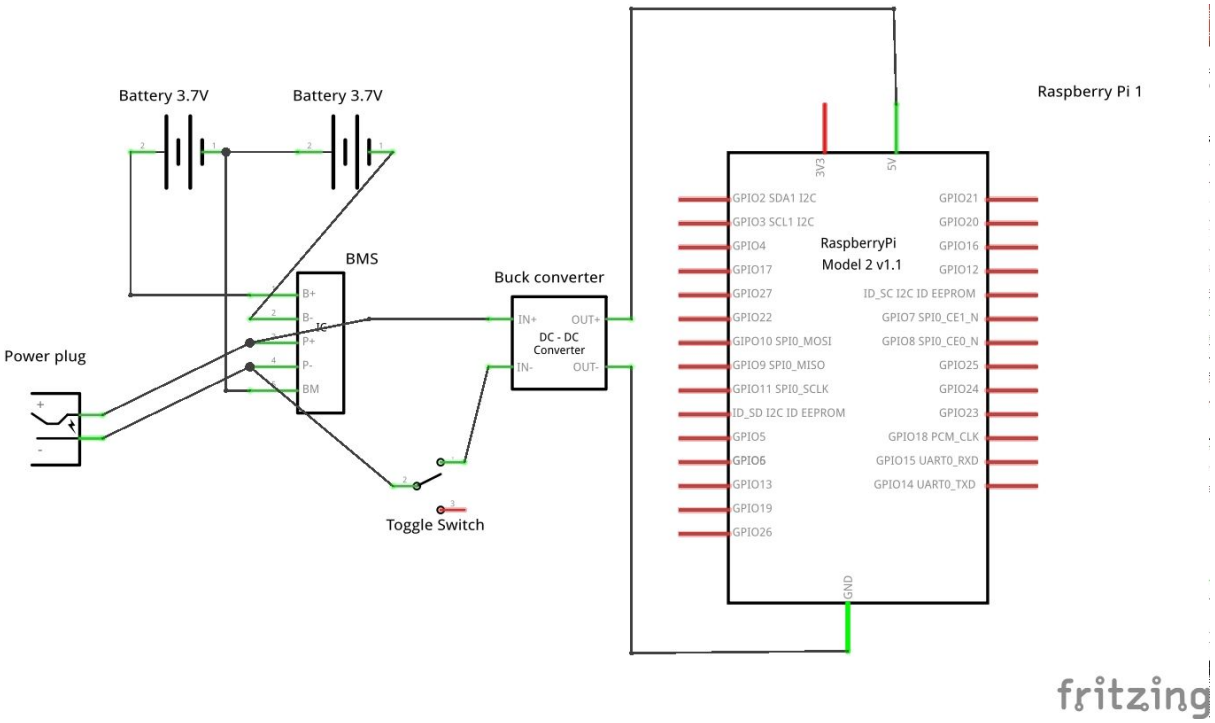
In order to make the device portable I decided to use two 3.7V (4.2V when fully charged) batteries. These batteries must be properly connected to a BMS (Battery Management System) in order to recharge them properly. More specifically, I used a HX-2S-01, shown on the figure below. P+ and P- will be connected to a power plug.



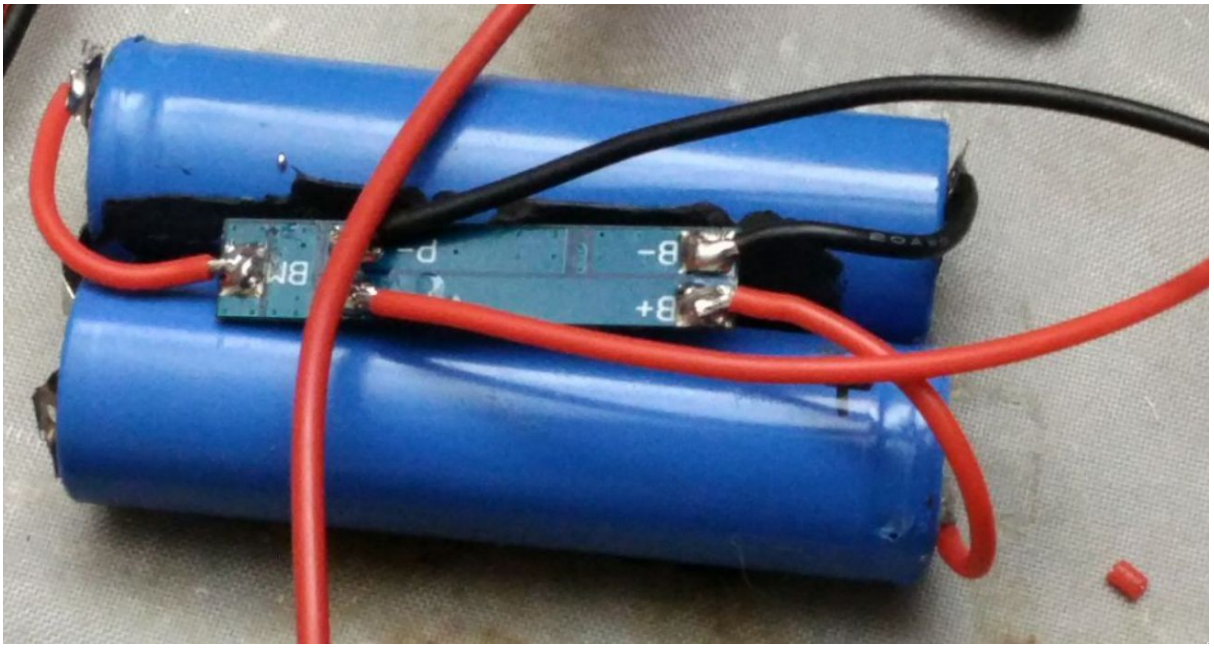
Finally I needed to step down from 8.4V to 5V, which is the right voltage to power up the Raspberry Pi 3. For this reason I used a mini360 buck converter based on the chip MP2307, like the one shown on the figure below. Keep in mind that the converter must be regulated before using it, you can do it with a screwdriver and a multimeter placed on the output to measure voltage.

