bill\_thermal\_code\_1

Rosalee Elting

2/23/2022

**Check Working directory for your computer**

getwd()

## [1] "C:/Users/Mellisuga/Documents/R/bill\_thermal"

sessionInfo()

## R version 4.1.1 (2021-08-10)  
## Platform: x86\_64-w64-mingw32/x64 (64-bit)  
## Running under: Windows 10 x64 (build 19044)  
##   
## Matrix products: default  
##   
## locale:  
## [1] LC\_COLLATE=English\_United States.1252   
## [2] LC\_CTYPE=English\_United States.1252   
## [3] LC\_MONETARY=English\_United States.1252  
## [4] LC\_NUMERIC=C   
## [5] LC\_TIME=English\_United States.1252   
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## loaded via a namespace (and not attached):  
## [1] compiler\_4.1.1 magrittr\_2.0.1 fastmap\_1.1.0 tools\_4.1.1   
## [5] htmltools\_0.5.2 yaml\_2.2.1 stringi\_1.7.4 rmarkdown\_2.10   
## [9] knitr\_1.33 stringr\_1.4.0 xfun\_0.25 digest\_0.6.27   
## [13] rlang\_0.4.11 evaluate\_0.14

**Load Libraries**

library(Thermimage)

## Warning: package 'Thermimage' was built under R version 4.1.2

library(ggplot2)  
library(tidyr)

To use this Thermal Package, you will also have to download ExifTools here: <https://exiftool.org/install.html>

**Convert the FLIR jpeg to a regular one**

#convertflirJPG("IR\_1913.jpg", exiftoolpath = "installed" , headerindex=1)

library(Thermimage)  
f<-paste0(system.file("extdata/IR\_2412.jpg", package="Thermimage"))  
img<-readflirJPG(f, exiftoolpath="C:/")  
dim(img)

## [1] 480 640

cams<-flirsettings(f, exiftoolpath="C:/", camvals="")  
head(cbind(cams$Info), 20)

## [,1]   
## ExifToolVersionNumber 12.4   
## FileName 2412   
## Directory ":-4.1"   
## FileSize 638   
## FilePermissions "----"   
## FileType ""   
## FileTypeExtension ""   
## MIMEType ""   
## JFIFVersion 1.01   
## ExifByteOrder "-"   
## Make ""   
## CameraModelName 660   
## Orientation ""   
## XResolution 72   
## YResolution 72   
## ResolutionUnit ""   
## Software "1.1.98"  
## YCbCrPositioning ""   
## ExposureTime 133   
## ExifVersion 220

plancks<-flirsettings(f, exiftoolpath="C:/", camvals="-\*Planck\*")  
unlist(plancks$Info)

## PlanckR1 PlanckB PlanckF PlanckO PlanckR2   
## 2.110677e+04 1.501000e+03 1.000000e+00 -7.340000e+03 1.254526e-02

cbind(unlist(cams$Dates))

## [,1]   
## FileModificationDateTime "2022-02-23 16:42:09"  
## FileAccessDateTime "2022-02-28 17:41:33"  
## FileCreationDateTime "2022-02-23 16:42:09"  
## ModifyDate "2013-05-09 13:22:23"  
## CreateDate "2013-05-09 13:22:23"  
## DateTimeOriginal "2013-05-09 19:22:23"

ObjectEmissivity<- cams$Info$Emissivity # Image Saved Emissivity - should be ~0.95 or 0.96  
dateOriginal<-cams$Dates$DateTimeOriginal # Original date/time extracted from file  
dateModif<- cams$Dates$FileModificationDateTime # Modification date/time extracted from file  
PlanckR1<- cams$Info$PlanckR1 # Planck R1 constant for camera   
PlanckB<- cams$Info$PlanckB # Planck B constant for camera   
PlanckF<- cams$Info$PlanckF # Planck F constant for camera  
PlanckO<- cams$Info$PlanckO # Planck O constant for camera  
PlanckR2<- cams$Info$PlanckR2 # Planck R2 constant for camera  
ATA1<- cams$Info$AtmosphericTransAlpha1 # Atmospheric Transmittance Alpha 1  
ATA2<- cams$Info$AtmosphericTransAlpha2 # Atmospheric Transmittance Alpha 2  
ATB1<- cams$Info$AtmosphericTransBeta1 # Atmospheric Transmittance Beta 1  
ATB2<- cams$Info$AtmosphericTransBeta2 # Atmospheric Transmittance Beta 2  
ATX<- cams$Info$AtmosphericTransX # Atmospheric Transmittance X  
OD<- cams$Info$ObjectDistance # object distance in metres  
FD<- cams$Info$FocusDistance # focus distance in metres  
ReflT<- cams$Info$ReflectedApparentTemperature # Reflected apparent temperature  
AtmosT<- cams$Info$AtmosphericTemperature # Atmospheric temperature  
IRWinT<- cams$Info$IRWindowTemperature # IR Window Temperature  
IRWinTran<- cams$Info$IRWindowTransmission # IR Window transparency  
RH<- cams$Info$RelativeHumidity # Relative Humidity  
h<- cams$Info$RawThermalImageHeight # sensor height (i.e. image height)  
w<- cams$Info$RawThermalImageWidth # sensor width (i.e. image width)

