Installing and Running PebbleCounts

Ben Purinton (purinton@uni-potsdam.de)

November 2018

1 Introduction

This guide will briefly walk you through the installation and running of PebbleCounts at the command line. PebbleCounts is a Python based application for the identification and sizing of gravel from either orthorectified, georeferenced (**UTM projected**) images with known resolution or simple non-orthorectified images taken from directly overhead with the image resolution approximated by the camera parameters and shot height. It is a semi-automated program in that edge detection and k-means segmentation are performed automatically, but the user must interactively hand-click the well outlined pebbles and ignore the bad results. The software is extremely useful for area-by-number pebble counts without painstaking field work or the disruption of the natural environment via gravel removal. If you really want some deeper background on the problem of grain size measurement from imagery and the testing of PebbleCounts, check out the publication accompanying the algorithm: PUBLICATION DOI. And cite it if you use the results in your own work.

Happy clicking!

1.1 Disclaimer

PebbleCounts is a free (released under GNU General Public License v3.0) and open-source application written by a geologist / amateur programmer. If you have problems installing or running the software contact me purinton@uni-potsdam.de and I can help!

2 Overview

PebbleCounts can be summed up in the flow chart shown in **Flowchart for PebbleCounts**. To briefly summarize, PebbleCounts pre-processes the image by allowing the user to subset the full scene, then interactively mask shadows (interstices between grains) and color (for instance sand). Following this, PebbleCounts windows the scene at three different scales with the window size determined by the input resolution and expected maximum grain size provided by the user. This multi-scale approach allows the algorithm to "burrow" through the grain size distribution beginning by removing the largest grains and ending on the smallest, with the medium sizes in between. At each window the algorithm filters the image, detects edges, and employs k-means segmentation to get an approximate cleaned-up mask of potential separate pebbles. The window is then shown with the mask overlain and the user is able to click the **good** looking grains and leave out the **bad** ones (see the below section **Running PebbleCounts** for the example). These grains are then measured via ellipse fitting to retrieve the long- and short-axis and orientation. This process is iterated through each window and the output from the counting is provided as a comma separated value (.csv) file for user manipulation.

2.0.1 Flowchart for PebbleCounts

3 Installation

The first step is downloading the GitHub repository somewhere on your computer. The folder should contain: 1. Three Python scripts: PebbleCounts.py, PCfunctions.py, calculate_camera_resolution.py 2. An `environment.yml` file containing the Python dependencies and a `install_openCV_env_ubuntu18.sh` shell script for creating an openCV environment with conda on Ubuntu 3. A folder `example_data` with two example images one orthorectified and the other raw 4. A folder `docs` containing this manual

3.1 For the Pros

For those familiar with Python, the best way to install PebbleCounts is by simply downloading the GitHub repository, navigating to the PebbleCounts folder at the command line, ensuring all Python dependencies are installed (see the `environment.yml` file) and getting started by skipping ahead to **Command-line Options**:

3.2 For Newbies

For newcomers to Python, no worries! Installation should be a cinch on most machines and I'll describe it here for Windows. First of all you'll want the Miniconda Python package manager to setup a new Python environment for running the algorithm (see this good article on Python package management).

Download either the 32- or 64-bit installer of Python 3.x then follow the installation instructions. It's recommend to add Miniconda to the system `PATH` variable when prompted. PebbleCounts has a number of important dependencies including gdal for geo-referenced raster manipulation, openCV for image manipulation and GUI operation, scikit-image for filtering and measuring, scikit-learn for k-means segmentation, along with a number of standard Python libraries including numpy, scipy, matplotlib, and tkinter.

