PlantAnalyzer Documentation

Release 0.1

Jan Lukas Bosse

August 31, 2016

CHAPTER

ONE

THE *PLANTANALYZER* CODE

The source code of the *PlantAnalyer* is grouped into six modules:

- The main module is the *measurement* class. This class holds all the data of an measurement and also the methods to analyze this data. It can be used on his own without the GUI in your Python scripts to allow for more specific measurements. This is especially useful, if you don't wish to extract all data possible but e.g. only NDVI values or leaf area and can save you a lot of time.
- The *image_processing* library holds all image processing algorithms that may also be useful outside of the use with the PlantAnalyzer and the measurement class.
- *native_stuff* holds some wrapper functions for image arithmetic methods that had to be implemented in C due to performance reasons. (So bascially all the jobs, which I didn't get properly vectorized.)
- *leds.py* provides some convenience functions to control the High power LEDs of the PlantAnalyzer
- The *ConfigParser* module provides the *config* object that reads the whole project configuration from `analyzer.cfg`

1.1 The *measurement* class

class `scripts.measurement.`**`measurement`**(*name*)

a class to hold all the informations about a measurement and process them

member functions:

look at the source code, or call *help(measurement)*

variables:

`imRGB`: RGB image as a opencv-matrix (numpy array)

`imRed`: self explanatory

`imIR`: ———“”———

`imNDVI`: ———“”———

`imRG` ———-“”———

`imRight`: image from the right camera. Used for the depth map

`Disp`: image of the disparity map

`NDVI_float`: a huge array with floats made to fool Biologists that they have super precise data (which they don't!)

`rg`: heat map of the red-green rations of the imRGB

`rg_float` red-green ratios of imRGB

`RGBFilename` Filename, where the RGB image is stored on the hard drive

`RedFilename` self explanatory....

`IRFilename` self explanatory

`RightFilename` ———“”———-

`DispFilename` Filename of the disparity map

`RGFilename` Filename of the RG Ratio heatmap (.jpg)
`NDVIFilename` Filename of the NDVI heatmap (.jpg)
a lot of Numbers I don't know about yet
`leafArea` An estimate of the total leaf area
`averageNDVI` average NDVI value of the leaves.
`averageRG` average Ratio of the red and green reflectances of the leaves

analyze(*statusbar_printer=None*)
This does the whole analyzation process. So it first deflickers the images, then undistorts them and after that calculates the NDVI values, red gree ratios and the disparity map.

Parameters **statusbar_printer** – a function that prints text to a statusbar. If handed none, the standard print command will be used

Return type None

calculateNumbers()
Calculates an estimate of the total leaf area of the plant using the disparity map and the mask.

computeDisparity()
Computes the disparity map using the parameters saved in `analyzer.cfg` and sets `self.disparity`. Returns none. Writes the disparity map also on the hard drive

maskLeaves()
creates a bitmask which is 255 where leaves are, and 0 everywhere else self.leafMask hast the same dimensions as all other images

open(*filename='mess.zip'*)
Loads a zip file as it is created by self.save and relocates all images to the right positions

Parameters **filename** – The name of the zip-file, that should be opened.

Return type None

save(*filename='gurkensalat.zip'*)
saves the images and all the data from the analyzation to a Zip file containing the images and a .txt-file with the analyzed values.

Parameters **filename** – The name of the zip file. If it has no filename extension .zip will be automatically added.

Return type None

takePhotos(*statusbar_printer=None*)
Takes all four photos and saves them to /home/pi/images/<name>.jpg

Parameters **statusbar_printer** – a function that prints text to a statusbar. If handed none, the standard print command will be used

Return type None

1.2 The *image_processing* library

This module is a collection of the most important image processing functions needed by the `measurement` class.

`scripts.image_processing.`**alignImages**(*im1*, *im2*, *showMatches=False*, *threshold=0.5*, *resizefactor=0.5*, *append_text_to_statusbar=None*)
expects two images, assumes that both of them are grayscale of both them are RGB. Returns an aligned version of im2 in the size of im1

Parameters

- **im1** – Any opencv image
- **im2** – OpenCV image with the same color space
- **showMatches** – If `True`, the matches are shown. Default = False
- **threshold** – Parameter, how good the matches must be to be considered. Default = 0.5 If too low, not enough matches might be found. If too high the alignment may fail :param resizefactor: The factor, by which the images are resized for the sift-detector. Too high -> too much RAM usage. Too low -> not accurate enough

Return type opencv image/numpy array with the same dimensions as `im2`

`scripts.image_processing.`**`calculateDisparityMap`**(*imR*, *imL*)
calculates the disparity map of imR and imL using the SGBM algorithm with the parameters supplied in the config parser thing config. *The images must be properly rotated and undistorted*

Parameters

- **imR** – The image taken by the right camera
- **imL** – The image taken by the left camera

Return type `disparity` is a grayscale opencv image with the same dimensions as imL and aligned to it (not to imR).

