Raspberry Pi High Speed Camera Software

Kim Miikki and Alp Karakoç

Version – 21.9.2020

Table of Contents

1 List of Abbreviations and Symbols

$	Ordinary User in BASH Shell
µs	microsecond = 10^{-6} s
ms	millisecond = 10^{-3} s
2K	2560x1400 resolution
720p	1280x720 resolution
A/R	Aspect Ratio
BW	Black and White (eg. Gray Scale)
CRF	Constant Rate Factor
CR2	Raw Image Format used by Canon Digital Cameras
CTRL	Control Key
FFmpeg	A complete, cross-platform solution to record, convert and stream audio and video
FHD	1980x1080, Full HD
FOV	Field Of View
FPS	Frames per Second
gparted	GNOME Partition Editor
H.264	Advanced Video Coding
HQ	High Quality (refers to the Raspberry Pi HQ camera module)
IR	Infrared
JPG	Lossy Compressed Image Format
Md	Raspberry Pi video mode
mkvmerge	Merge multimedia streams into a Matroska$^{(tm)}$ file
MKV	Matroska Multimedia Container (free and open-standard format)
MP	Mega Pixels
NoIR	No Infrared Filter
PIL	Python Imaging Library
PNG	Lossless Compressed Image Format
raspistill	Program for capturing still photographs with the camera module
raspivid	Program for capturing video with the camera module

RGB	Red, Green and Blue Color Channels
ROI	Region of Interest
RPI	Raspberry Pi
VGA	640x480 resolution
VIS	Visible light

2 Introduction

Raspberry Pi combined with a camera module is a portable and cheap imaging system which. The camera (version 2.x) has an 8 MP sensor with a 1 µs minimum exposure time. There are currently two different camera modules which the first is equipped with an IR filter and the second (NoIR) without it. Both modules provides 200 FPS high speed video recording at VGA resolution and 90 FPS at 720p resolution. A new HQ camera module has also been released on April of 2020. All of the camera applications has been updated to support the new camera module.

A camera module connected to a RPI can be used in many different ways. This is due to the programmability of the system. There are four applications (raspistill, raspivid, raspiyuv and raspividyuv) which can be used without programming new applications. Unfortunately they have a lot of options and therefore with script generation software it is possible to make their use much easier. Some of the Python application described in this manual work in this way. An other way is to use RPI camera libraries and create applications with their functions.

All of these programs (Python scripts) are licensed under MIT license.

3 Prerequisites

3.1 Hardware

All programs have been tested to work with the following hardware:

Raspberry Pi 4 Model B 4 GB
Raspberry Pi CMOS CAMERA MODULE NoIR V2 8MPx or Raspberry Pi CMOS CAMERA MODULE V2 8MPx
Raspberry Pi HQ Camera 12.3 MP camera module
16 GB Raspberry Pi NOOBS (New Out of Box Software)
USB 3 SSD drive for storage of images or videos
Flicker free light source(s) if exposure time is under 10-20 ms

Some of the programs have been developed with a Raspberry Pi 3 Model B, but the newest versions of the applications have not been tested to work in that environment.

3.2 Operating System and Programs

The system is developed and tested to work with *Raspbian GNU/Linux 10 (buster)*. Following steps (software installation etc.) have to be done in order to run applications which are described in this manual:

Configure a SSD Drive

$ lsblk → /sda

Start gparted
Remove all partitions from the SSD drive
Device → create partition table
    msdos <Apply>
Partition > New
    File system: ext4
    Label: ssd

$ sudo tune2fs -m 0 /dev/sda1

Create /opt/tools directory and add it into the PATH variable
$ sudo mkdir /opt/tools
Copy RPI camera applications to /opt/tools directory
$ sudo chmod ugo+rx *.py

$ sudo nano /etc/profile.d/tools.sh
```
export PATH="$PATH:/opt/tools"
```

$ reboot

Create desktop shortcuts
campreview.py
vidpreview.py
Documents -> /opt/tools/documentation

Raspberry Pi Configuration (Preferences > RPC)
System > Auto login: [v] Login as user 'Pi'
Interfaces
    Camera (*) Enable
    SSH (*) Enable
    SPI (*) Enable
    I2C (*) Enable
Performance
    GPU Memory:76 → 512

$ reboot

Run the next command in order to test that camera module is working normally:
$ raspistill -t 0
This will show camera live preview forever (exit by CTRL+C).

More comprehensive test can be done with the following command:
$ raspistill -v -o test.png

Update RPI Firmware

$ sudo apt rpi-update
$ reboot

Additional programs
$ sudo apt install mkvtoolnix* exif exifprobe exiftags xscreensaver gparted gnome-disk-utility libraw-dev xclip xsel mediainfo ipython3 filezilla gnome-screenshot dos2unix

Prerequisites for gtlvideo.py
$ sudo apt install libraw-dev
$ sudo apt install ffmpeg

Gimp
$ sudo apt install gimp gimp-plugin* gimp-data-extras gimp-lensfun gimp-g* gimp-help-common gimp-help-en gimp-resynthesizer gimp-texturize

Development libraries for opencv-python
sudo apt install libgtk2.0-dev pkg-config

3.3 Python and Required Libraries

In order to run RPI camera applications Python version must be at least 3.5. The version is checked at least by gtlvideo.py.

These steps are required to be done before running the RPI camera applications:

```
sudo pip3 install --upgrade pip
sudo pip3 install path.py
sudo pip3 install rawpy
sudo pip3 install Pillow
sudo pip3 install imageio
sudo pip3 install exifread
sudo pip3 install numpy pandas
sudo pip3 install pyperclip
sudo pip3 install dlib face_recognition imutils
sudo pip3 install pysimplegui
sudo pip3 install Cython
sudo pip3 install scikit-build
```

In addition, the following applications must be installed:

```
sudo apt install opencv*
sudo apt install libatlas-base-dev
sudo apt install libjasper-dev
sudo apt install libqtgui4
sudo apt install python3-pyqt5
sudo apt install libqt4-test
sudo apt install python3-opencv
sudo apt install build-essential cmake pkg-config
sudo apt install libatlas3-base libgfortran5
```

Version – 21.9.2020

$ sudo pip3 install numpy==1.19.1 --extra-index-url https://www.piwheels.org/simple

$ sudo pip3 install matplotlib==3.3.1 --extra-index-url https://www.piwheels.org/simple

$ sudo pip3 install --no-binary :all: opencv-python
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting opencv-python
  Using cached opencv-python-4.4.0.42.tar.gz (88.9 MB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.19.1)
Building wheels for collected packages: opencv-python
  Building wheel for opencv-python (PEP 517) ... done
  Created wheel for opencv-python: filename=opencv_python-4.4.0.42-cp37-cp37m-linux_armv7l.whl size=10294672 sha256=1af48e51176c1ae154d2939686ec27f04e14f7fe53381541c46f82d576fd5eff
  Stored in directory: /root/.cache/pip/wheels/87/89/42/0452b33622ef8f0fd9b90a8f199a7ab909e914e1432ea17b06
Successfully built opencv-python
Installing collected packages: opencv-python
Successfully installed opencv-python-4.4.0.42
Build time ~2 h

sudo pip3 install scipy
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting scipy
  Using cached scipy-1.5.2.tar.gz (25.4 MB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Building wheels for collected packages: scipy
  Building wheel for scipy (PEP 517) ... done
  Created wheel for scipy: filename=scipy-1.5.2-cp37-cp37m-linux_armv7l.whl size=50722066 sha256=58285d415bd0469e304a76154822db4eae20a0a58463a1241bda8c5204a6b68c
  Stored in directory: /root/.cache/pip/wheels/3c/79/d0/bc51e2a8870685fb1ae515c967d15b7a89dfd530a99717f1fc
Successfully built scipy
Installing collected packages: scipy
Successfully installed scipy-1.5.2
Build time <1 h

Version – 21.9.2020

Pyhton IDE
$ sudo apt install spyder3

Changing pyhton on pip versions by using alias
$ nano .bash_aliases
alias python=python3
alias pip=pip3

$ reboot

3.4 RPI High Speed Camera Software

The RPI High Speed Camera Software will be available from GitHUB in the summer of 2020. Meanwhile they are only available from the author in Aalto University. Until they are released, the applications may only be used at Aalto University.

Some of these applications can be executed on any operating system if Python 3.5 is installed. Also some open source programs must be present, especially if Python application generates scripts for these programs.

List of RPI camera module applications

awb_gains.py

- Get auto white balance gains for red and blue channels

bracket-exposure.py

- Determine an optimal exposure time by capturing images with different exposures

bw-rgbsplit

- Split RGB channels of pictures into separate images (1 channel gray scale)

bw-rgbcombine

- Combine one channel R, G, B pictures into RGB pictures

calibratecam.py

- Calibrate camera module white balance red and blue gains

camapps.py

- High speed camera software suite application launcher

camera_model.py

- Get camera information

campreview.py

- Camera preview for composition

capturepics.py

- Capture pictures manually without preview

capturepics-preview.py

- Capture pictures manually with preview

dminexp.py

- Detect minimum exposure time

expss.py

- Get auto exposure shutter speed

fpsvideo.py

- Record videos in slow, normal or high speed

gen_tlfiles.py

- Copy and rename pictures to time-lapse files

gtlvideo.py

- Generate videos from pictures

optimum_gains.py

- Get optimal red and blue gains from a Calibration File

pics2png.py

- Convert image formats which PIL supports to png format

rgbinfo.py

- Calculate RGB and BW means of pictures

rgbsplit.py

- Split RGB channels of pictures into separate images (3 color channels)

rgbcombine.py

- Combine R, G, B channel pictures into RGB pictures

timelapse.py

- Capture time-lapse pictures

vidpreview.py

- Video preview with ROI for composition

4 Launcher for RPI Camera Applications

camapps.py

- High speed camera software suite application launcher
- RPI is not required

All of applications which are described in this manual, can be launched from this program. Different tasks are organized under following categories: Composition, Exposure, White Balance Calibration, Images and Videos, Image Analysis and Operations. Their tasks are listed in the following example:

```
$ camapps.py
RPI High Speed Camera Software Suite (C) Kim Miikki 2020

Composition
 1 Camera info
 2 Camera preview
 3 Video preview

Exposure
 4 Auto exposure shutter speed
 5 Exposure bracketing
 6 Detect minimum exposure time

White Balance Calibration
 7 Auto white balance (AWB) gains
 8 Optimal gains from a calibration file
 9 Calibration

Images and Videos
10 Camera
11 Camera without preview
12 Time-lapse
13 Video recorder

Image Analysis and Operations
14 Create videos from images
15 Color analysis
16 Split images to RGB channels
17 Combine RGB channels to color images
18 Split RGB channels to gray scale
19 Combine RGB gray scale channels to color images
20 Convert images to PNG format
21 Images to time-lapse files
 0 Exit

Select task (0...21, Default=0: <Enter>):
```

5 Raspberry Pi Camera Module Information

camera_model.py

- Get Raspberry Pi camera module information
- RPI camera module is required

This application gives information about attached camera module: sensor model, megapixels and camera resolution. Here is an example of the HQ camera module:

```
$ camera_model.py
Raspberry Pi camera module
  revision:    imx477
  model:       RP_imx477
  make:        RaspberryPi
  megapixels:  12.3
  max x:       4056
  max y:       3040
```

6 Camera Preview for Composition

camerapreview.py

- Camera Preview for Composition
- Requires a RPI and a camera module
- Uses PiCamera library

The first step in order to capture pictures or videos is to preview the composition. An application designed for this task is named **campreview.py**. Although same task can be done with the following command: raspistill -t 0, campreview.py does little more. From the current screen resolution it determines if there are two preview choices. Those are demonstrated by the following two pictures:

Full preview composition with margins.