Once you've got `conda` commands installed, you can open a command-line terminal and create a conda environment with:

```
conda create --name pebblecounts python=3.6 opencv \
   scikit-image scikit-learn numpy gdal scipy matplotlib tk
```

Or just use the environment .yml file provided with:

```
conda env create -f environment.yml
```

and once installation is complete (and assuming no errors during the install) activate the new environment to run PebbleCounts by:

```
activate pebblecounts
```

Deactivate the environment to exit anytime by:

```
deactivate
```

3.3 For Mac and Linux Users

Those using Mac OS or Linux shouldn't have much trouble modifying the above commands slightly (just add a leading `source` to the `activate` and `deactivate` commands above). Note that installing openCV and getting it to function properly can be a pain sometimes, especially in the case of Linux. In that case it is recommended to find some instructions for installing openCV's Python API for your specific Linux operating system online. The shell script `install_openCV_env_ubuntu18.sh` should allow for a clean install of an openCV inclusive `pebblecounts` conda environment on an Ubuntu v.18 system.

4 Command-line Options

Great you've got it installed! Hopefully that is, we're about to find out! The first step to running the software is navigating to the directory where the three scripts live. On Windows that might look like:

```
cd C:\Users\YourName\PebbleCounts
```

Just replace everything after `cd` with the path on your computer to the downloaded `PebbleCounts` folder.

4.1 Calculate Camera Resolution

First off, if the imagery you intend to use is not orthorectified and georeferenced you'll want to calculate the approximate ground resolution of the photos in millimeters per pixel. To do so you can run the script `calculate_camera_resolution.py` at the command line. Parameters to be provided can be listed with `python calculate_camera_resolution.py -h`. Here they all are:

```
usage: calculate_camera_resolution.py [-h] [-focal FOCAL]  \
    [-height HEIGHT]
                                      [-sensorHW SENSORHW  \
                                          [SENSORHW ...]]
                                      [-imageHW IMAGEHW  \
                                          [IMAGEHW ...]]

optional arguments:
  -h, --help            show this help message and exit
  -focal FOCAL          Camera focal length in millimeters
  -height HEIGHT        Photo capture height in meters
  -sensorHW SENSORHW [SENSORHW ...]
                        The height and width of the internal  \
                            camera sensor in
                        millimeters
  -imageHW IMAGEHW [IMAGEHW ...]
                        The height and width of the  \
                            photography in pixels
```

4.1.1 Example Use

Let's say I have a photo that I took from a height of 1.5 meters at a camera focal length of 35 mm. The camera has a sensor size of 15 by 26 mm and was shot at 24 MP resolution, providing a 4000 pixel high by 6000 pixel wide picture. My command would look like this:

```
python calculate_camera_resolution.py -focal 35 -height 1.5  \
    -sensorHW 15 26 -imageHW 4000 6000
```

And the output printed to the screen would be:

```
Focal length 35.00 mm; Shot from 1.50 m; Sensor size (15.00,  \
    26.00) mm; Image size (4000, 6000) pixels:
```

```
The field of view is 0.64 by 1.11 m

approximate (x,y) resolution in mm/pixel = (0.1607, 0.1857)
average resolution in mm/pixel = 0.1732
```

And I could then pass this resolution (`0.1732`) to the `PebbleCounts.py` script.

Note on Shot Height: If you aren't sure exactly what height the image was shot from, use an approximate value. Even for differences of up to 1 m in shot height the ground resolution for most cameras will change by less than 0.2 mm, and thus have a negligible effect on the resulting grain-sizes measured.

4.2 PebbleCounts

The code can be run from the command line with

```
python PebbleCounts.py ...
```