`scripts.image_processing.`**`calculateNDVI`**(*rgb*, *ir*, *grayscale=False*)
calculates the NDVI-values from the images rgb and IR

Parameters

- **rgb** – The RGB-Image which should be used for the NDVI-Calculation
- **ir** – The IR-Image which sould be used for the NDVI-Calculation
- **grayscale** – specifies, whether *rgb* and *ir* are grayscale images (only one color channel) or rgb images (three color channels). *Make sure, that both images have the same number of color channels*

Return type `(ndvi, ndvi_float)`: ndvi is a heatmap image of the ndvi-values. ndvi_float is a float-array with the same dimensions as ndvi containing the raw NADVI-Values

`scripts.image_processing.`**`calculateRGRatio`**(*im*)
calculates the Ratio of the red and green channel

Parameters **im** – Obviously the image whose channels get divided, duh... Must be BGR. (3 Color channels)

Return type `(rg, rg_float)`: rg is a heatmap image of the r/g ratios. ndvi_float is a float-array with the same dimensions containing the raw values for lookup in the GUI

`scripts.image_processing.`**`cropFrame`**(*im*, *framesize=0.1*)
crops the image by a constant percentage on each border.

Parameters

- **im** – the input image
- **framesize** – is the relative size of the frame, that needs to be cropped away. e.g. 0.1 equals a frame 10% of the size of image

Return type opencv image/numpy array

`scripts.image_processing.`**`deflickerImage`**(*im*, *column*)
Attempts to get rid of the flickering bars caused by the LEDs. To do so it tries to straighten out the brightness values on a supposedly uniformly lit calibration bar at the edge of the image.

Parameters

- **im** – The image whose flickering bars you want to remove. Can be a grayscale image or RGB. In case of grayscale, every channel is deflickered independently

- **column** – The number of the column, where the calibration bar is located in the image *im*

Return type `im_corrected` A hopefully deflickered version of *im*.

`scripts.image_processing.`**`floatIm2RGB`**(*floatIm*)
converts a float Image to a normalized RGB image with the maxmimum possible dynamic range. Should work with both, grayscale and RGB images

Parameters **floatIm** – Literally any numpy array, but presumably a image that exceeds the normal limitations of a opencv image

Return type `rescaled` is a rescaled version of the input, having the maximum possible dynamic range. (in a uint8)

`scripts.image_processing.`**`undistortStereoPair`**(*imR*, *imL*)
undistorts a pair of Stereo images based on the paramers given in paramFile. The distortion parameters are supplied by the config object and stored in ../data/stereoParams.npz. This file must be created using the *calibrate_stereo_cameras* Tool

Parameters

- **imR** – is the right image
- **imL** – is the left image

Return type `(undistR, undistL)` **:** The undistorted versions of `imR` and `imL`

1.3 The *native_stuff*

native_stuff.pyx

Contains all Python Wrappers for the functions that had to be implemented in native C-Code.

Note: If the C-Code changes, the whole module must be recompiled using `setup.py build_ext --inplace` with thes setup.py script in the `scripts` folder

1.4 The *leds.py* library

This library provides basic functions to turn the LED-lights on and off. Please be careful, not to turn the LEDs on for too long, as the transistors might overheat!

`scripts.leds.`**`initLEDs`**()
Starts the PWM Processes for the LEDs.

Warning: **Must** be called before using the LEDs

`scripts.leds.`**`setIR`**(*brightness*)
sets The IR LED to the PWM value `brightness`

Parameters **brightness** – The PWM value. Can range from 0 (off) to 100 (completely turned on)

Return type None

`scripts.leds.`**`setRed`**(*brightness*)
sets The Red LED to the PWM value `brightness`

Parameters **brightness** – The PWM value. Can range from 0 (off) to 100 (completely turned on)

Return type None

scripts.leds.**setWhite**(*brightness*)
sets The White LED to the PWM value brightness

Parameters **brightness** – The PWM value. Can range from 0 (off) to 100 (completely turned on)

Return type None

1.5 The *ConfigParser*

Provides a ConfigParser object which provides all the settings for the PlantAnalyzer from the config-file analyzer.cfg.