Full screen and cropped preview composition without margins.

One can find in the source code following variables: maxx_tested and maxy_tested. They describe maximum preview window resolution which gives in full screen preview mode full composition. Beyond these values full screen preview will give a cropped composition. Unfortunately the values are not the same for every resolution. In the code the maximum tested preview size 1615x1200. So full composition in full screen will be always available for the resolution of 1280x1024 (which is available on RPI 4 Model B). FHD resolution will give margins if full composition is desired.

Caution! The application must be executed in terminal. Otherwise it is very difficult to exit the program.

Here is an example how to run the application in FHD mode:

```
Raspberry Pi camera module live preview
Full screen and cropped preview (Y/N, default: N <ENTER>)
Default selected
Preview:      ENTER
Exit program: ESC
```

Preview will start by pressing Enter and end by pressing ESC.

7 Video Preview with ROI for Composition

vidpreview.py

- Video Preview for Composition
- Requires a RPI and a camera module
- Calculates ROIs for all supported video modes
- Uses PiCamera library

RPI camera module (ver. 2.x) have a set of video modes that can be used when recording videos. The following table summarizes which modes are available, their resolutions, regions of interests etc:

Md	Video mode	A/R	FPS	FOV	Binning	Sensor area	ROI X0	ROI Y0	ROI W	ROI H
1	1920x1080	16:9	30	Partial	None	1920x1080	680	692	1920	1080
2	1920x1080	16:9	15	Full	None	3280x1845	0	310	3280	1845
3	1920x1080	16:9	15	Full	None	3280x1845	0	310	3280	1845
4	1640x1232	4:3	40	Full	2x2	3280x2464	0	0	3280	2464
5	1640x922	16:9	40	Full	2x2	3280x1845	0	310	3280	1845
6	1280x720	16:9	90	Partial	2x2	2560x1440	360	512	2560	1440
7	640x480	4:3	200	Partial	2x2	1280x960	1000	752	1280	960

RPI camera module 2.x video modes

The table of video modes shows that the only mode, which gives full field of view (= maximum sensor area), is mode number 4. Modes 2-5 gives full FOV, but in modes 2, 3 and 6 not whole sensor area is used for capturing videos. This is due to different A/R from the sensor to video resolution. Mode 1 captures videos in FHD mode and modes 6 and 7 are high speed recording modes (up to 90 or 200 FPS). Binned 2x2 modes have better sensitivity than non binned mode, because the sensor data per pixel is collected from a 2x2 square of sensor pixels.

This application only asks for a desired video mode (1-8). Normalized ROI values (0...1) are calculated and a live preview of the selected video mode is enabled, until ESC is pressed.

The HQ camera has fever video modes which are shown in the following table. All of the documented resolutions [1] did not work on 3.6.2020. Hence a manual testing was required in order to get working video resolutions.

Md	Video mode	A/R	FPS	FOV	Binning
1	2028x1080	16:9	50	Partial	2x2
2	2028x1080	16:9	50	Full	2x2
3	1600x1200	16:9	10	Full	None
4	1012x760	4:3	120	Full	4x4

RPI HQ camera module manually tested video modes

Here is an example how to run this application:

```
$ vidpreview.py
Raspberry Pi camera v. 2.x video preview

Video modes
Md Video      A/R  W    H     ROI W  ROI H
1  1920x1080 16:9 1920 1080  1920   1080
2  1920x1080 16:9 1920 1080  3280   1845
3  1920x1080 16:9 1920 1080  3280   1845
4  1640x1232  4:3 1640 1232  3280   2464
5   1640x922 16:9 1640  922  3280   1845
6   1280x720 16:9 1280  720  2560   1440
7    640x480  4:3  640  480  1280    960

Preview:      Md
Exit program: ESC
```

Key 6 pressed

```
Md:  6
ROI: 0.109756,0.207792,0.780488,0.584416
```

ESC key terminates the video preview and the program

Cited references:

[1] https://www.raspberrypi.org/documentation/raspbian/applications/camera.md, 3.6.2020

8 Auto Exposure Shutter Speed

expss.py

- Get auto exposure shutter speed
- Optionally get red and blue gains
- Requires a RPI and a camera module

The second step before starting to capture images or recording videos is to determine what is the optimum exposure time. This script returns the actual shutter speed in auto exposure mode. Also it is possible to get the red and blue gains by enabling gains in the code (enable_gains=True).

Shutter speed is returned in ms and µs units, as shown here:

```
$ expss.py
Auto exposure shutter speed for Raspberry Pi camera 2.x

Shutter speed:    8 ms
Shutter speed: 7533 µs
```

9 Exposure Bracketing

bracket-exposure.py

- Determine an optimal exposure time by capturing images with different exposures
- 26 JPG files will be created from exposures between 1 µs to 0.33 s
- Requires a RPI and a camera module
- Requires raspistill
- Generates and executes scripts (capture commands) which can be executed without Python

A better way to determine optimum exposure time is to do exposure bracketing. The **bracket-exposure.py** captures a series of pictures with different exposure times, starting from 1 µs and ending to 0.33 s. From these pictures, it is possible quickly to determine what is the optimum exposure for a imaging project.

Here is an example how bracketing exposures works:

1 µs

5 µs

10 µs

25 µs

50 µs

100 µs

200 µs

350 µs

500 µs

750 µs

1 ms

2 ms

3.5 ms

5 ms

7.5 ms

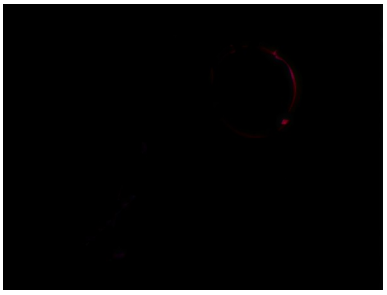
10 ms

15 ms

20 ms

35 ms

50 ms



100 ms (too much light for sensor)

The rest exposure times were 150 ms, 200 ms, 250 ms, 300 ms and 330 ms. The camera sensor can not handle so much light. Therefore, these images are almost black (not shown here).

In this example, the composition was illuminated by two non flickering high power LED lamps (550 lx and 440 lx) and with ambient light sources (also non flickering).

The usage of this application is very simple which is shown here:

```
$ bracket-exposure.py
Exposure bracketing program for Raspberry Pi camera v. 2.x (c) Kim
Miikki 2019

Select image quality (1...100; default: 90):
Default value selected: 90

List disk and partitions:
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda           8:0    0 223.6G  0 disk
└─sda1        8:1    0 223.6G  0 part /media/pi/data
mmcblk0     179:0    0  14.8G  0 disk
├─mmcblk0p1 179:1    0   2.2G  0 part
├─mmcblk0p2 179:2    0     1K  0 part
├─mmcblk0p5 179:5    0    32M  0 part
├─mmcblk0p6 179:6    0   256M  0 part /boot
└─mmcblk0p7 179:7    0  12.3G  0 part /

Path to images (current directory: <Enter>):
Exposure bracketing name: test
Select ISO (100, 200, ... 800): 100
...
```

10 Detect Minimum Exposure Time

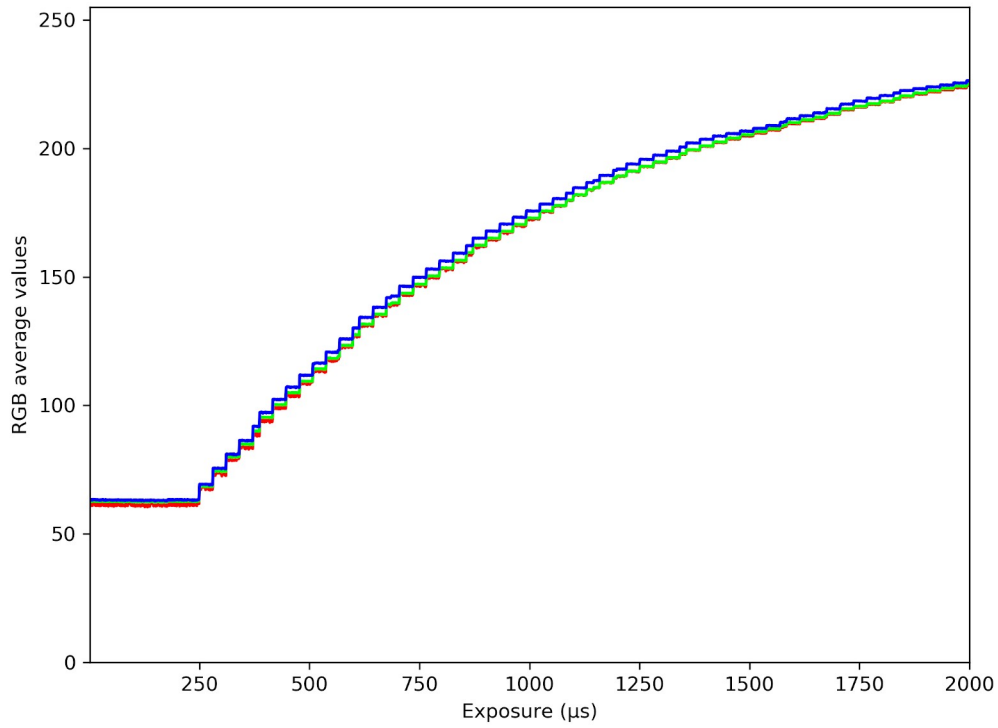
dminexp.py

- Detect the minimum exposure time for a RPI camera module
- Requires a RPI and a camera module
- Uses PiCamera library