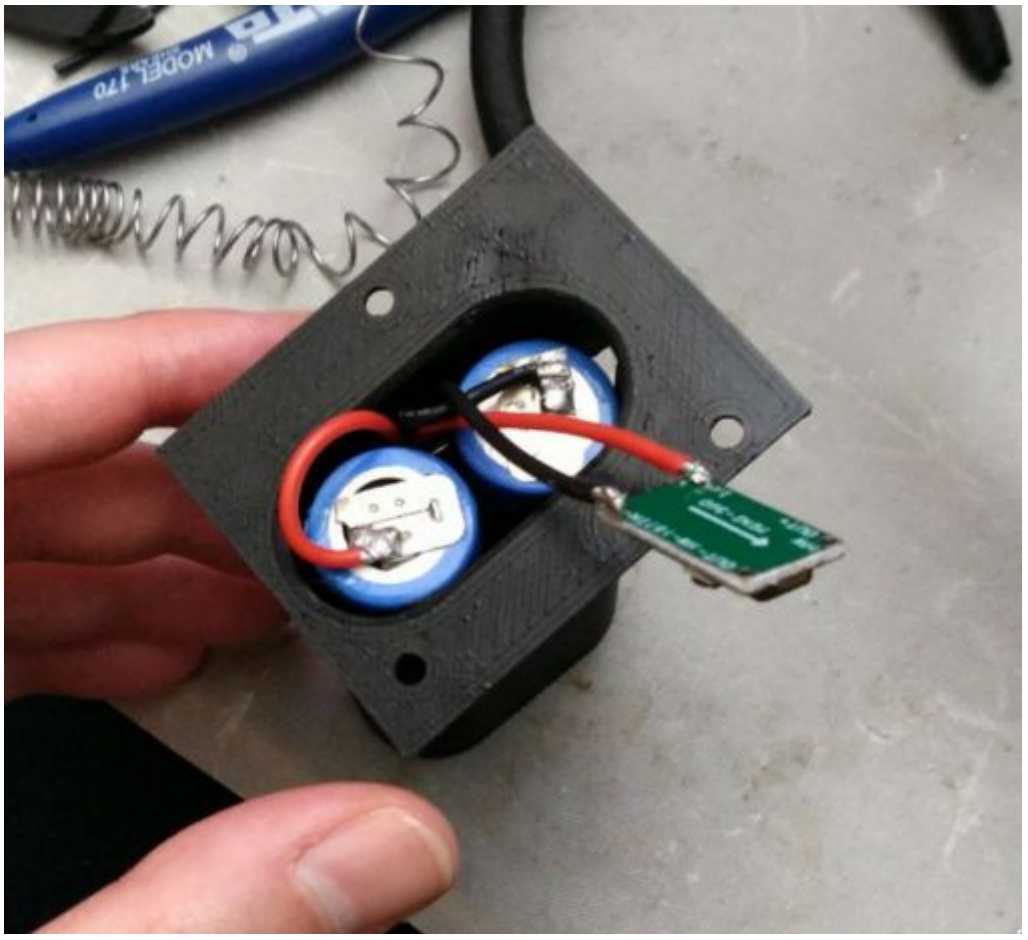


P+ of the BMS gets connected to the positive input of the DC-DC buck converter, while P- will be connected to an on-off toggle switch. One of the output of the toggle switch is connected to the negative input of the buck converter in order to close the circuit when the toggle is on. Finally the positive output of the buck converter is connected to a 5V pin of the Raspberry Pi 3, while the negative output is connected to a GND pin.
The full schematics, created using the software Fritzing, are shown below.