Parameters to be provided can be listed with `python PebbleCounts.py -h`. Here they all are:

```
usage: PebbleCounts.py [-h] [-im IM] [-ortho ORTHO]
                       [-input_resolution INPUT_RESOLUTION]
                       [-lithologies LITHOLOGIES] [-maxGS MAXGS]
                       [-cutoff CUTOFF]
                       [-min_sz_factors MIN_SZ_FACTORS  \
                           [MIN_SZ_FACTORS ...]]
                       [-win_sz_factors WIN_SZ_FACTORS  \
                           [WIN_SZ_FACTORS ...]]
                       [-improvement_ths IMPROVEMENT_THS  \
                           [IMPROVEMENT_THS ...]]
                       [-coordinate_scales COORDINATE_SCALES  \
                           [COORDINATE_SCALES ...]]
                       [-overlaps OVERLAPS [OVERLAPS ...]]
                       [-nl_means_chroma_filts  \
                           NL_MEANS_CHROMA_FILTS  \
                           [NL_MEANS_CHROMA_FILTS ...]]
                       [-bilat_filt_szs BILAT_FILT_SZS  \
                           [BILAT_FILT_SZS ...]]
                       [-tophat_th TOPHAT_TH] [-sobel_th  \
                           SOBEL_TH]
                       [-canny_sig CANNY_SIG] [-resize RESIZE]

optional arguments:
  -h, --help            show this help message and exit
```

```
-im IM                The image to use including the full  \
    path and
                      extension.
-ortho ORTHO          'y' if geo-referenced ortho-image, 'n'  \
    if not. Supply
                      input resolution if 'n'.
-input_resolution INPUT_RESOLUTION
                      If image is not ortho-image, input the  \
                          calculated
                      resolution from  \
                          calculate_camera_resolution.py
-lithologies LITHOLOGIES
                      What is the expected number of  \
                          lithologies with
                      distinct colors? DEFAULT=1
-maxGS MAXGS          Maximum expected longest axis grain  \
    size in meters.
                      DEFAULT=0.3
-cutoff CUTOFF        Cutoff factor in pixels for inclusion  \
    of pebble in
                      final count. DEFAULT=10
-min_sz_factors MIN_SZ_FACTORS [MIN_SZ_FACTORS ...]
                      Factors to multiply cutoff value by at  \
                          each scale.
                      Used to clean-up the masks for easier  \
                          clicking. The
                      default values are good for ~1  \
                          mm/pixel imagery but
                      should be doubled for sub-millimeter  \
                          or halved for
                      centimeter resolution imagery.  \
                          DEFAULT=[100, 10, 2]
-win_sz_factors WIN_SZ_FACTORS [WIN_SZ_FACTORS ...]
                      Factors to multiply maximum grain-size  \
                          (in pixels) by
                      at each scale. The default values are  \
                          good for
                      millimeter and sub-millimeter imagery,  \
                          but should be
                      doubled for coarser centimeter  \
                          imagery. DEFAULT=[10,
                      2, 0.5]
-improvement_ths IMPROVEMENT_THS [IMPROVEMENT_THS ...]
                      Improvement threshold values for each  \
                          window scale
                      that tells k-means when to halt.  \
                          DEFAULT=[0.01, 0.1,
                      0.1]
-coordinate_scales COORDINATE_SCALES [COORDINATE_SCALES ...]
                      Fraction to scale X/Y coordinates by  \
                          in k-means.
                      DEFAULT=[0.5, 0.5, 0.5]
-overlaps OVERLAPS [OVERLAPS ...]
                      Fraction of overlap between windows at  \
                          the different
                      scales. DEFAULT=[0.5, 0.3, 0.1]
-nl_means_chroma_filts NL_MEANS_CHROMA_FILTS  \
    [NL_MEANS_CHROMA_FILTS ...]
                      Nonlocal means chromaticity filtering  \
                          strength for the
                      different scales. DEFAULT=[3, 2, 1]
-bilat_filt_szs BILAT_FILT_SZS [BILAT_FILT_SZS ...]
                      Size of bilateral filtering windows  \
                          for the different
                      scales. DEFAULT=[9, 5, 3]
-tophat_th TOPHAT_TH  Top percentile threshold to take from  \
    tophat filter
                      for edge detection. DEFAULT=90
-sobel_th SOBEL_TH    Top percentile threshold to take from  \
    sobel filter for
                      edge detection. DEFAULT=90
-canny_sig CANNY_SIG  Canny filtering sigma value for edge  \
    detection.
                      DEFAULT=2
-resize RESIZE        Value to resize windows by should be  \
    between 0 and 1.
                      DEFAULT=0.8
```

Here's a bit more detail on some of the less obvious inputs to clarify:

- `-resize` controls the pop-up window size for the GUI. If you notice the window is too small to see the grains then use a high value like 0.9, but if the image is partially off-screen you should try lowering the value to around 0.7.
- `-lithologies` is the expected number of different rock types in the image with distinct coloration differences. It defaults to 1, meaning the lithology is uniform or the color differences between lithologies are minimal and therefore difficult to discern from the image alone.
- `-maxGS` is the expected size in meters of the largest rock in the image based on some field knowledge. This value is used during the windowing to set the appropriate sizes at the three scales in conjunction with the `-win_sz_factors` input.

- `-cutoff` is the algorithm's lower limit on b-axis measurement given in pixels. The default value of 10 is what we found to be reliable for accurate distribution measurement using PebbleCounts. This value is also used by the `-min_sz_factors` input to cleanup the mask at each of the three window scales and should also be scaled depending on the imagery resolution (sub-mm, mm, cm).
- `-improvement_ths` is the fractional percentage (from 0-1) that k-means uses to assess convergence and stopping. The default values are probably good here.
- `-coordinate_scales` is the fractional percentage (from 0-1) to scale the x,y coordinates of each pixel compared with the color information in the k-means segmentation. Since we want to allow for anisotropic grains covering large areas if they have semi-uniform color, we want to scale the relative importance of pixel location by approximately 50% of the color, hence the default values of 0.5 at each scale.
- `-nl_means_chroma_filts` is the level of chromaticity filtering to apply during non-local means denoising, which should be reduced at each scale. Higher values lead to more smoothing of the image and a cartoonish appearance. The default values should again be good here.
- `-bilat_filt_szs` is the square window size to apply for bilateral filtering, with the aim of further smoothing the image while preserving interstices between the grains. The size of this filter window should be reduced with the windowing scale. The default values are also good here.
- `-tophat_th`, `-sobel_th`, and `-canny_sig` are the tophat filter percentile threshold, Sobel filter percentile threshold, and Canny edge detection smoothing standard deviation. These are the values used on edge detection from the gray-scale image and are probably good at the default value. The same value is used for each scale.

5 Running PebbleCounts

Now you're ready to run an image. Because PebbleCounts doesn't allow you to save work in the middle of clicking it's recommended that you don't use images covering areas of more than 2 by 2 meters or so. For higher resolution (sub-mm) imagery it's recommended not to go above 1 by 1 meters. If you want to cover a larger area simply break the image into smaller parts and process each individually, so you can give yourself a break. If at anytime you want to end the application simply press *CTRL + C*.

5.0.1 Note on ortho-imagery

Georeferenced ortho-photos should be in a **UTM projection**, providing the scale in meters. You can use the gdal command line utilities to translate rasters between various projections

5.1 Step-by-step

1. Depending on whether you're going to use an ortho or non-ortho image run one of the following commands:

- **Ortho:** Be sure to set the `-ortho` flag to `y` and the resolution will be automatically read by gdal:

```
python PebbleCounts.py -im  \
    C:\Users\YourName\PebbleCounts\example_data\ortho_resolution_1.2mmPerPix.tif  \
    \
                -ortho y -lithologies 1 -maxGS 0.2 -cutoff 10  \
                    -min_sz_factors 100 10 2 \
                -win_sz_factors 10 2 0.5 -improvement_ths 0.01  \
                    0.1 0.1 -coordinate_scales 0.5 0.5 0.5 \
                -overlaps 0.5 0.3 0.1 -nl_means_chroma_filts 3  \
                    2 1 -bilat_filt_szs 9 5 3 \
                -tophat_th 90 -sobel_th 90 -canny_sig 2  \
                    -resize 0.8
```

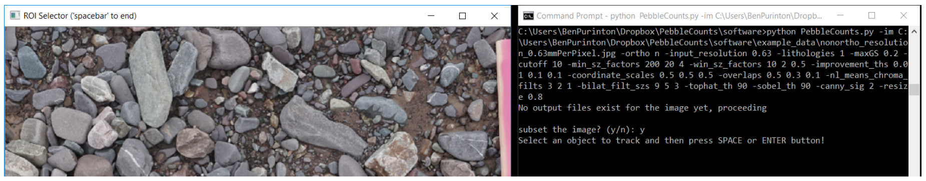
I've written out the full command here, including all options, even though I'm using all the default values for this ~1 mm/pixel ortho image.