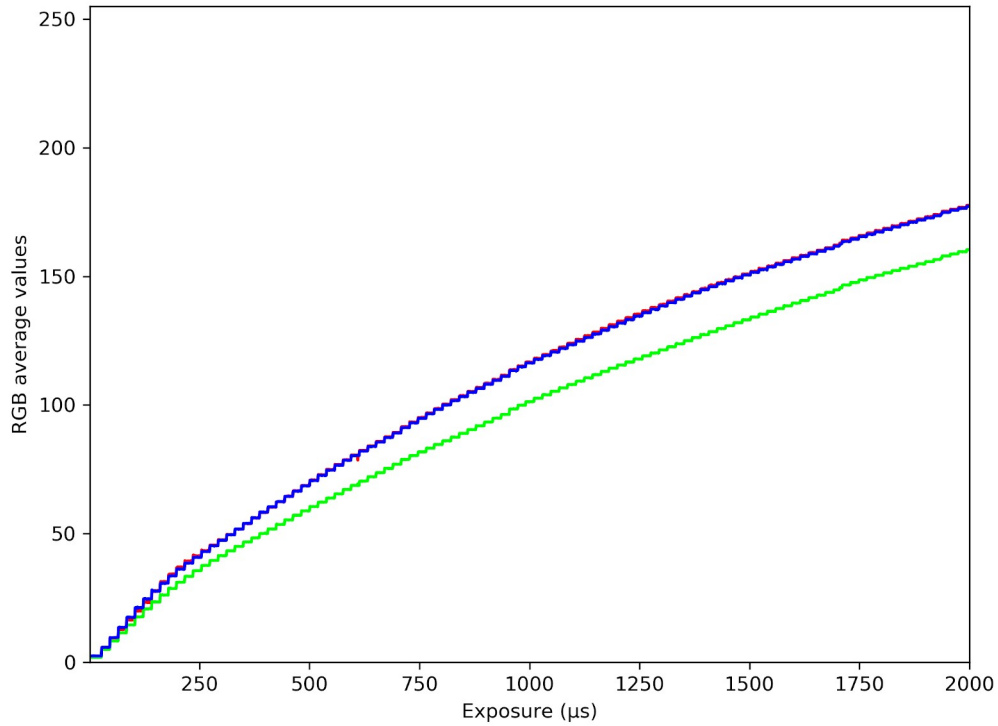
This application captures initially 2000 pictures from 1 µs to 2000 µs by increasing exposure time with 1 µ s for each picture. Consistent results can be obtained by using red and blue gains instead of auto white balance. The images are not saved as default.

As a result a RGB analysis file is created and optionally each images are saved in 320x240 resolution. For an immediate analysis a 300 dpi matplotlib figure is also created, where each color channels (RGB) are represented as functions of exposure time in µs scale.

Some experiments were conducted on 24.6.2020 in order to determine the minimum exposure time for two different camera modules: version 2 (8 MP) and HQ (12.3 MP). Here are the results of these experiments:



HQ camera module RGB response when red gain=3.16 and blue gain=1.24



Camera module (version 2) RGB response in AWB mode

Fromn these pictures can be detected that HQ camera sensor starts to give respons at 250 µs and camera module version 2 almost immediately after 1 µs.

Here an output of a test run of this application:

```
$ dminexp.py
Raspberry Pi Detect Minimum Exposure Time

List disk and partitions:
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda           8:0    0 223.6G  0 disk
└─sda1        8:1    0 223.6G  0 part /media/pi/data
sdb           8:16   1   7.5G  0 disk
└─sdb1        8:17   1   7.5G  0 part /media/pi/INTENSO
mmcblk0     179:0    0  28.8G  0 disk
├─mmcblk0p1 179:1    0   2.4G  0 part
├─mmcblk0p2 179:2    0     1K  0 part
├─mmcblk0p5 179:5    0    32M  0 part
├─mmcblk0p6 179:6    0   256M  0 part /boot
└─mmcblk0p7 179:7    0  26.2G  0 part /

Current directory:
/media/pi/INTENSO/20200624-rpi-ouput

Path to images (current directory: <Enter>):
Project name (default=exp: <Enter>):
Select ISO (100, 200, 320, 400, 500, 640, 800; Default=100):
Default value selected: 100

AWB mode on (Y/N, Default y: <Enter>):
Default selected: AWB enabled

Save captured images (Y/N, Default n: <Enter>):
Default selected: Save images disabled

Capturing images:
1 [3.01457031 2.02445312 3.06352865]
2 [2.98997396 2.02684896 3.04075521]
3 [2.60704427 1.75122396 2.54891927]
4 [2.57395833 1.77282552 2.52416667]
5 [2.5983724  1.75949219 2.53858073]
6 [2.99708333 2.02283854 3.06354167]
7 [2.99395833 2.01251302 3.08651042]
8 [3.00848958 2.00389323 3.07411458]
9 [2.99511719 2.01322917 3.06808594]
10 [2.62606771 1.70393229 2.62039063]
...
```

11 Auto White Balance Gains

awb_gains:

- Get red and blue gains
- Requires a RPI and a camera module

Consistent images can be captured when exposure times, white balance and gains are fixed. For frame rates over 120 fps (8.3 ms shutter speed), automatic exposure and gain control must be turned of [1]. However, the auto white balance can be used even with faster shutter speeds in order to determine red and blue gains. This method is used by **awb_gains.py**. As default the script uses auto exposure mode, but it can be turned off in the code (exp_mode_manual=”on”). Here is an example how to get the gains:

```
$ awb_gains.py
Auto white balance gains for Raspberry Pi camera 2.x

Red gain:  1.89 121/64
Blue gain: 1.55 397/256
```

A more accurate method to determine red and blue gains is described in next chapter.

Cited references:

[1] https://www.raspberrypi.org/documentation/raspbian/applications/camera.md, 1.4.2020

12 White Balance Calibration

calibratecam.py

- Calibrate camera module white balance red and blue gains
- Requires a RPI and a camera module
- Requires raspistill

In order to capture consistent pictures or videos AWB has be turned off. In manual white balance mode red and blue gains must be provided. A fast way to do this is described in previous chapter, but **calibratecam.py** will loop trough all combinations red and blue gains within their ranges, create calibration figures and calculate the optimum gains.

As the first automation step of the white balance calibration, the color data is obtained from three separate (R,G,B) channels in a range of [0, 255] for the investigated gain combinations and thereafter the average values of the data are calculated. Following this step, the absolute differences of all possible 2-tuples (RB, RG, GB) of the averaged data are summed up for different gain combinations. Eventually, the gain combination resulting in the minimum absolute difference is taken as the optimum gain.

Version – 21.9.2020

This application has predefined ranges for red and blue gains 1.0 - 2.0 by step 0.1. The ranges can be overridden with custom steps. As default 11 x 11 directories will be created which results in 121 combination of red and blue gains. A series of pictures are then captured with different exposure times (user selected exposure range) in each directory. RGB color analysis is performed when all series are captured. Color analysis results are stored as calibration figures and in a text file. The later one is used for calculating optimum red and blue gains.

12.1 Calibration procedure

This section describes how to do the calibration for red and blue gains. The calibration method is based on white or gray reference color (white or gray card). The goal is to get red, green and blue average sensor response curves as close as possible each other. Only red and blue gains is adjustable. By increasing red or blue gains will decrease the response of the other channels. Hence the calibration becomes an optimizing task. The second requirement for fast shutter speeds (<10-20 ms) is to use flicker free light source(s).

The rough optimum shutter speed has to be determined when the composition with the illumination is ready. This is important when selecting a predefined ranges of exposure times. To get auto exposure shutter speed, expss.py can be executed.

In order to select proper ranges for calibration, awb_gains reads and returns AWB gain values. These are only rough values for read and blue gains. The calibration will give more accurate, if the optimization task is done in proper ranges and with small steps.

These two steps are recommended to execute before calibration:

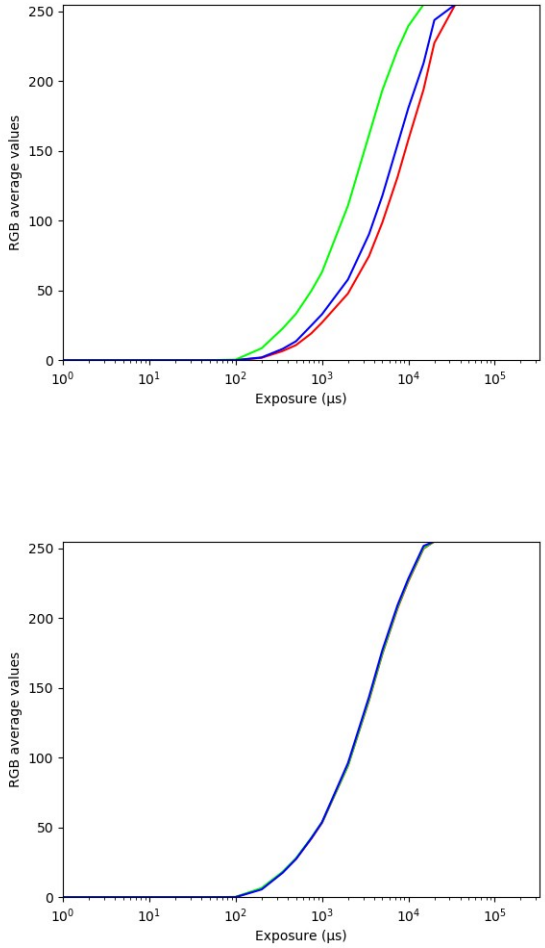
```
$ expss.py
Auto exposure shutter speed for Raspberry Pi camera 2.x

Shutter speed:    2 ms
Shutter speed: 2410 µs

$ awb_gains.py
Auto white balance gains for Raspberry Pi camera 2.x

Red gain:  1.85 237/128
Blue gain: 1.51 387/256
```

