**Contents**

[USAGE 2](#_Toc12877093)

[How to start the application 2](#_Toc12877094)

[Application modes 2](#_Toc12877095)

[Settings 3](#_Toc12877096)

[Config files options description: local.py 3](#_Toc12877097)

[Config files options description: config\_local.json 8](#_Toc12877098)

[ARCHITECTURE DESCRIPTION 13](#_Toc12877099)

[Common logic and class/function diagram 13](#_Toc12877100)

[Functions description 15](#_Toc12877101)

USAGE

How to start the application

1. Clone Github repository by the following command**\***:

*git clone* [*https://github.com/natuition/histogram*](https://github.com/natuition/histogram)

1. Open repository root folder by the following command:

*cd histogram*

1. Switch to dev branch (this step is required until full code testing and release):

*git checkout dev*

1. Make settings files from examples and set the desired mode (see “Application modes” and “Settings” sections in this document)
2. Place image to process into the ‘input’ directory (located in the root of repository), and set image name in the config/local.py settings file (see more in the settings section)
3. Run the script:

*python3 histogram.py*

**\*** If you already have downloaded repository and have no uncommitted changes – you can simply type:

*git pull*

to get the last updates and be ready for start (you still need to check if configuration is set as you need).

Application modes

There are two modes now: “database” mode, when app is processing input image (from image file or camera, you can choose source in settings), makes image patterns from area or whole image for future comparison and creates database from them. The second mode is called “searching”, in this mode app loads one (for now) image or takes one from the Raspberry Pi camera, and looks for the most “unlike” part in image (called fragment) comparing to patterns in database, calculates center coordinates for that fragment, and if smoothie board control enabled (see settings) converts this coordinates into a g-code and sends to the smoothie board for execution. Also writes debug information (including image, fragment and fragment center) into the output directory (it is located in the same directory as histogram.py script).

You can set application mode in config/local.py file. Application will show message about wrong mode if there incorrect mode was set.

Settings

There are two settings files which are used by application: “local.py” and “config\_local.json”. Both are absent in repository and ignored by version control system. This makes you to keep your own settings after updating code from Github repository and always have examples which are defaults too. For starting first time you need to rename or copy and rename (preferred) “local - example.py” and “config\_local - example.json” files (located in the config folder), and save with names “local.py” and “config\_local.json” respectively.

After that you can set your own configurations in “local.py” and “config\_local.json” files. See each option description below.

Config files options description: local.py

Paths section

Pay attention that different python versions require different directory separation symbols:

Example for python 3.5 (RPi): dir/subdir/filename.extension

Example for python 3.7 (PC): dir\\subdir\\filename.extension

**config["hist\_database\_path"] = "database\\database.npy"**

Path where script will search for database file. DB file always should be with .npy extension.

**config["patterns\_dataset\_dir"] = "database\\images"**

Path where script will put and search for patterns which are used for database making.

**config["query\_image\_path"] = "input\\prise8.jpg"**

Path to input image. This image will be used if camera disabled.

**config["output\_image\_dir"] = "output\\"**

Path to directory where script will put debug information or results.

**config["output\_image\_name"] = "output"**

Suffix name of the output image file.

**config["output\_image\_extension"] = ".jpg"**

Output image file extension. Should be “.jpg”.

