Read me first:
This document is split into several sections, each describing a different problem, our solution to that problem, and guidance for how to use our code to solve it. Code (Python and MATLAB) is available at:

https://github.com/bblonder/flir_thermal_control
https://github.com/bblonder/thermal_analysis

The first sections are devoted to logging data from a FLIR camera, while the latter sections are devoted to calibrating those data and extracting useful temperature measurements from time-series measurements. `calibrated_temperature_simple.m`, `align_thermal`, and `analyze_image.m` are likely the most useful functions for other investigators.

Caveats:
The below code and documentation are provided to guide other investigators, and are not meant to be turnkey solutions for other tasks, given that the code has been closely adapted to the particular sensors and needs of our particular projects. Rather, the code should be seen as a reference and resource that can be extended and adapted for particular use cases. We encourage prospective users to contact us if they have questions or concerns.

More portable code is currently being developed (including more user-friendly code to control the camera via MATLAB rather than Python).

Contact:
Benjamin Blonder
bblonder@gmail.com

To automatically log frames from a thermal camera on a single board computer:
The below instructions assume that you have a FLIR A615 camera connected via Gigabit ethernet to an Odroid C2 computer running Linux (Ubuntu 14) with Python installed, and also that the C2 is connected to the Odroid Weather Board 2 sensor suite, a USB GPS unit, a USB camera, and a USB memory stick (for data logging). Then follow the below instructions to configure the system. The instructions should be considered a rough guide as every system's exact configuration will differ slightly. The system configuration steps are indicated in black and may differ between systems; the core specific details required to use our code are highlighted in red.

```
# make sure boot.ini file in partition is set to a useful
resolution with VGA output

# install basics
sudo apt-get update
sudo apt-get install git
sudo dpkg --purge mali-fbdev
sudo apt-get install python-numpy

# now get image processing kits
sudo apt-get install libjpeg-dev libfreetype6 libfreetype6-
dev zlib1g-dev
sudo apt-get install python-imaging

# get python setup stuff
sudo apt-get install python-setuptools

# follow
https://github.com/sightmachine/SimpleCV/wiki/Aravis-
(Basler)-GigE-Camera-Install-Guide

sudo apt-get install autoconf intltool libxml2-dev python-
gobject-dev gobject-introspection gtk-doc-tools
libgstreamer0.10-dev python-gst0.10-dev
sudo apt-get install autoconf intltool python-gobject-dev
gobject-introspection gtk-doc-tools libgstreamer0.10-dev
python-gst0.10-dev libxml2-dev
sudo apt-get install libgtk-3-dev libnotify-dev
libgstreamer1.0 libgstreamer-plugins-base1.0-dev
gstreamer1.0-plugins-bad
sudo apt-get install libgtk-2-dev
sudo apt-get install python-pip
pip install matplotlib
sudo apt-get install libopencv-dev python-opencv
sudo apt-get install python-matplotlib

git clone https://github.com/sightmachine/aravis.git
```

```
cd aravis
./autogen.sh --enable-viewer --enable-gst-plugin --enable-
introspection=yes
make
sudo make install

# set path variables
pico .bashrc
export GI_TYPELIB_PATH=$GI_TYPELIB_PATH=~/aravis/src
export LD_LIBRARY_PATY=~/aravis/src/.libs
export PYTHONPATH=~/aravis

# also install python aravis from
https://github.com/SintefRaufossManufacturing/python-
aravis/blob/master/aravis.py
python setup.py install

# add weather board
sudo modprobe aml_i2c
ls /dev/i2c-*
echo "aml_i2c" | sudo tee -a /etc/modules

git clone https://github.com/hardkernel/wiringPi
sudo ./build

# install wiring pi
git clone https://github.com/hardkernel/WiringPi2-
Python.git
cd WiringPi2-Python
git submodule init
git submodule update

# install vidoestream
sudo pip install imutilsx

# install GPS control
sudo apt-get install python gpsd gpsd-clients

# get camera working
sudo apt-get install cmake
sudo apt-get install libusb-dev libusb-1.0-0-dev
sudo apt-get install pkg-config

# logout and login

# get ethernet working
sudo ip link set eth0 down
sudo ifconfig eth0 mtu 3710 # no bigger possible
```

```
sudo avahi-autoipd eth0 &
# repeat three times (first two don't work)

# edit /etc/network/interfaces as suggested below
# autho eth0
# iface eth0 inet static
# address 169.254.100.1
# netmask 255.255.0.0
# mtu 3700

# edit /etc/NetworkManager/NetworkManager.conf as below
# [main]
# plugins=ifupdown,keyfile
# dns=dnsmasq
# [ifupdown]
# managed=false

# create /media/usbexternal using fdisk for /dev/sda1

# edit /etc/default/gpsd to include
# DEVICES="/dev/ttyACM0"

# download all files from
https://github.com/bblonder/flir_thermal_control and place
in a directory ~/control/

# in /etc/rc.local add a line for executable script:
/home/odroid/Desktop/control/run_thermal_control.sh

# restart the system – the control program will now begin
working automatically.
```