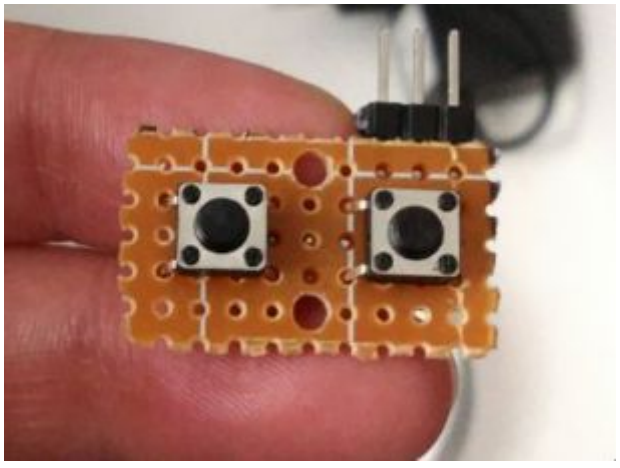
Below there are some images that show the construction process.

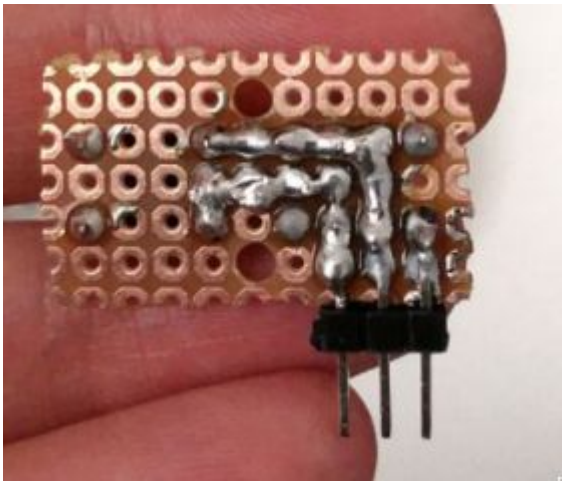




Push buttons circuit

In order to have a support structure for the push buttons when inserting the device in the 3D printed case, I put the two tactile push buttons previously described (shutdown button and screenshot button) on a veroboard and connected them on a 3 pin male header. The figures below show the result and the trails used to connect the buttons to the pins: the middle pin is the shared ground, while the pins to the left and to the right will be connected to two different GPIOs pins of the Raspberry Pi.