Image processing section

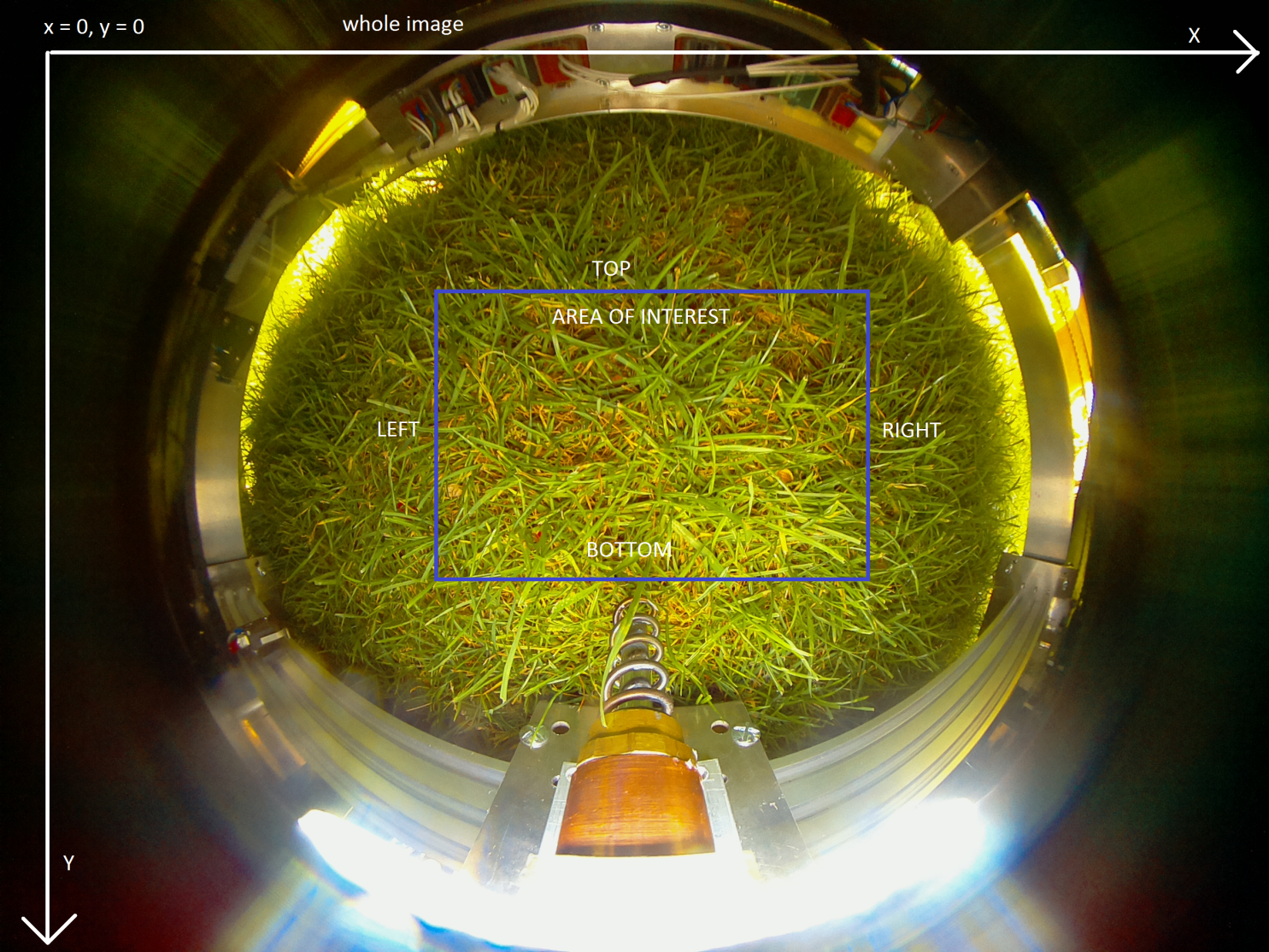
**config["aoi\_top\_border"] = 595**

**config["aoi\_bottom\_border"] = 1200**

**config["aoi\_left\_border"] = 854**

**config["aoi\_right\_border"] = 1734**

AOI - area of interest. Determines input image area to work with (where to search for weeds or which area is can be used for patterns making), can be less or equal to the size of the input image. Each value is a pixel from where starts or ends area. Whole image will be processed if values equal to the size of the image.



**config["fragment\_w"] = 120**

**config["fragment\_h"] = 120**

When searching or making database, input image’s AOI is divided for separate fragments, then each fragment is analyzed. These options determine how large the fragments should be. Fragments shouldn’t be greater than AOI or errors will be occurred.

**config["fragment\_x\_offset"] = 60**

**config["fragment\_y\_offset"] = 60**

These options determine mask offsets between current image part to analyze (which is fragment) and the next one. By default offset values is half of the fragment sizes respectively. See details in algorithm descriptions below.

**config[“hist\_channels”] = [0, 1, 2]**

This option defines color channels count that are used when making histogram. Changing this option requires database recreation.

**config[“hist\_size”] = (8, 12, 3)**

Count of bins. More bins means more sensitivity and size of the histogram. See details [here](https://docs.opencv.org/3.4/d8/dbc/tutorial_histogram_calculation.html), [here](https://www.pyimagesearch.com/2014/12/01/complete-guide-building-image-search-engine-python-opencv/) and [here](https://en.wikipedia.org/wiki/Histogram). Changing this option requires database recreation.