Next step is to execute the calibration application:

```
$ calibratecam.py
Calibrate Raspberry Pi camera v. 2.x fast (c) Kim Miikki and Alp
Karakoc 2020

Calibration from existing files (Y/N, Default N: <Enter>):
Default selected: Analyze only mode disabled
Select quality (1...100, Default=90: <Enter>):
Default value selected: 90

List disk and partitions:
...

Current directory:
/media/pi/data/20200403-calibration_for_manual
```

```
Path to images (current directory: <Enter>):
Calibration name: manfrotto-white-2xled8-step0.2
Select ISO (100, 200, 320, 400, 500, 640, 800; Default=100):
Default value selected: 100

Calculating auto exposure range for calibration:
Auto shutter speed:    3 ms
Auto shutter speed: 2524 µs
Auto shutter speed range for calibration (µs):
[616, 1233, 1850, 2467, 3083, 3700, 4317]

Calibration modes
Md  Pictures  EXP min  EXP max
1          6       50     1000 µs
2          6     1000    15000 µs
3          6    10000    60000 µs
4          6    20000   300000 µs
5         12       50   200000 µs
6         26        1   330000 µs
7          7      616     4317 µs
Select exposure range mode (1...7): 7
```

Calibration modes 1-4 captures only 6 pictures per series. With default gain ranges and steps the calibration is ready under 10 min by using a Raspberry Pi 4 Model B 4 GB with a USB3 SSD-drive. Mode 6 will last for about 1 h. The red, green and blue average of a picture are in a range of 0 to 255. The calibration application considers a picture as underexposed or overexposed if R,G,B<0.5 or R,G,B>254.5. This protects the calibration against not valid images which would distort the gain values. Additionally it enables the use of a full exposure range, when this kind of images are disposed. The following figures demonstrates, that all images under 100 µs are disposed as underexposed:



Uncalibrated and calibrated sensor RGB channel curves in logarithmic scale.

Calibration mode 7 reads the auto exposure value and generates the following exposure array:

[0.25*ss, 0.50*ss, 0.75*ss, ss, 1.25*ss, 1.50*ss, 1,75*ss]

where ss is the shutter speed.

The mode 7 normally captures pictures which are properly exposed, and are valid for calibration. If consisted calibration with the same exposure time is required, a predefined array (i.e. exp_fast or exp_medium1) could be used in the code by replacing the original exposure times with exposure times calculated from the auto exposure value.

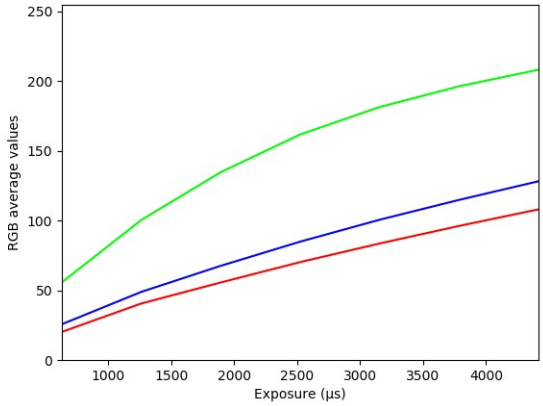
In the forth stage default ranges for red and blue gains are accepted or modified:

```
Default calibration ranges:
Red gain range:  1.0-2.0
Red gain step:   0.1
Blue gain range: 1.0-2.0
Blue gain step:  0.1

Accept default ranges (Y/N, Default Y: <Enter>): n
Current ranges disabled

Select red gain min (1.0...2.0, Default=1.0: <Enter>): 1.6
Select red gain max (1.6...2.0, Default=2.0: <Enter>): 1.8
Select red gain step (0.001...0.2, Default=0.1: <Enter>): 0.02

Select blue gain min (1.0...2.0, Default=1.0: <Enter>): 1.3
Select blue gain max (1.3...2.0, Default=2.0: <Enter>): 1.5
Select blue gain step (0.001...0.2, Default=0.1: <Enter>): 0.02

Select scale type (linear, log; Default=linear):
Default value selected: linear
```

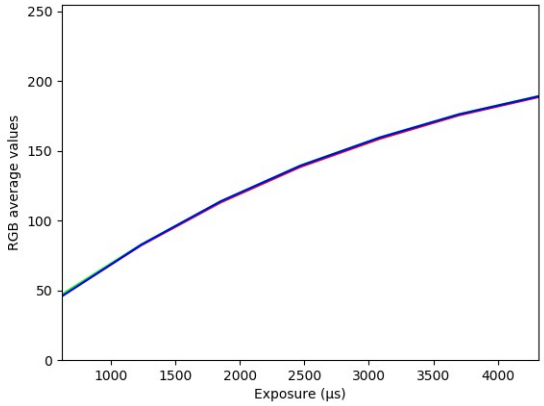
Pictures are captured in stage 5 with the raspistill program. Normally the duration of this stage is from few minutes up to 1 hour. In stage 6 color analysis is done for each series of pictures and calibration figures are saved into a figures directory. In the last stage optimum red and blue gains are calculated. Here is a part of a calibration log file, where the results are presented:

```
…
Calibration images: 847
Optimal r_gain: 1.66
Optimal b_gain: 1.44
Total distance: 11.140924479166713
Time elapsed: 0:11:12.595477
```

The following pictures demonstrates the difference of uncalibrated and calibrated gains:



Uncalibrated white card picture and sensor RGB curves (gains: red 1.0 and blue 1.0).



Calibrated white card picture and sensor RGB curves (gains: red 1.66 and blue 1.44).

12.2 Calibration with ROI

If the the white card or calibration area is too small, the camera module has to be closer the object or ROI must be applied. The field of view can be narrowed easily with the ROIs variable in the code:

```
roi_x0=0.0
roi_y0=0.0
roi_w=1.0
roi_h=1.0
```

The default values gives full field of view. All values are normalized from 0.0 to 1.0. Two first variables defines the coordinates where ROI starts from the upper left corner. The latter varibles defines width and height for ROI.Raspstill can be used in order to get desired ROI like the following example:

`raspistill -t 0 -roi 0.25,0.25,0.5,0.5` (middle area of FOV with 25 % margins)

12.3 Analyze only mode

It is possible to do post-calibration from existing calibration pictures. Optimum gains can be calculated very fast if the calibration file exist (rgb-calibration_name*.txt). In such case, the gains can be calculated very fast with optimum_gains.py. However, if the calibration file is missing, change of scale type is desired (log→linear or linear → log) or calibration figures has to be re-created, then campreview.py can be executed in *analyze only mode*.

The Analyze only mode asks path to images (calibration base directory), calibration name and the scale type. Thereafter color analysis is performed for all valid calibration directories, which mean that the calibration name must be same as the calibration directories base name followed by _rgain,bgain, where rgain and bgain are positive numbers. All invalid directory names are disposed. The calibration program will create automatically these calibration directory names correctly when analyze only mode is disabled.

At the final stage optimum red and blue gains are calculated.

13 Get Optimum Gains from a Calibration File

optimum_gains.py

- Get optimal red and blue gains from a calibration file

This utility reads a calibration file (rgb-project_name.txt), which can be produced by calibratecam.py application. Optimal red and blue gains are calculated when the calibration data is read. Here is an example how to execute optimal_gains.py:

```
$ optimal_gains.py rgb-manfrotto-white-2xled8-step0.02_iso100.txt
Calibration images: 847
Optimal r_gain =  1.66
Optimal b_gain =  1.44
Total distance =  11.140924479166713
```

14 Capturing Pictures in Manual Mode

These applications are designed to capture pictures manually by pressing the space bar. The reason for two programs is that there is a different delay before a new picture can be captured. A live preview will lead to longer time before a new picture can be captured. That is to reason for the camera application without a preview.

14.1 Capturing Pictures with Preview Enabled

capturepics-preview.py

- Camera preview is enabled
- Captures pictures by pressing space bar
- Auto numbering of captures
- Delay before capturing next picture is several seconds
- Requires a RPI and a camera module
- Uses PiCamera library

This application would be very convenient if the time between captures would be smaller. The application can be used when the composition must be accurate and the amount of captures is small.

Here is an example how to run this application:

```
$ capturepics-preview.py
Raspberry Pi capture pictures
Select image quality (1...100; default: 90):
Default value selected: 90

List disk and partitions:
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdb           8:16   0 223.6G  0 disk
└─sdb1        8:17   0 223.6G  0 part /media/pi/data
mmcblk0     179:0    0  14.8G  0 disk
├─mmcblk0p1 179:1    0   2.2G  0 part
├─mmcblk0p2 179:2    0     1K  0 part
├─mmcblk0p5 179:5    0    32M  0 part
├─mmcblk0p6 179:6    0   256M  0 part /boot
└─mmcblk0p7 179:7    0  12.3G  0 part /

Path to images (current directory: <Enter>):
Project name (default=pic: <Enter>):
Select ISO (100, 200, ... 800): 100
Select exposure time (1...330000) µs: 35000
```

```
Select first frame number (default=1: <Enter>):
Default value selected: 1
Select digits (min=1, default=4: <Enter>):
Default value selected: 4
No artist.txt file
Full screen and cropped preview (Y/N, default: N <ENTER>)
Default selected

Start capturing images: ENTER
Capture image: SPACE
Exit program:  ESC

Capture mode enabled.
pic_0001
pic_0002
```

ESC key pressed → termination of the application

14.2 Capturing Pictures without Preview

capturepics.py

- Camera preview is disabled
- Captures pictures by pressing space bar
- Auto numbering of captures
- Time between captures is about 1 second
- Requires a RPI and a camera module
- Requires raspistill ? <-???
- Generates and executes scripts (capture commands) which can be executed without Python

The capturepics.py generates commands, which will capture pictures by using raspistill program. The delay before capturing a new picture is much smaller than with the capturepics-preview.py. The drawback is that there is no preview. Therefore, the composition must be done before using this application.