Converting Raw binary to thermal data If stored with a TIFF header, the data load in as a pre-allocated matrix of the same dimensions of the thermal image, but the values are integers values, in this case ~18000. The data are stored as in binary/raw format at 2^16 bits of resolution = 65536 possible values, starting at 0. These are not temperature values. They are, in fact, radiance values or absorbed infrared energy values in arbitrary units. That is what the calibration constants are for. The conversion to temperature is a complicated algorithm, incorporating Plank’s law and the Stephan Boltzmann relationship, as well as atmospheric absorption, camera IR absorption, emissivity and distance to namea few. Each of these raw/binary values can be converted to temperature, using the raw2temp function:

str(img)

## int [1:480, 1:640] 18090 18074 18064 18061 18081 18057 18092 18079 18071 18071 ...

Can add many parameters in Raw2Temp, need to look into that. Looks like with sample data, just the plain Raw2Temp command brings up correct values to within 0.1 degress.

temperature<-raw2temp(img)  
str(temperature)

## num [1:480, 1:640] 23.6 23.5 23.4 23.4 23.5 ...

Will need to change height and width of following code for image. Get from flirsettings command (xresolution, yresolution)

library(fields)

## Warning: package 'fields' was built under R version 4.1.2

## Loading required package: spam

## Warning: package 'spam' was built under R version 4.1.2

## Spam version 2.8-0 (2022-01-05) is loaded.  
## Type 'help( Spam)' or 'demo( spam)' for a short introduction   
## and overview of this package.  
## Help for individual functions is also obtained by adding the  
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##   
## Attaching package: 'spam'

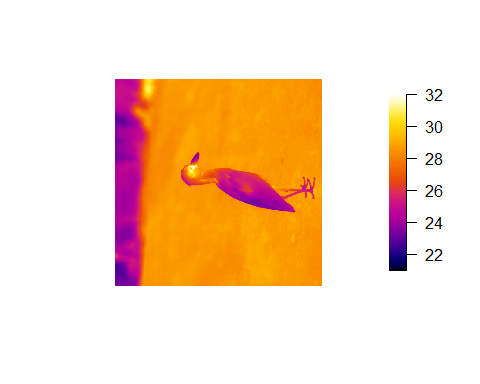
## The following objects are masked from 'package:base':  
##   
## backsolve, forwardsolve

## Loading required package: viridis

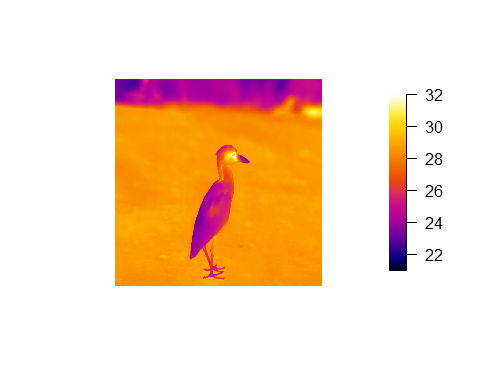
## Loading required package: viridisLite

##   
## Try help(fields) to get started.

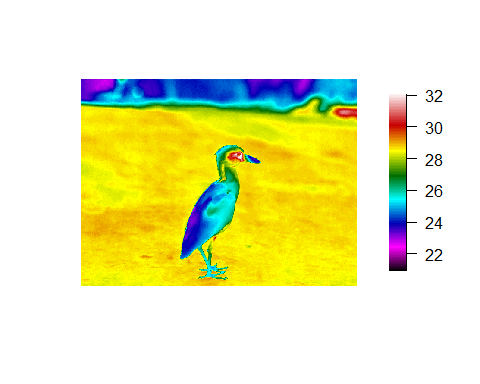
plotTherm(temperature, h=72, w=72, minrangeset=21, maxrangeset=32)



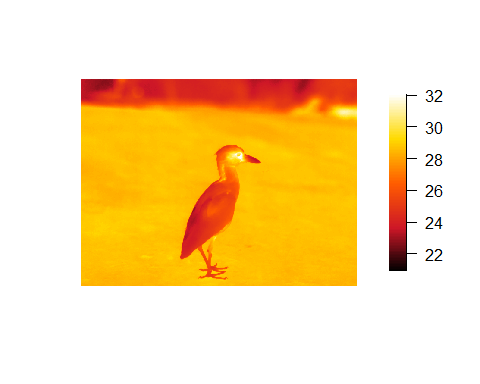
#if rotated, can manipulate with the trans= argument in plotTherm  
plotTherm(temperature, w=72, h=72, minrangeset = 21, maxrangeset = 32, trans="rotate270.matrix")