This ConfigParser object can be simultaneously used in different modules

CHAPTER

TWO

AUXILIARY TOOLS

2.1 The *calibrate_stereo_cameras.py* Tool

2.1.1 Purpose

The `calibrate_stereo_cameras.py` script is used to obtain the Camera Matrices, Distortion Coefficients as well as the rotation matrix between the cameras and the translation vector between them. It calculates them using the image pairs of checkerboards provided in the `images` file and saves the parameters as a `.npz`-file. These parameters are needed to undistort the images for creating a disparity/depth map. *You only need to run this, if the camera configuration has changed.* If it did change, take the new image pairs using e.g. the `doublePhoto.py` script (at least 10 pairs in different perspectives, compare to the pairs currently saved in `data/calibration\_images`), run the script and move the resulting `.npz`-file to the right location.

2.1.2 Usage

```
usage: calibrate_stereo_cameras.py [-h] [--images IMAGES] [--outfile OUTFILE]
                                   [--width WIDTH] [--height HEIGHT]

calibrate the stereo cameras with images of checkerboards

optional arguments:
  -h, --help            show this help message and exit
  --images IMAGES, -i IMAGES
                        a csv-list with the image pairs to be used for
                        calibration. Each line should contain two filenames,
                        first the right image, then the left image
  --outfile OUTFILE, -o OUTFILE
                        where the calculated undistortion parameters should be
                        stored
  --width WIDTH, -w WIDTH
                        widht of the checkerboard
  --height HEIGHT, -he HEIGHT
                        height of the checkerboard
```

2.2 The *takePhoto.py* Tool

2.2.1 Purpose

A wrapper script for the picamera. Basically the same purpuse as `raspistill` program and thus is actually redundant, but still needed by he PlantAnalyzer to take the pictures.

2.2.2 Usage

```
usage: takePhoto.py [-h] [--filename FILENAME] [--shutterspeed SHUTTERSPEED]
                    [--saturation SATURATION] [--sharpness SHARPNESS]
                    [--iso ISO] [--hflip] [--vflip] [--rotation ROTATION]
                    [--red RED] [--blue BLUE] [--contrast CONTRAST]
                    [--brightness BRIGHTNESS]

Take photo and save it to the specified filename

optional arguments:
  -h, --help            show this help message and exit
  --filename FILENAME, -f FILENAME
                        specify the filename, where the pictures is saved.
                        Default is the current timestamp
  --shutterspeed SHUTTERSPEED, -ss SHUTTERSPEED
                        set the shutterspeed to -ss ms
  --saturation SATURATION, -sa SATURATION
                        set image Saturation (-100 to 100)
  --sharpness SHARPNESS, -sh SHARPNESS
                        set image sharpness (-100 to 100)
  --iso ISO, -i ISO     set the ISO (0=auto,100,200,320,400,500,...)
  --hflip               if set the image is flipped horizontally
  --vflip               if set the image is flipped vertivally
  --rotation ROTATION, -r ROTATION
                        set roation. Allowed values are 0,90 , 180, 270
  --red RED             set red gain for the AWB. If this AND --blue aren't
                        set, automatic white balance will be used
  --blue BLUE           set blue gain for the AWB
  --contrast CONTRAST, -c CONTRAST
                        set the contrast value. (-100 to 100)
  --brightness BRIGHTNESS, -b BRIGHTNESS
                        Set the brightness. (0 to 100)
```

2.3 The *doublePhoto.py* Tool

2.3.1 Purpose

This is a simple script two take a image pair with both cameras. This is useful e.g. to create the image pairs of the checkerboard needed by `calibrate_stereo_cameras.py` or for testing of the disparity map functions. It assumes, that the IPs in `/etc/hosts` are set correctly and that the script `~/bin/takePhoto.py` is installed correctly.

2.3.2 Usage

```
usage: doublePhoto.py [-h] [--filename FILENAME]

Take photo and save it to the specified filename

optional arguments:
  -h, --help            show this help message and exit
  --filename FILENAME, -f FILENAME
                        specify the filename, where the pictures is saved.
                        Default is the current timestamp
```

S

E