This example shows how to use capturepics.py:

```
$ capturepics.py

Raspberry Pi capture pictures
Select image quality (1...100; default: 90):
Default value selected: 90

List disk and partitions:
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdb           8:16   0 223.6G  0 disk
└─sdb1        8:17   0 223.6G  0 part /media/pi/data
mmcblk0     179:0    0  14.8G  0 disk
├─mmcblk0p1 179:1    0   2.2G  0 part
├─mmcblk0p2 179:2    0     1K  0 part
├─mmcblk0p5 179:5    0    32M  0 part
├─mmcblk0p6 179:6    0   256M  0 part /boot
└─mmcblk0p7 179:7    0  12.3G  0 part /

Path to images (current directory: <Enter>):
Project name (default=pic: <Enter>):
Select ISO (100, 200, ... 800): 100
Select exposure time (1...330000) µs: 35000
Select first frame number (default=1: <Enter>):
Default value selected: 1
Select digits (min=1, default=4: <Enter>):
```

```
Default value selected: 4
No artist.txt file

Start capturing images: ENTER
Capture image: SPACE
Exit program:  ESC

Capture mode enabled.
pic_0001
pic_0002
pic_0003
pic_0004
pic_0005
```

ESC key pressed → termination of the application

The log file which contains data about to project, looks like this:

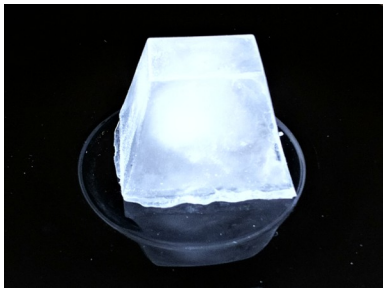
```
Log created on 2019.12.29-18:14:07

File path: Not defined

Artist:
Capture pictures parameters:
Quality: 90
ISO value: 100
Exposure: 35000 µs
Start frame: 1
Digits: 4
First file name: pic_0001.png

raspistill -n -t 1 -ISO 100 -q 90 -ss 35000 -bm -drc high -o pic_0001.png
raspistill -n -t 1 -ISO 100 -q 90 -ss 35000 -bm -drc high -o pic_0002.png
raspistill -n -t 1 -ISO 100 -q 90 -ss 35000 -bm -drc high -o pic_0003.png
raspistill -n -t 1 -ISO 100 -q 90 -ss 35000 -bm -drc high -o pic_0004.png
raspistill -n -t 1 -ISO 100 -q 90 -ss 35000 -bm -drc high -o pic_0005.png
```

15 Time-Lapse Photography

timelapse.py

- Capture pictures in time-lapse mode
- Minimum interval between captures is 1 s
- Pictures are stored in JPG format
- Requires a RPI and a camera module
- Requires raspistill
- Generates and executes a script (capture command) which can be executed without Python

The idea of time-lapse photography can be easily demonstrated by melting an ice cube:



0 min

2 min

4 min

6 min

8 min

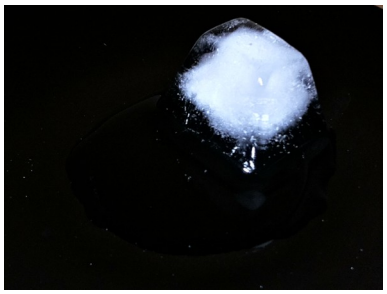
10 min

12 min

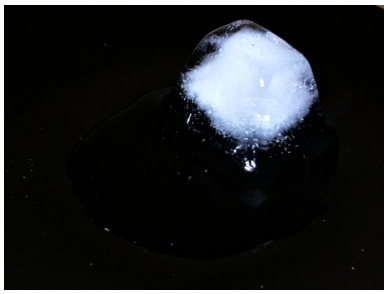
14 min

16 min

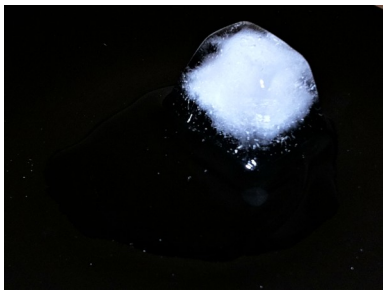
18 min



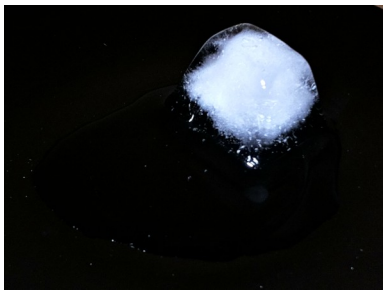
20 min



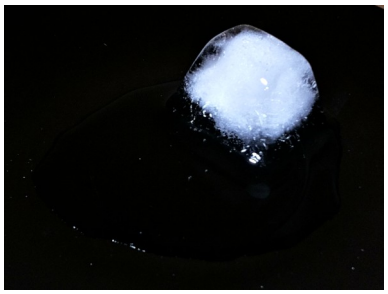
22 min



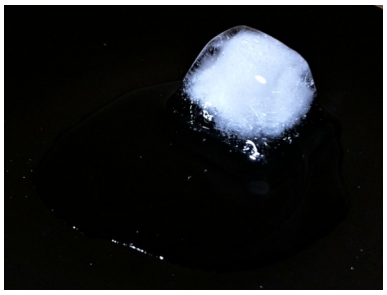
24 min



26 min



28 min



30 min

32 min

Capture parameters for the pictures above:

```
Time-lapse parameters:
Quality: 90
ISO value: 100
Exposure: 30000
Duration: 1800000
Interval: 120000
Start frame: 1
```

This application does not provide capture preview in order to avoid loss of frames. The following example shows what kind of questions has to be answered before starting to capture 10 s time-lapse with 1 s interval:

```
$ timelapse.py
Time-lapse program for Raspberry Pi camera v. 2.x (c) Kim Miikki
2019
Select image quality (1...100; default: 90): 90

List disk and partitions:
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda           8:16   0 223.6G  0 disk
└─sda1        8:17   0 223.6G  0 part /media/pi/data
mmcblk0     179:0    0  14.8G  0 disk
├─mmcblk0p1 179:1    0   2.2G  0 part
├─mmcblk0p2 179:2    0     1K  0 part
├─mmcblk0p5 179:5    0    32M  0 part
├─mmcblk0p6 179:6    0   256M  0 part /boot
└─mmcblk0p7 179:7    0  12.3G  0 part /

Path to images (current directory: <Enter>):
Time-lapse project name: tl
Select ISO (100, 200, ... 800): 100
Select exposure time (1...330000) µs: 35000
Select interval (1...86400) s: 1
Select time-lapse duration (1...31536000) s: 10

Start time-lapse shooting now (Y/N, Default Y: <Enter>):
Default selected: starting the time-lapse shooting.

...
```

16 Video Recording

fpsvideo.py

- Record videos in slow, normal or high speed (up to 200 FPS)
- Requires a RPI and a camera module
- Requires raspivid
- Generates and executes a script (recording command) which can be executed without Python

Although it is possible to record videos with the RPI camera module up to 1000 frames per second [1], **fpsvideo.py** uses standard RPI **raspivid** application which limits the FPS to 200. This can be done without compiling source codes and playing with raw Bayer format frames.

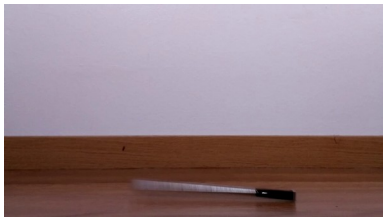
Here is an example of a 90 FPS at 1280x720 (720 p) what raspivid is capable to produce:

Frame: 420

Frame: 435

Frame: 445

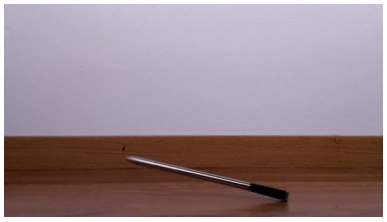
Frame: 455



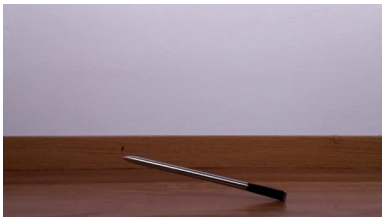
Frame: **463**

Frame:**464**

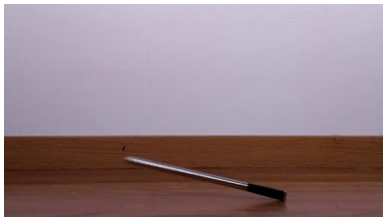
Frame: **465**



Frame: **466**

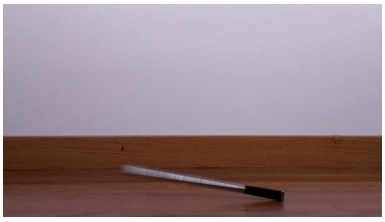


Frame: **467**

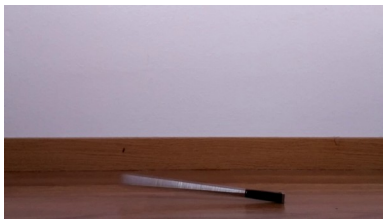


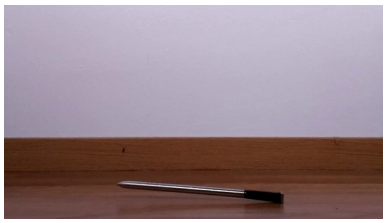
Frame: **468**

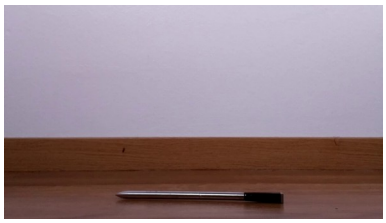
Frame: **469**



Frame: **470**


Frame: **471**


Frame: 475


Frame: 480

The video slowdown rate is 3.6 when the normal play speed is 25 FPS. The capture parameters are shown below:

```
Log created on 2019.12.31-12:27:01

File path: Not defined

Files:
Video: probe-1280x720_90fps_11111ss_30s.h264
PTS:   probe-1280x720_90fps_11111ss_30s.pts
Log:   probe-1280x720_90fps_11111ss_30s.log
REC:   probe-1280x720_90fps_11111ss_30s.rec

FPS video parameters:
Sensor mode: 6
Resolution: 1280x720
Exposure: 11111 µs
FPS: 90
AWB: Auto
Duration: 30 s
Preview: on

Record video:

raspivid -md 6 -w 1280 -h 720 -ss 11111 -fps 90 -t 30000 -v -pts
probe-1280x720_90fps_11111ss_30s.pts -o probe-
1280x720_90fps_11111ss_30s.h264 2>&1 | tee probe-
1280x720_90fps_11111ss_30s.rec

mkvmerge -o probe-1280x720_90fps_11111ss_30s.mkv probe-
1280x720_90fps_11111ss_30s.h264 2>&1 | tee -a probe-
1280x720_90fps_11111ss_30s.rec
```

In this experiment, compressed air was used to make the probe to fall. The settle down time for the probe, after it started to fall was about 0.7 s. The frames were extracted by using FFmpeg [2].