 Other palette options

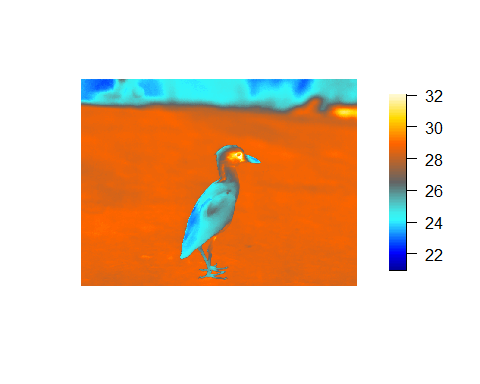
plotTherm(temperature, w=w, h=h, minrangeset = 21, maxrangeset = 32, trans="rotate270.matrix",   
 thermal.palette=rainbowpal)



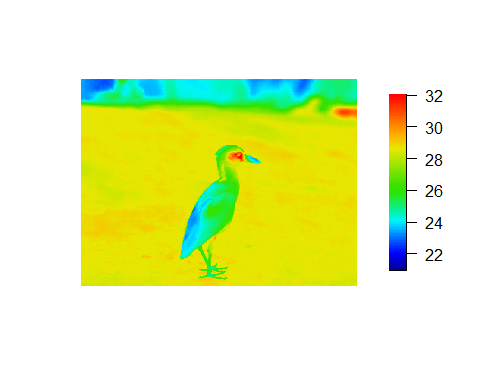
plotTherm(temperature, w=w, h=h, minrangeset = 21, maxrangeset = 32, trans="rotate270.matrix",   
 thermal.palette=glowbowpal)



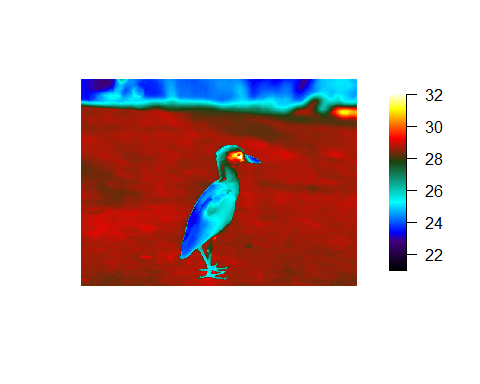
plotTherm(temperature, w=w, h=h, minrangeset = 21, maxrangeset = 32, trans="rotate270.matrix",   
 thermal.palette=midgreypal)



plotTherm(temperature, w=w, h=h, minrangeset = 21, maxrangeset = 32, trans="rotate270.matrix",   
 thermal.palette=midgreenpal)



plotTherm(temperature, w=w, h=h, minrangeset = 21, maxrangeset = 32, trans="rotate270.matrix",   
 thermal.palette=rainbow1234pal)



#there are many more

Deconvolute temperature to raw and back to temperature: With thermal imaging analysis, there are at least 7 environmental parameters that must be known to convert raw to temperature. Sometimes, the parameters might have been incorrectly input by the user or changing the parameters is too cumbersome in the commercial software. temp2raw() is the inverse of raw2temp(), which allows you to convert an estimated temperature back to the raw values (i.e. deconvolute), using the initial object parameters used.

#For example, convert a temperature estimated at 23 degrees C, under the default blackbody conditions:  
temp2raw(23, E=1, OD=0, RTemp=20, ATemp=20, IRWTemp=20, IRT=1, RH=50, PR1=21106.77, PB=1501, PF=1, PO=-7340, PR2=0.012545258)

## [1] 17994.06

Which yields a raw value of 17994.06 (using the calibration constants above). Now you can use raw2temp to calculate a better estimate of an object that has emissivity=0.95, distance=1m, window transmission=0.96, all temperatures=20C, 50 RH Note: the default calibration constants for my FLIR camera will be used if you leave out the calibration data during this two step process, but it is more appropriate to look up your camera’s calibrations constants using the flirsettings() function.

raw2temp(17994.06, E=0.95, OD=1, RTemp=20, ATemp=20, IRWTemp=20, IRT=0.96, RH=50, PR1=21106.77, PB=1501, PF=1, PO=-7340, PR2=0.012545258)

## [1] 23.31223

Finding a way to quantitatively analyse thermal images in R is a challenge due to limited interactions with the graphics environment. Thermimage has a function that allows you to write the image data to a file format that can be easily imported into ImageJ.

First, the image matrix needs to be transposed (t) to swap the row vs. column order in which the data are stored, then the temperatures need to be transformed to a vector, a requirement of the writeBin function. The function writeFlirBin is a wrapper for writeBin, and uses information on image width, height, frame number and image interval (the latter two are included for thermal video saves) but are kept for simplicity to contruct a filename that incorporates image information required when importing to ImageJ:

#writeFlirBin(as.vector(t(temperature)), templookup=NULL, w=w, h=h, I="", rootname="Uploads/FLIRjpg")

See this link: <https://github.com/gtatters/Thermimage#import-raw-file-into-imagej> For converting to Image J, processing a video, or bulk processing many photos for ImageJ analysis.