**config["hist\_range"] = [0, 180, 0, 256, 0, 256]**

Range of color values that used in histogram. See details [here](https://docs.opencv.org/3.4/d8/dbc/tutorial_histogram_calculation.html). Changing this option requires database recreation.

**config["hist\_comp\_method"] = cv.HISTCMP\_CHISQR**

Defines which method should be used for histograms comparison. See available methods [here](https://docs.opencv.org/3.4/d6/dc7/group__imgproc__hist.html#ga994f53817d621e2e4228fc646342d386).

**config["lesser\_dist\_more\_similar"] = True**

Histogram comparison functions returns some numeric value called distance. This option defines is bigger distance means more differences between two images or not. This option’s value depends on histogram comparison method option, and should be set correctly for proper script working. Set to “True” if lesser distance means more similar images. Set to “False” if lesser distance means more unlike images.

App config

**config["app\_mode"] = "searching"**

Defines script mode. Currently there are two modes:

"database" - loads image and adds fragments from AOI to patterns and histogram database. Image source depends on use\_camera setting.

"searching" - loads image and searches most unlike grass fragment and tries to extract this plant. Image source depends on use\_camera setting.

**config["use\_camera"] = False**

Defines image source. Set to True to take image from camera, set to False to load image from memory (proper path should be defined in paths section). This option can be enabled when running on RPi, otherwise you’ll get an error.

**config["camera\_w"] = 2592**

**config["camera\_h"] = 1944**

This option defines camera resolution. See possible resolutions for your camera [here](https://picamera.readthedocs.io/en/release-1.13/fov.html#sensor-modes). Camera resolution changings means image size changes, this affects AOI area coordinates (see above). Also check if corkscrew pixel coordinates was changed.

**config["camera\_framerate"] = 32**

This option defines how many images per second camera will try to get, has no effect as script takes only single image, just leave this value by default.

**config["cork\_center\_x"] = 1292**

**config["cork\_center\_y"] = 1172**

Pixel coordinates of the point where the center of the extractor on the input image. This option should be set properly and be updated every time image size was changed or corkscrew will go to the wrong coordinates.

**config["one\_mm\_in\_px"] = 4**

This option defines mm:pixel ratio on the input image. This option is important for smoothie g-code coordinates calculation. Example: for image, where 1 mm of real object takes 4 pixels in image value should be 4.

Smoothie settings

**config["smoothie\_host"] = "169.254.232.224"**

IP address of Smoothie board to connect.

**config["smoothie\_xy\_F"] = 1000**

Defines force value for moving corkscrew on X and Y axes.

**config["smoothie\_z\_F"] = 1000**

Defines force value for extraction (Z axis).

**config["extraction\_z"] = 100**

Defines how deep corkscrew should go into the ground when extracting (Z axis).

Config files options description: config\_local.json

**"CONFIG\_VERSION": 0.1,**

Config file version may be used inside the scripts, this option shouldn’t be edited.

**"X\_MIN": 0,**

Min value for X axis on smoothie board. G-code commands that lead to movement beyond min and max values for axes will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move that hardware cannot do.

**"X\_MAX": 198,**

Max value for X axis on smoothie board. G-code commands that lead to movement beyond min and max values for axes will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move that hardware cannot do.

**"Y\_MIN": 0,**

Min value for Y axis on smoothie board. G-code commands that lead to movement beyond min and max values for axes will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move that hardware cannot do.

**"Y\_MAX": 79,**

Max value for Y axis on smoothie board. G-code commands that lead to movement beyond min and max values for axes will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move that hardware cannot do.

**"Z\_MIN": 0,**

Min value for Z axis on smoothie board. G-code commands that lead to movement beyond min and max values for axes will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move that hardware cannot do.

**"Z\_MAX": 100,**

Max value for Z axis on smoothie board. G-code commands that lead to movement beyond min and max values for axes will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move that hardware cannot do.

**"F\_MIN": 1,**

The minimum value of the force with which the movement can be performed on smoothie. G-code commands that lead to movement with force beyond min and max force values will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move with too high or too low force value.