This application generates scripts, which will start raspivid and mkvmerge [3]. The first program will record the video and the second convert it to a MKV file.

If normal video playback is 25 FPS, then the slowdown (depending on the video mode), is at maximum FPS:

Md	Video resolution	Maximum FPS	Video slowdown
1	1920x1080	30	1.2
2	1920x1080	15	0.6
3	1920x1080	15	0.6
4	1640x1232	40	1.6
5	1640x922	40	1.6
6	1280x720	90	3.6
7	640x480	200	8

Video slowdown at 25 FPS

Standard raspistill command for composition is not enough before recording a video. RPI application vidpreview.py will do the job with ROI.

This application creates some additional files which can be useful for debugging or if exact times for frames are required:

LOG All selections and video commands are stored in this file
PTS Save timestamp information to the specified file
REC Redirect stdout and stderr to the REC file

Videos are stored at least as H.264 and (optionally) transformed to MKV.

The requested FPS is not necessary what is acquired. Hence a timing correction is required. The following formula calculates the correct time for one frame

$$t_f = \frac{FPS_{requested}}{FPS_{reference}} \cdot \frac{recorded\,video\,duration}{frames\,recorded}$$

where $FPS_{reference}$ is the normal playback speed (25 frames/s).

Running the Application

This example shows how to run the application:

```
$ fpsvideo.py
FPS video program for Raspberry Pi camera v. 2.x (c) Kim Miikki
2019

List disk and partitions:
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda           8:0    0 223.6G  0 disk
└─sda1        8:1    0 223.6G  0 part /media/pi/data
mmcblk0     179:0    0  14.8G  0 disk
├─mmcblk0p1 179:1    0   2.2G  0 part
├─mmcblk0p2 179:2    0     1K  0 part
├─mmcblk0p5 179:5    0    32M  0 part
├─mmcblk0p6 179:6    0   256M  0 part /boot
└─mmcblk0p7 179:7    0  12.3G  0 part /

Path to images (current directory: <Enter>):
Video project name: test
Select bitrate (1-25000000 bits/s, automatic: <Enter>):
Automatic bitrate selected.

Video modes
1     1920x1080      16:9      30
2     1920x1080      4:3       15
3     1920x1080      4:3       15
4     1640x1232      4:3       40
5     1640x922       16:9      40
6     1280x720       16:9      90
7     640x480   4:3       200
8     640x480   4:3       180

Select video mode (1...8): 7
Select FPS: (40...200, Default 200: <Enter>):
Default selected: 200 FPS
Select exposure time (1...5000) µs: 5000
Select video duration (1...31536000) s: 2

Enable preview (Y/N, Default N: <Enter>): y
Preview enabled
Convert to MKV file (Y/N, Default Y: <Enter>):
Default selected: MKV conversion enabled
Start recording now (Y/N, Default Y: <Enter>):
Default selected: starting to record.
```

```
raspivid -md 7 -w 640 -h 480 -ss 5000 -fps 200 -t 2000 -ex off -ag
1.0 -dg 1.0 -awb off -awbg 1,1 -v -pts test-
640x480_200fps_5000ss_2s.pts -o test-640x480_200fps_5000ss_2s.h264
2>&1 | tee test-640x480_200fps_5000ss_2s.rec

mkvmerge -o test-640x480_200fps_5000ss_2s.mkv test-
640x480_200fps_5000ss_2s.h264 2>&1 | tee -a test-
640x480_200fps_5000ss_2s.rec

"raspivid" Camera App (commit )

Camera Name imx219
Width 640, Height 480, filename test-640x480_200fps_5000ss_2s.h264
Using camera 0, sensor mode 7

GPS output Disabled

bitrate 17000000, framerate 200, time delay 2000
H264 Profile high
H264 Level 4
H264 Quantisation level 0, Inline headers No
H264 Fill SPS Timings No
H264 Intra refresh type (null), period -1
H264 Slices 1
Wait method : Simple capture
Initial state 'record'

Preview Yes, Full screen Yes
Preview window 0,0,1024,768
Opacity 255
Sharpness 0, Contrast 0, Brightness 50
Saturation 0, ISO 0, Video Stabilisation No, Exposure compensation 0
Exposure Mode 'off', AWB Mode 'off', Image Effect 'none'
Flicker Avoid Mode 'off'
Metering Mode 'average', Colour Effect Enabled No with U = 128, V = 128
Rotation 0, hflip No, vflip No
ROI x 0.000000, y 0.000000, w 1.000000 h 1.000000
Camera component done
Encoder component done
Starting component connection stage
Connecting camera preview port to preview input port
Starting video preview
Connecting camera video port to encoder input port
Opening output file "test-640x480_200fps_5000ss_2s.h264"
Opening output file "test-640x480_200fps_5000ss_2s.pts"
Enabling encoder output port
Starting video capture
Finished capture
Closing down
Close down completed, all components disconnected, disabled and destroyed
```

Version – 21.9.2020

```
mkvmerge v31.0.0 ('Dolores In A Shoestand') 32-bit
'test-640x480_200fps_5000ss_2s.h264': Using the demultiplexer for the format
'AVC/h.264'.
'test-640x480_200fps_5000ss_2s.h264' track 0: Using the output module for the
format 'AVC/h.264 (unframed)'.
The file 'test-640x480_200fps_5000ss_2s.mkv' has been opened for writing.
Warning: 'test-640x480_200fps_5000ss_2s.h264' track 0: This AVC/h.264 track's
timing information indicates that it uses a variable frame rate. However, no
default duration nor an external timestamp file has been provided for it, nor
does the source container provide timestamps. The resulting timestamps may not
be useful.
Progress: 100%
The cue entries (the index) are being written…
Multiplexing took 0 seconds.
```

Cited references:

[1] https://github.com/6by9/raspiraw, 1.1.2020
[2] https://www.ffmpeg.org, 2.1.2020
[3] https://mkvtoolnix.download/doc/mkvmerge.html, 2.1.2020

17 Creating Videos from Pictures

gtlvideo.py

- Generates videos in MKV format from PNG, JPG and CR2 pictures
- Python version 3.5 or higher is required
- Requires exifread, PIL, numpy and pandas Python libraries
- Requires FFmpeg and darktable_cli (if using CR2 files)
- Generates and executes a script (recording command) which can be executed without Python

This application is the most complicated of these RPI camera applications. PNG, JPG or CR2 files are required for video creation. The program search for those files in current directory. If it finds same amount of CR2, PNG and JPG files, the priority for video creation is CR2 > PNG > JPG. The images must be numbered consecutively (with leading zeros) before the extension and the file name have to be constant before the numbering. The amount of image files (PNG, JPG or CR2) are counted and checked that there are no missing or additional frames. Otherwise to program will terminate. The program assumes that all frames have the same resolution.

In order to create a video of pictures, aspect ratio for source and target is one of the most important thing to consider. The easiest thing is to crop images without scaling. In this case, an arbitrary A/R can be selected. In crop and scale mode the maximum amount of information will be saved when either source width or height will be constant, and cropping have to be done on the opposite axis, in order to get same geometry for source and destination resolution.

However, the program start by asking for the project name. By pressing Enter the default project name will be *video*. The next question is about video mode. Possible choices are landscape or portrait (default is landscape).

A dynamic table of video modes will be presented on the basis of the picture resolution. This example illustrates this (resolution was 1024x768):

```
Video modes
 0 Native  1024x768    1024:768
 9 SD      720x576     16:9
10 XGA     1024x768    4:3
11 VGA     640x480     4:3
12 GIF     320x200     4:3
13 Custom  ?x?         x:y

Select video mode (Default mode is: 10 = 1024x768 <Enter>):
```

Next selection is how to crop the images when producing a video. Default mode is C&R on order to save as much information as possible. The images on the following pages will illustrate what different cropping means.

Original picture 1

Version – 21.9.2020

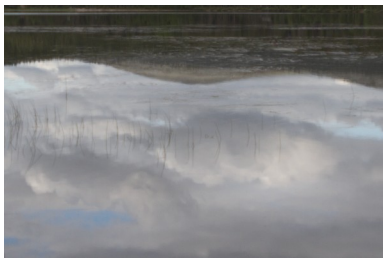
Predefined crop areas in Crop mode:

Top and Left

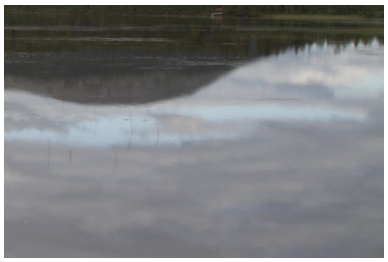
Top and Center

Top and Right

Center and Left



Center and Center



Center and Right

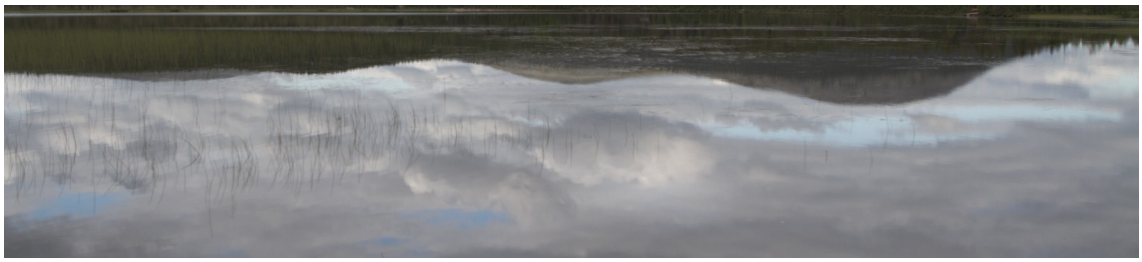
Bottom and Left

Bottom and Center

Bottom and Right

Predefined crop areas in Crop & Resize mode:

Top crop



Horizontal center crop

Bottom crop

Left crop

Center vertical crop

Right crop

Custom crop area:

Original picture 2

ROI: Crop size 640x480 and start coordinates: (1111, 315)

The menus depends of the source and target geometry of resolutions, and choices done. Therefore it is not useful to show every possible menu in this manual.