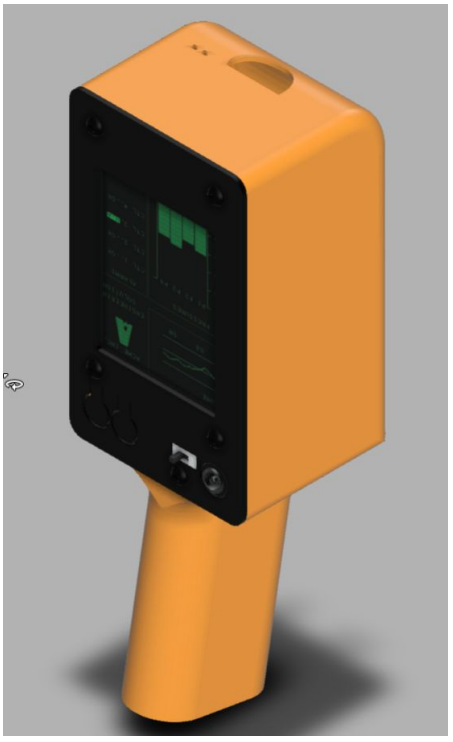


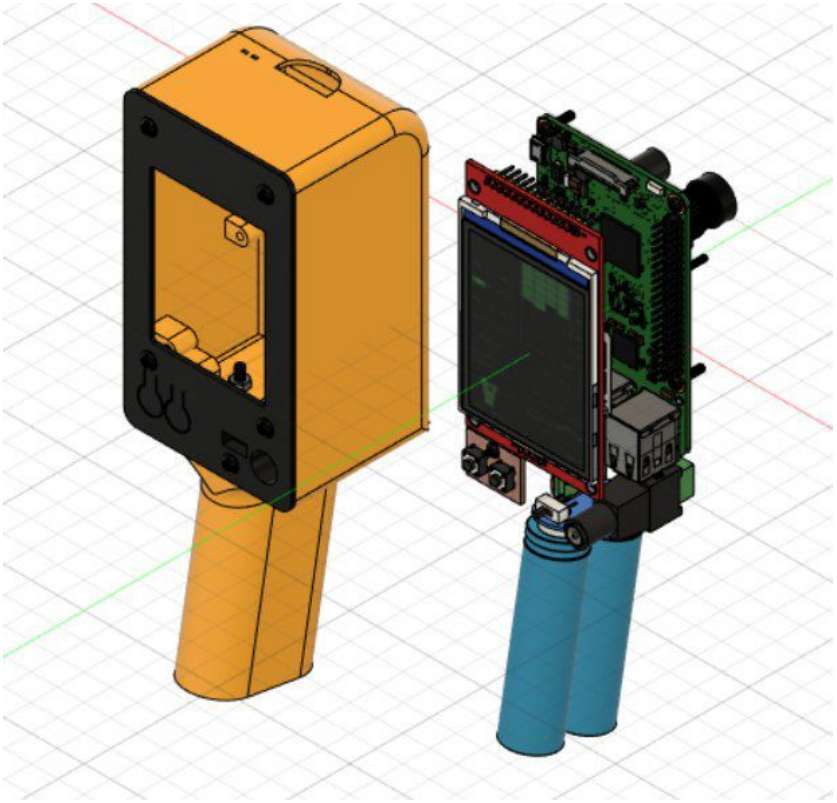
3D printed case

The case has been modeled using Autodesk Fusion360 (it can be used freely for non commercial use) and it is available here: https://a360.co/2KxB6Wr. In case you can't use Fusion360, the STLs of the components are available in zipped format here: https://drive.google.com/file/d/169e0E9VViFzuFO-B8-8mtHYRwapbZudJ/view?usp=sharing.

Each component of the case has been exported in STL format, then feeded into PrusaSlicer (https://github.com/prusa3d/PrusaSlicer), in order to generate the G-Code for the 3D printer, and finally printed.

Below there are some rendering of the case and an image that shows how the component will be organized inside it.

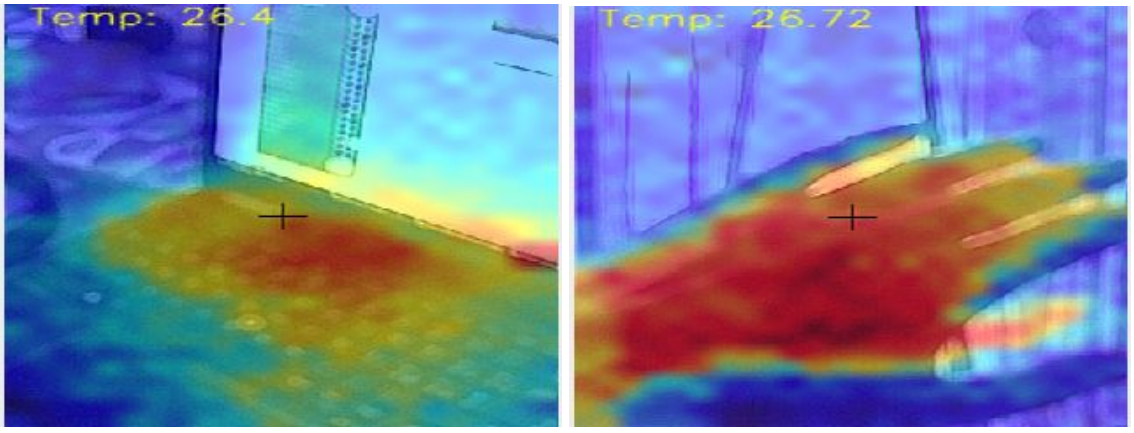




The final result

Here’s what the device looks like after assembling and making it work.

Below there are some examples of screenshots taken from the device, more specifically they show the most heated parts of my laptop and my hand.



Conclusions and further developments

Even though the device works fine at the end of the development and assembly process, a few things could be improved. First of all, there's a 7 seconds delay between booting up the desktop environment and launching the application responsible for visualizing the thermal images. This delay is needed in order to make the picamera ready and avoid initial crashes of the application. Maybe one could devise a solution to this problem while also avoiding initial crashes. Right now during

the firsts second random crashes could still occur, but, thanks to the delay, they are less frequent and anyway the monitor coded will readily restart the application.
Another thing that could be improved is the addition of a visual feedback when making screenshots. Right now, when the appropriate button is pressed, a screenshot is taken, but there's no feedback of the process.
Finally more buttons could be added in order to handle the nmin and nmax parameters of the thermal.py script, for example allowing the user to lower nmin and reduce background thermal noise.

License