To configure the software:

You can edit the file control/thermal_control.py to determine how logging occurs. There are several parameters that can be changed in the header of this file. Default values are highlighted below.

```
# set the interval between acquisitions in seconds
sleep_time = 5
# set the number of frames between logging a visible image
visible_interval = 10
# set interval at which a PNG is saved in addition to raw
thermal data
infrared_interval = 10
# set frame gap between weather sensor updates
weather_interval = 1
# set frame gap between autofocus events
focus_interval = 1000000
# set frame gap between non-uniformity correction
nuc_interval = 25
# set camera buffer depth (higher more lag fewer dropped
frames)
num_buffers = 4
# set nominal framerate of camera internally
fps = 10
# set number of tries before giving up on GPS reception
max_gps_tries = 120
```

To control the logging:

1. Wait for computer to initialize
    a. The LCD screen should light up.
    b. The LEDs should at first be blank.
    c. The third LED should blink 3 times at a 3-second interval.
    d. All LEDs should blink in order from left to right at 1 second intervals. This blinking stops either when the GPS has acquired a lock or when 120 seconds have passed.
    e. All LEDs should blink from left to right at 0.1 second intervals once.
    f. The LCD screen will say either 'No thermal camera' or 'Paused'
    g. If the screen has no thermal camera, unplug computer, wait 2 seconds, and repeat.
    h. If otherwise, the system is probably ready to go. The rightmost LED should be solid blue.
2. Point the camera at the target and make sure the lens cap is removed.
3. Start data logging
    a. Press the right button on the computer for approximately one second. You will hear the camera focus itself. The rightmost LED will turn off.
    b. The second right most LED will blink at approximately 1 second intervals. Each blink indicates one data capture.
    c. The second through fourth LEDs should be lit up solid blue. These indicate that all sensors are working. The fourth LED may sometimes not be lit indicating that the GPS is not locked on to satellites.
    d. If only the second rightmost LED is solid, then the thermal camera did not get detected. Press both buttons simultaneously to turn off the computer safely. Then unplug the power cable and replug and repeat.
4. Stop data logging
    a. Press the right button on the computer for approximately one second. The rightmost LED will turn on and all other LEDS will turn off. The system is now paused.
5. To image another target once paused…
    a. Press the leftmost button. The LCD screen should increment in index by one on the file name.
    b. Press the right button to unpause and continue data logging.
6. To turn off the computer…
    a. Press both buttons at once. The LCDs will turn off and the system will shut down.
    b. Unplug the power cable.
    c. If the system does not shut down safely, pull out the power cable.
7. When else the computer will turn off…
    a. If the memory stick has < 100 MB free
    b. If the battery dies
    c. If the computer code has a bug (seems to be with refocusing)
    d. If the sensors fail or disconnect accidentally

To download data:

1. Unplug the USB stick.
2. The data will be stored in a folder named thermal_control_YYMMDD_HHMMSS. The HHMMSS data will be in GMT, so seven hours ahead of Colorado time. The start time will be the moment the camera got the GPS lock.
3. Verify that the –infrared.png images are focused.
4. Copy data to computer and erase from flash drive.

To connect to the Odroid and connect to a USB memory stick:

1. Turn on a laptop, plug in Ethernet cable between Odroid and laptop.
2. Configure IP interface manually, with address 169.254.100.5 (or 7, ... until one works), subnet 255.255.0.0
3. Turn on odroid
4. `ssh odroid@169.254.100.1`
5. `sudo ~/Desktop/control/usb_mount.sh`

To do any subsequent image processing or analysis, first:
Download all files from https://github.com/bblonder/thermal_analysis. This code includes a copy of npy-matlab (https://github.com/kwikteam/npy-matlab) which is needed to read numpy format matrices into MATLAB.

To collate and stabilize image frames into a radiometric array:

1. Put all thermal images in the same 'combined' folder along with all accessory (e.g. stats) files
2. Identify frames that are misfocused or have other artifacts and manually delete.
3. Run in MATLAB: `[cm mm aa stats] = align_thermal(mypath, stabilization_interval, frame_interval);`
   which will stitch together every `frame_interval` frames (e.g. 1) in order of filename with stabilization every `stabilization_interval` keyframes (e.g. 10) for all files within directory `mypath` (e.g. 'combined'). This process assumes that the input folder also contains weather statistics CSV files. You will need to comment this section of the code out if you are not using the same weather station as we were. Output is stored in MATLAB and not written to a file.