After crop menus (if cropping was required) the program asks for video FPS (default is 25). Then the duration of the video is calculated and shown. Next decision has to be made on CRF (video quality; default is 23). Lower value is better but the size of video file will increase if better quality is required.

Finally, a summary of selections is given, and the final question asks if video creation should be done.

Two sub directories will be created: png and video. All crops and image conversion files will be stored in the first one. The video directory consist of the video and log files.

This example illustrates how to run the program:

```
$ gtlvideo.py
Generate time-lapse video from images
Version 0.921 (beta) (C) Kim Miikki 2020

Python version: 3.5

Scanning supported image files in current directory...

CR2 info:
Files:  0
Width:  0
Height: 0

PNG info:
Files:  49
Width:  5202
Height: 3464

JPG info:
Files:  0
Width:  0
Height: 0
```

```
Sample file name: IMG_5010.png
Digits before file extension: 4
Default picture format: PNG

Select project name (Default: video <Enter>): sunset
Project name: sunset
Video name  : sunset.mkv

Select video mode (landscape=L, portrait=P, Default: Landscape
<Enter>):
Landscape video mode selected

Video modes
 0 Native  5202x3464   5202:3464
 2 4K/DCI  4096x2160   256:135
 3 4K/UHD  3840x2160   16:9
 4 2K      2560x1440   16:9
 5 Full HD 1920x1080   16:9
 6 Pi/md4  1640x1232   4:3
 7 Pi/md5  1640x922    16:9
 8 HD      1280x720    16:9
 9 SD      720x576     16:9
10 XGA     1024x768    4:3
11 VGA     640x480     4:3
12 GIF     320x200     4:3
13 Custom  ?x?         x:y

Select video mode (Default mode is: 0 = 5202x3464 <Enter>): 5

Crop & Resize or only Crop?
Select Crop & Resize or only Crop (C&R=Y, C=N, Default: C&R
<Enter>):
C&R mode selected

Crop mode: vertical
Image original size:        5202 x 3464
Image size before scaling:  5202 x 2926
Video size after scaling:   1920 x 1080

0 = Middle crop
1 = Top crop
2 = Bottom crop
3 = Custom crop start Y (0...537)
Select crop mode (Default: 0)? 2
```

```
Standard video frame rates: 24, 25, 30, 48, 50 and 60
Select video FPS (1...60, Default=25 <Enter>): 10

Video information
Frames:   49
FPS:      10
Duration: 4.9 s

Select 10 frames/s (Y/N, Default: Y <Enter>):
Default FPS selected

Constant Rate Factor: 0 (lossless) - 18 - 23 (default) - 28 - 51
(worst)
Select video CRF for x264 (0...51, Default=23 <Enter>):
Default CRF selected

"/home/.../png/"

---------------------

Summary of selections:

Program version: 0.921 (beta)

Project name: sunset
Video name:   sunset.mkv

Video resolution is smaller than image size

Crop mode:                    crop and resize
Crop direction:               vertical
Crop selection:               Bottom crop
Crop coordinates:             (0,537)-(5201,3463)
Crop geometry error:          -0.00 %

Image size:                   5202x3464
Cropped image size:           5202x2926
Video size:                   1920x1080

Picture geometry (h/w):       0.6658977316416763
Cropped image geometry (h/w): 0.5624759707804691
Video geometry (h/w):         0.5625

File format:                  PNG
File sample name:             IMG_5010.png
First new file name:          IMG_4980.png
Working path:                 /home/...
PNG path:                     /home/.../png/
Video and log path:           /home/.../video/
Video name:                   sunset.mkv
```

```
Log name:                       sunset.log
File format digits:             4
First frame:                    4980
Last frame:                     5028
Calculated number of frames:    49
Files:                          49
FPS:                            10
Duration:                       4.9 s
Constant Rate Factor (CRF):     23

Stage 1: PNG→PNG
for i in "/home/.../"*.png ; do ffmpeg -y -i "$i" -vf
crop=5202:2926:0:537,scale=1920:1080 "/home/.../png/""sunset-$
(basename "$i" ".png")".png ; done

Stage 2: Create video
ffmpeg -y -r 10 -start_number 4980 -i "/home/.../png/sunset-IMG_
%04d.png" -c:v libx264 -crf 23 "/home/.../video/sunset.mkv"

Are you sure you want to proceed (Y/N, Default=Y <Enter>)? y

...

Stage 1 duration: 0:01:20.655601
Stage 2 duration: 0:00:04.607428
```

18 RGB Color Analysis

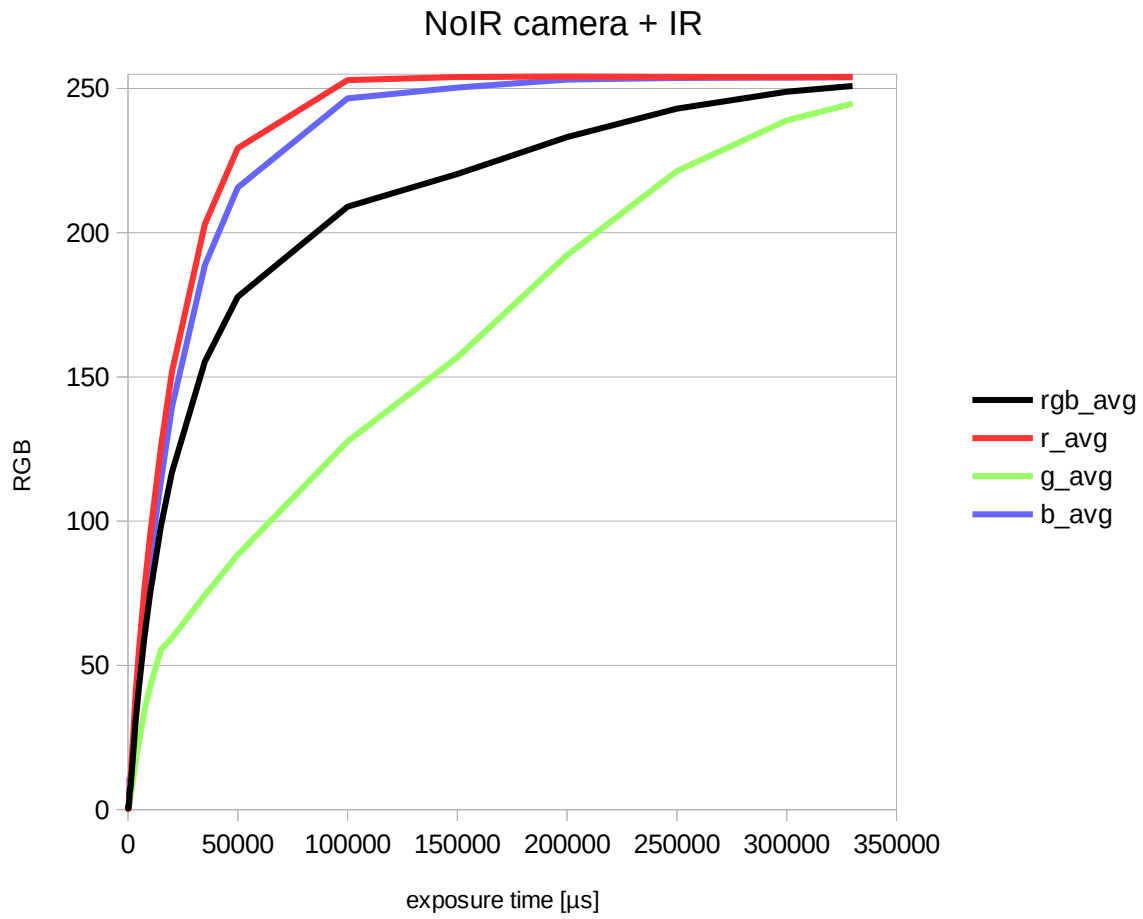
rgbinfo.py

- Calculate RGB and BW means of PNG and JPG pictures
- Requires numpy and opencv Python libraries
- RPI is not required

Current directory is scanned and color analysis is carried out if the picture is in PNG or JPG format. Here is an example of an experiment where the palm tree was illuminated by an infrared heater. The RPI camera module used was v. 2.x NoIR. The sample picture’s exposure time was 20 ms.

VIS and IR illuminated composition at 20 ms exposure.

An exposure bracketing experiment was performed and here is the result as a function of time:



Color analysis performed by rgbinfo.py.

Version – 21.9.2020

This program calculates the average R, G and B channels and BW of each picture in current directory.

Here is an example run of the program:

```
$ rgbinfo.py
RGBinfo 1.0, (c) Kim Miikki 2019

Analyzing:
20190322_102606.png: 4032x2268
20190322_102611.png: 4032x2268
20190322_102612.png: 4032x2268

Pictures analyzed: 3
Time elapsed: 0:00:01.290984
```

```
$ cat rgb-png.txt
picture_name;rgb_avg;r_avg;g_avg;b_avg;rgb_ratio;r_ratio;g_ratio;b_ratio
20190322_102606.png;106.70739222901095;95.30768676426332;104.93878688306599;119.
87570303970354;0.41846036168239586;0.3737556343696601;0.41152465444339603;0.4701
0079623413153
20190322_102611.png;126.4928493859821;119.13333105876096;125.56441917044596;134.
7807979287394;0.4960503897489494;0.46718953356376847;0.4924094869429253;0.528552
1487401544
20190322_102612.png;125.87577109461755;118.75658915186445;124.93207055198623;133
.93865358000195;0.49363047488085315;0.46571211432103704;0.4899296884391617;0.525
2496218823606

Pictures analyzed:;3
Time elapsed:;0:00:01.290984
```

19 Splitting RGB Channels into Separate Images

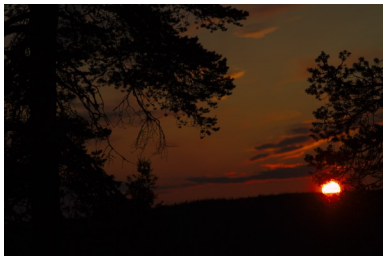
The RPI camera software suite provides tools to split images into RGB channels in two different ways. The first one splits images into 3 channel red, green and blue colored images. The second one splits them into 1 channel gray scale images, where the RGB channel information is stored as a filename prefix of the color split image.