- **Non-ortho Imagery:** Be sure to set the `-ortho` flag to `n` and also provide the `-input_resolution` in mm/pixel, which can be found as in the above section **Calculate Camera Resolution**:

```
python PebbleCounts.py -im  \
    C:\Users\YourName\PebbleCounts\software\example_data\nonortho_resolution_0.63mmPerPixel.
    \
                -ortho n -input_resolution 0.63 -lithologies 1  \
                    -maxGS 0.2 -cutoff 10 -min_sz_factors 200  \
                    20 4 \
                -win_sz_factors 10 2 0.5 -improvement_ths 0.01  \
                    0.1 0.1 -coordinate_scales 0.5 0.5 0.5 \
                -overlaps 0.5 0.3 0.1 -nl_means_chroma_filts 3  \
                    2 1 -bilat_filt_szs 9 5 3 -tophat_th 90 \
                -sobel_th 90 -canny_sig 2 -resize 0.8
```

Note: I've changed some of the default values for `-min_sz_factors` (doubled the default), since the resolution of this imagery is sub-mm.

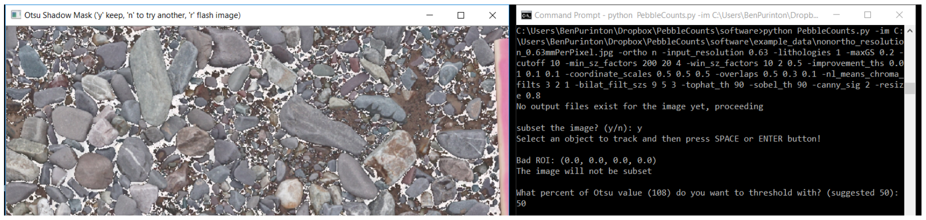
2. Interactively subset the image by typing `y` or don't by typing `n`. If you do subset, click and drag a box on the pop-up window and press the *spacebar* to close the window again.



```
C:\Users\BenPurinton\Dropbox\PebbleCounts\software>python PebbleCounts.py -im C:
\Users\BenPurinton\Dropbox\PebbleCounts\software\example_data\nonortho_resolutio
n_0.63mmPerPixel.jpg -ortho n -input_resolution 0.63 -lithologies 1 -maxGS 0.2 -
cutoff 10 -min_sz_factors 200 20 4 -win_sz_factors 10 2 0.5 -improvement_ths 0.0
1 0.1 0.1 -coordinate_scales 0.5 0.5 0.5 -overlaps 0.5 0.3 0.1 -nl_means_chroma_
filts 3 2 1 -bilat_filt_szs 9 5 3 -tophat_th 90 -sobel_th 90 -canny_sig 2 -resiz
e 0.8
No output files exist for the image yet, proceeding

subset the image? (y/n): y
Select an object to track and then press SPACE or ENTER button!
```

Interactive subsetting of full image. Click and drag a rectangle then press the *spacebar*.

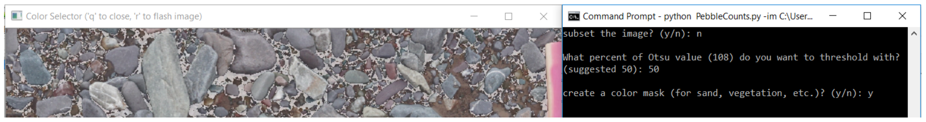
3. Input a percentage (0-100) of the Otsu shadow threshold value, then press enter. This will open a pop-up window displaying the image with the Otsu mask in white. On the keyboard press *r* to flash the original un-masked image, *y* to accept the mask and move on, and *n* to close the window and enter a new value.