To create a calibrated temperature video:
The below code uses the above radiometric matrix and statistics information to transform camera data (pixel counts) to temperatures (in Kelvin). To do so a multi-step process is used: first the image frames are combined into a matrix; second, they are converted to temperatures using a temperature calibration subroutine (`calibrated_temperature` or `calibrated_temperature_simple`; see additional information in a below section); third (optional), the values in a region of interest are regressed against known values from ground-truth data for that region of interest, i.e. from thermocouples, with samples representing different time points, and the regression model is used to correct all other pixel values at all other times.

1. Run in MATLAB: `calibrate_thermal(image_array, stats, bound_temp_lo, bound_temp_hi, dogroundcalibration, xlsinputname, time_offset, file_visible_lores, outputname)` where `image_array` is the aa output variable from the previous section, stats is the stats variable from the previous section, `bound_temp_lo` is a minimum temperature (K) below which pixels will be clipped (e.g. 263), `bound_temp_hi` is a maximum temperature (K) above which pixels will be clipped (e.g. 333), `dogroundcalibration` is a binary flag determining whether additional temperature data (e.g. from thermocouples and separately measured) will be used to recalibrate the temperature (e.g. 1). If such a file exists, it should be given as argument `xlsinputname` in Excel format with the fourth column indicating hours, the fifth minutes, the sixth seconds, and the seventh the temperature of the ground-truth object (e.g. 'mytime.xls'). `timeoffset` is a value in seconds used to shift the ground-truth data relative to the thermal data (i.e. if the first thermal frame does not occur at the same time as the first ground-truth measurement) (e.g. 0). `file_visible_lores` is the path to a visible image of the scene, used to help identify the thermal image pixels corresponding to the ground-truth object (e.g. 'myimage.jpg'). `outputname` is a filename (ending in '.mat') where the calibrated temperature array and associated statistics will be stored (e.g. 'myoutput.mat').

To calibrate a single image frame:

We provide two programs that convert between FLIR radiometric output and temperature. Both output a matrix of the same size as the input, but in Kelvin instead of in counts. The first program `calibrated_temperature` provides full control over the calibration process, while `calibrated_temperature_simple` chooses default values for most parameters and can simplify usage for novice users. These subroutines do not need to be called directly as they are used by the above program, but are described here for investigators with other use-cases. Briefly, the subroutine first corrects the pixel counts for the sensor gain and offset, then solves an energy balance calculation involving other radiation fluxes (e.g. from attenuation from water in the atmosphere, radiation from the camera itself, and specular reflections from other sources) to isolate the radiation flux from the object, the inverts the Stefan-Boltzmann law for a graybody object to infer the object temperature.

1. `calibrated_temperature` is a function that takes several arguments:

Parameter	*Meaning*	*Suggested value*
`lPixval`	Input matrix	NA
`m_RelHum`	Relative humidity	0.5
`m_AtmTemp`	Atmospheric temperature (K)	293
`m_ObjectDistance`	Distance to object in scene (m)	1
`m_X`	Spectral response coefficient	1.9
`m_alpha1`	Spectral response coefficient	0.006569
`m_beta1`	Spectral response coefficient	-0.002276
`m_alpha2`	Spectral response coefficient	0.01262
`m_beta2`	Spectral response coefficient	-0.00667
`m_Emissivity`	Emissivity of object	0.97
`m_ExtOptTransm`	External optics transmittance	1
`m_AmbTemp`	Ambient temperature (K)	293
`m_ExtOptTemp`	External optics temperature (K)	293
`m_J0`	Offset coefficient	4214
`m_J1`	Gain coefficient	69.6244965
`m_R`	Blackbody coefficient	16671
`m_F`	Blackbody coefficient	1
`m_B`	Blackbody coefficient	1430.09998

2. `calibrated_temperature_simple` is a function that takes several arguments:

Parameter	*Meaning*	*Suggested value*
`counts`	Input matrix	NA
`temp_atm`	Atmospheric temperature (K)	293
`temp_reflected`	Reflected temperature (K)	293
`temp_external_optics`	Distance to object in scene (K)	293
`relative_humidity`	Relative humidity	0.5
`emissivity`	Emissivity of object	0.97
`distance_focal`	Distance to object in scene (m)	1

To extract temperature values from objects in a calibrated temperature array:
This program aligns a visible image to the calibrated thermal arrays created in the prior sections, enabling the investigator to select a region of interest in the visible image corresponding exactly to the same region of interest in the thermal array. The program then extracts the mean, standard deviation, 5%, 50%, and 95% quantile temperature value within the region of interest for each image frame, and produces a CSV output summarizing these time series paired with the underlying statistics logged from the weather sensors. The program then allows the investigator to repeat the extraction process for multiple objects in the aligned images. To run in MATLAB:

1. `analyze_image(file_thermal_mat, file_visible, folder_out)` where `file_thermal_mat` is an output of the `calibrate_thermal` program (e.g. `'mytemp.mat'`), `file_visible` is the path to a visible image with one or more objects of interest (e.g. `'myimage.jpg'`), and `folder_out` is a directory where the output file will be stored (e.g `'out'`).