19.1 Splitting RGB Channels into separate RGB Images

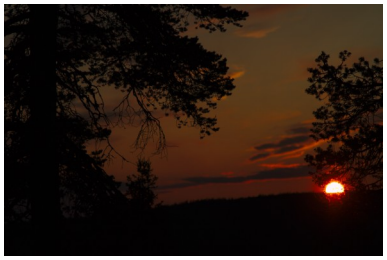
rgbsplit.py

- Split RGB channels of PNG or JPG pictures into separate images
- Requires numpy and cv2 Python libraries
- RPI is not required

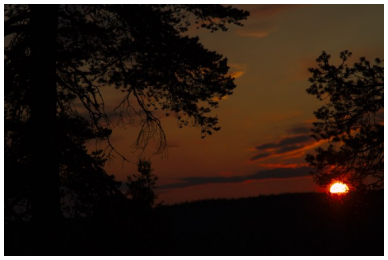
An experiment was conducted in Ylläs, Lapland, Finland on 16.7.2019 (by shooting in time-lapse mode) with a Canon 7D camera. The interval between pictures was 20 s. This sample shows how rgbsplit.py can be used to split images into red, green and blue color channel images, and black & white images:



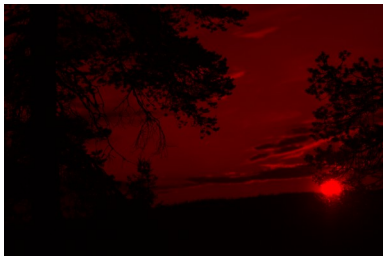
sunset-IMG_5010.png



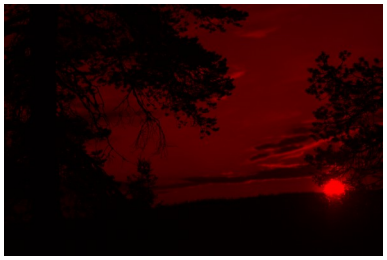
sunset-IMG_5011.png



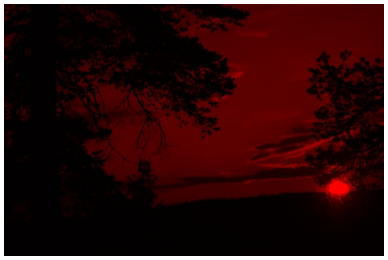
sunset-IMG_5012.png



R-sunset-IMG_5010.png



R-sunset-IMG_5011.png



R-sunset-IMG_5012.png

G-sunset-IMG_5010.png

G-sunset-IMG_5011.png

G-sunset-IMG_5012.png

B-sunset-IMG_5010.png

B-sunset-IMG_5011.png

B-sunset-IMG_5012.png

BW-sunset-IMG_5010.png

BW-sunset-IMG_5011.png

BW-sunset-IMG_5012.png

This program does not require a RPI. The only requirements is that the pictures have to be in PNG or JPG format. R-, G-, B- and BW- prefixed images are stored in the same directory where the source images are located.

This conversion was executed this way:

```
$ rgbsplit.py
RGBsplit 1.0, (c) Kim Miikki 2019

Splitting color channels:
sunset-IMG_5010.png: 5202x3464
sunset-IMG_5011.png: 5202x3464
sunset-IMG_5012.png: 5202x3464

Pictures processed: 3
Time elapsed: 0:00:08.240715
```

19.2 Splitting RGB Channels into Separate Gray Scale Images

bw-rgbsplit.py

- Split RGB channels of PNG or JPG pictures into separate gray scale images
- Requires numpy and cv2 Python libraries
- RPI is not required

This program splits RGB color images into 8-bit gray scale images, like the following example shows:

Original 3 channel RGB picture

Red channel as 8-bit gray scale image

Green channel as 8-bit gray scale image

Blue channel as 8-bit gray scale image

The color channel information is stored in file names as prefixes:
BWR-: Red channel
BWG-: Green channel
BWB-: Blue channel

Otherwise original file names are stored directly after these prefixes. The program should be executed in the same directory, where the pictures are located:

$ **bw-rgbsplit.py**
BW-RGBsplit 1.0, (c) Kim Miikki 2020

Splitting color channels:
20200406_090114.jpg: 4032x3024

Pictures processed: 1
Time elapsed: 0:00:00.644501

20 Combining RGB split Images into Color Images

All of these color channel separation operations can be reversed by merging color channels back into RGB color images. Normally merging is done after enhancing etc. the individual color channels.

20.1 Combining 3 Channel RGB split Images into Color Images

rgbcombine.py

- Merge RGB split 3 channel PNG or JPG images into color images
- Requires numpy and cv2 Python libraries
- RPI is not required

This program search for B-, G- and R- prefixed images in current directory, and merge them into color images. The color split images must be 3 channel RGB color split images. Here is an example how to use the program:

```
$ ls -1
B-20200406_090114.jpg
G-20200406_090114.jpg
R-20200406_090114.jpg

$ rgbcombine.py
RGBcombine 1.0, (c) Kim Miikki 2020

Combining color channels:
20200406_090114.jpg: 4032x3024

Pictures processed: 1

Time elapsed: 0:00:00.858192
```

20.2 Combining 1 Channel RGB split Images into Color Images

bw-rgbcombine.py

- Merge RGB split 1 channel PNG or JPG images into color images
- Requires numpy and cv2 Python libraries
- RPI is not required

This program search for BWB-, BWG- and BWR- prefixed images in current directory, and merge them into color images. The color split images must be 1 channel gray scale images. The following example demonstrate how to use this program:

```
$ ls -1
BWB-20200406_090114.jpg
BWG-20200406_090114.jpg
BWR-20200406_090114.jpg

$ bw-rgbcombine.py
BW-RGBcombine 1.0, (c) Kim Miikki 2020

Combining color channels:
20200406_090114.jpg: 4032x3024

Pictures processed: 1

Time elapsed: 0:00:00.600683
```

21 Convert Images to PNG Format

pics2png.py

- Convert image formats which PIL supports to PNG format
- Requires python PIL library
- RPI is not required

This simple program tries to read all files in current directory as PIL images. If it succeeds, then the image will be converted as a PNG file saved in a png named sub directory. The program does not require a RPI.

Here is an example run of the program:

```
$ pics2png.py
Pics to PNG files 1.0, (c) Kim Miikki 2019

20190322_102606.jpg (4032x2268) -> 20190322_102606.png
20190322_102611.jpg (4032x2268) -> 20190322_102611.png
20190322_102612.jpg (4032x2268) -> 20190322_102612.png
No PNG conversion: 20190322_102628.mp4

Pictures converted: 3
Time elapsed:       0:00:07.493193
```

22 Pictures to Time-lapse Files

gen_tlfiles.py

- Copy and rename PNG or JPG files to time-lapse files
- RPI is not required

This application converts file names into time-lapse format (img_0001, img_0002 etc.). The files are copied and renamed. The program calculates a ratio of the nearest $10^{\text{digits of pictures}}$. If the amount of pictures is equal or more than 90 % to the following tenfold, the program will increase leading zeroes by one. New files are stored in a *tl* named directory under the the current directory.

Here is an example run of the program:

```
$ gen_tlfiles.py
Pictures to Time-lapse Files 1.0, (c) Kim Miikki 2020

cp /home/kim/pictemp/20190710_184746.jpg /home/kim/pictemp/tl/img_01.jpg
cp /home/kim/pictemp/20190710_184747.jpg /home/kim/pictemp/tl/img_02.jpg
cp /home/kim/pictemp/20190710_184749.jpg /home/kim/pictemp/tl/img_03.jpg
cp /home/kim/pictemp/20190710_184805.jpg /home/kim/pictemp/tl/img_04.jpg
cp /home/kim/pictemp/20190710_184808.jpg /home/kim/pictemp/tl/img_05.jpg
cp /home/kim/pictemp/20190710_184810.jpg /home/kim/pictemp/tl/img_06.jpg
```

```
cp /home/kim/pictemp/20190710_184814.jpg /home/kim/pictemp/tl/img_07.jpg
cp /home/kim/pictemp/20190710_184815.jpg /home/kim/pictemp/tl/img_08.jpg
cp /home/kim/pictemp/20190710_184821.jpg /home/kim/pictemp/tl/img_09.jpg
cp /home/kim/pictemp/20190710_184825.jpg /home/kim/pictemp/tl/img_10.jpg
cp /home/kim/pictemp/20190710_184826.jpg /home/kim/pictemp/tl/img_11.jpg
cp /home/kim/pictemp/20190710_184827.jpg /home/kim/pictemp/tl/img_12.jpg
cp /home/kim/pictemp/20190710_184828.jpg /home/kim/pictemp/tl/img_13.jpg
cp /home/kim/pictemp/20190710_184830.jpg /home/kim/pictemp/tl/img_14.jpg
cp /home/kim/pictemp/20190710_184832.jpg /home/kim/pictemp/tl/img_15.jpg
cp /home/kim/pictemp/20190710_184833.jpg /home/kim/pictemp/tl/img_16.jpg
cp /home/kim/pictemp/20190710_184834.jpg /home/kim/pictemp/tl/img_17.jpg
cp /home/kim/pictemp/20190710_184835.jpg /home/kim/pictemp/tl/img_18.jpg
cp /home/kim/pictemp/20190710_185016.jpg /home/kim/pictemp/tl/img_19.jpg
cp /home/kim/pictemp/20190710_185020.jpg /home/kim/pictemp/tl/img_20.jpg
cp /home/kim/pictemp/20190710_185026.jpg /home/kim/pictemp/tl/img_21.jpg
cp /home/kim/pictemp/20190710_185105.jpg /home/kim/pictemp/tl/img_22.jpg

Time elapsed:0:00:00.072638
```

23 Additional Info

RPI High Speed Camera Suite, documentation and information for the RPI camera module:

https://github.com/kmiikki/rpi-camera, 27.8.2020

https://www.raspberrypi.org/documentation/raspbian/applications/camera.md, 27.12.2019

https://picamera.readthedocs.io/en/release-1.13, 2.1.2020