Otsu Shadow Mask ('y' keep, 'n' to try another, 'r' flash image)

```
C:\Users\BenPurinton\Dropbox\PebbleCounts\software>python PebbleCounts.py -im C:
\Users\BenPurinton\Dropbox\PebbleCounts\software\example_data\nonortho_resolutio
n_0.63mmPerPixel.jpg -ortho n -input_resolution 0.63 -lithologies 1 -maxGS 0.2 -
cutoff 10 -min_sz_factors 200 20 4 -win_sz_factors 10 2 0.5 -improvement_ths 0.0
1 0.1 0.1 -coordinate_scales 0.5 0.5 0.5 -overlaps 0.5 0.3 0.1 -nl_means_chroma_
filts 3 2 1 -bilat_filt_szs 9 5 3 -tophat_th 90 -sobel_th 90 -canny_sig 2 -resiz
e 0.8
No output files exist for the image yet, proceeding

subset the image? (y/n): y
Select an object to track and then press SPACE or ENTER button!

Bad ROI: (0.0, 0.0, 0.0, 0.0)
The image will not be subset

What percent of Otsu value (108) do you want to threshold with? (suggested 50):
50
```

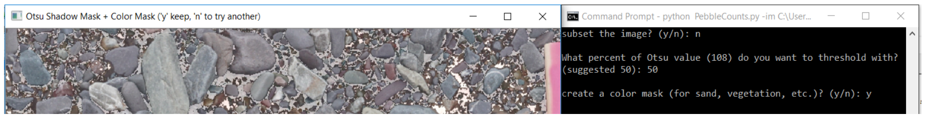
Otsu thresholding of the image with an entered value 0-100. Press *r* to flash the original image, *y* to accept the mask, or *n* to try a different value.

4. Is there a color you want to mask out in the scene? Maybe the sand is a uniform color distinct from the pebbles. If so, then in the next step enter `y`, which will bring up another pop-up window. With the window active, you can press *q* to close it if you decide not to color mask and *r* to flash the original image. Once you click a point in the window with a color you'd like to mask a second pop-up will open displaying the result of applying a mask to this color. Press *y* to accept the mask or *n* to close it and try another click in the first window. Pressing *y* here will return you to the command prompt where you can finish color masking by entering `n` or adding additional color masks by entering `y`.



```
subset the image? (y/n): n

What percent of Otsu value (108) do you want to threshold with?
(suggested 50): 50

create a color mask (for sand, vegetation, etc.)? (y/n): y
```

Color masking clicking window. Click on a color you want to mask to open a second window and check it. Press *q* to close window or *r* to flash the original image.


Otsu Shadow Mask + Color Mask ('y' keep, 'n' to try another)

```
subset the image? (y/n): n

What percent of Otsu value (108) do you want to threshold with?
(suggested 50): 50

create a color mask (for sand, vegetation, etc.)? (y/n): y
```

Color masking result window. Press *y* to accept or *n* to try a different click in the previous window.

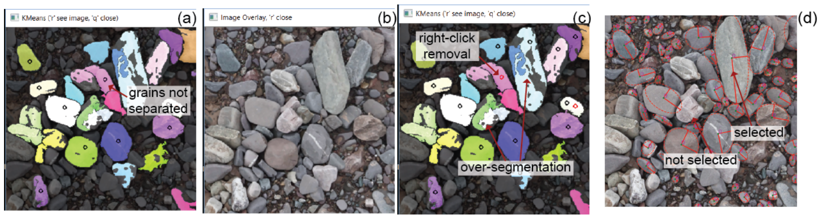
5. After these couple interactive steps, PebbleCounts will take over the automated windowing, filtering, edge detection, and k-means segmentation at each window. The command prompt should look something like this: “‘ Beginning k-means segmentation

   Scale 1 of 3

   Window 1 of 1

   Non-local means filtering Bilateral filtering Black tophat edge detection Canny edge detection Sobel edge detection Running k-means Current number of clusters: 2, total inertia: 59896.391 Current number of clusters: 3, total inertia: 48804.694 Current number of clusters: 4, total inertia: 39567.055 Current number of clusters: 5, total inertia: 34193.667 Current number of clusters: 6, total inertia: 29956.034 Current number of clusters: 7, total inertia: 25652.019 Current number of clusters: 8, total inertia: 23202.660 Current number of clusters: 9, total inertia: 22270.410 Current number of clusters: 10, total inertia: 21233.908 Current number of clusters: 11, total inertia: 19145.581 Current number of clusters: 12, total inertia: 18094.143 Current number of clusters: 13, total inertia: 18043.373 Cleaning up k-means mask “‘

6. After the mask is cleaned a new window will open where you need to click the good looking grains and ignore the bad ones. Left clicking anywhere on the image will produce a black circle at that point, meaning that you’ve selected all the pixels in this connected region as one grain. A right click anywhere on the image will remove the last click and exchange the black circle for a red one, indicating this area will not be considered (unless of course you add another left click to the region). Overlay the original image to help decide what is and is not a well delineated grain by pressing *r* once to open the image and *r* again to close the image and return to the mask. Once you are satisfied with your clicks press *q* to close the window and automatically move on to the next window and/or scale. The clicked grains will be automatically measured and added to the final output.



(a) Interactive k-means mask clicking window. A left click adds a pebble region and black circle. Pressing *q* will close the image and continue segmentation on the next window. (b) Pressing *r* opens the original image to check the mask against, *r* again to close the original image. (c) Right click anywhere on the image to remove the last clicked point and replace the black circle with a red one. (d)

Shows the final ellipses fit to each of the clicked regions. Numbering in the ellipse corresponds to the potential lithologic unit.

7. Repeat the clicking on each window that pops up (see the command window for what number window out of the full number you are on). With a little practice this will go quickly. After the windows are done the results will be saved out and you can repeat from step 1 with another image.

5.1.1 An important note on clicking!

As shown in the previous figure, PebbleCounts does not provide a perfect segmentation. Two errors you will commonly note are: 1. Under-segmentation of overlapping grains. Avoid clicking these regions or the resulting ellipse will be fit to many grains. 2. Over-segmentation of single grains. Here it is up to the user to decide which part of the segmented grain (if any) to select. If the mask covers the majority of the grain despite some holes or shrinkage, then it is advisable to select the grain, since the final ellipse will be fit to the full region covered. Even if the center of a grain is entirely missing from the mask, if the ends of the grain are in the same mask then the fit ellipse will approximate the grain well.

5.2 Ouput

PebbleCounts saves out a few outputs in the same folder that the image resides: * csv: filename_PebbleCounts_CSV.csv * label image (geo-referenced if ortho image): filename_PebbleCounts_LABELS.tif * figure showing the fit ellipses and potential rock type: filename_PebbleCounts_FIGURE.png

The results .csv has a line for each grain showing the fraction of the scene not measured (combined background shadow, color masked area, and unmeasured grains) the fraction of the scene that was selected by the color mask as background color (sand perhaps) and each grains' characteristics including a- and b-axis of the fit ellipse in pixels and in meters, the area covered by the grain mask in pixels and square meters, the orientation of the fit ellipse measured from -pi/2 to pi/2 relative to the positive x-axis (orientation=0) in cartesian coordinates. If the input imagery is georeferenced the UTM Northing (Y) and Easting (X) coordinates of the pebble's centroid are be provided:

```
perc. not meas.,perc. background color,a (px),b (px),a (m),b  \
    (m),area (px),area (m2),orientation,lithology
0.17551331897681643,0,111.9574708212346,102.05260381916898,0.0705332066173778,0.064293140406
0.17551331897681643,0,262.65677592565635,91.77581426816505,0.1654737688331635,0.057818762988
0.17551331897681643,0,139.03899178134733,62.223301209895446,0.08759456482224882,0.0392006797
```