**"F\_MAX": 1000,**

The maximum value of the force with which the movement can be performed on smoothie. G-code commands that lead to movement with force beyond min and max force values will not be sent to smoothie and executed. This option is used to prevent hardware damage if wrong g-code requires to make a move with too high or too low force value.

**"USE\_X\_AXIS\_CALIBRATION": false,**

It indicates if calibration on X axis is allowed. **Be aware** this option should be set to false if [stoppers](http://smoothieware.org/endstops) are not installed properly on that axis, otherwise hardware can be damaged! True – allow calibration, false – disable calibration. Calibration is used to move corkscrew in starting position and set current coordinates properly.

**"USE\_Y\_AXIS\_CALIBRATION": false,**

It indicates if calibration on Y axis is allowed. **Be aware** this option should be set to false if [stoppers](http://smoothieware.org/endstops) are not installed properly on that axis, otherwise hardware can be damaged! True – allow calibration, false – disable calibration. Calibration is used to move corkscrew in starting position and set current coordinates properly.

**"USE\_Z\_AXIS\_CALIBRATION": false,**

It indicates if calibration on Z axis is allowed. **Be aware** this option should be set to false if [stoppers](http://smoothieware.org/endstops) are not installed properly on that axis, otherwise hardware can be damaged! True – allow calibration, false – disable calibration. Calibration is used to move corkscrew in starting position and set current coordinates properly.

**"X\_AXIS\_CALIBRATION\_TO\_MAX": false,**

Defines on which side of the X axis stopper is installed. Corkscrew should moving in the stopper direction when calibration. Set “true” if stopper installed on min value side, set “false” if stopper installed on max value side. **Be aware:** wrong value may damage your hardware when doing calibration!

**"Y\_AXIS\_CALIBRATION\_TO\_MAX": true,**

Defines on which side of the Y axis stopper is installed. Corkscrew should moving in the stopper direction when calibration. Set “true” if stopper installed on min value side, set “false” if stopper installed on max value side. **Be aware:** wrong value may damage your hardware when doing calibration!

**"Z\_AXIS\_CALIBRATION\_TO\_MAX": true,**

Defines on which side of the Z axis stopper is installed. Corkscrew should moving in the stopper direction when calibration. Set “true” if stopper installed on min value side, set “false” if stopper installed on max value side. **Be aware:** wrong value may damage your hardware when doing calibration!

**"AFTER\_CALIBRATION\_AXIS\_OFFSET": 15,**

Distance by which the corkscrew will be shifted from the stopper after calibration.

**"SMOOTHIE\_HOST": "169.254.232.224",**

IP address of smoothie board. It is legacy option, use local.py to configure smoothie connection.

**"USE\_SMOOTHIE\_CONNECTION\_SIMULATION": true,**

Set to true to allow real connection to smoothie, set to false to use connection simulation. It is required for testing or running code without real smoothie connected.

**"WEB\_SERVER\_HOST": "127.0.0.1",**

Defines IP address of the web server. Legacy option, not used.

**"WEB\_SERVER\_PORT": 8080,**

Defines port of the web server. Legacy option, not used.

**"STREAM\_SERVER\_HOST": "192.168.8.100",**

Defines IP address of the stream server. Legacy option, not used.

**"STREAM\_SERVER\_PORT\_HTTP": 8082,**

Defines port for HTTP requests (from browser when loading page) of the stream server. Legacy option, not used.

**"STREAM\_SERVER\_PORT\_WS": 8084,**

Defines port for web socket connection inside the stream server. Legacy option, not used.

**"TURNING\_MOTOR\_E\_MIN": -20,**

Determines the maximum rotation value of the wheels to the left. Legacy option, not used.

**"TURNING\_MOTOR\_E\_MAX": 20,**

Defines the maximum rotation value of the wheels to the right. Legacy option, not used.

**"TURNING\_MOTOR\_F\_MIN": 1,**

Defines minimum force value for turning wheels. Legacy option, not used.

**"TURNING\_MOTOR\_F\_MAX": 500,**

Defines maximum force value for turning wheels. Legacy option, not used.

**"MOTION\_MOTOR\_E\_MIN": -100,**

**"MOTION\_MOTOR\_E\_MAX": 100,**

Defines navigation motor minimal and maximal step. Commands that lead to overshooting will not be sent to smoothie and executed.

**"MOTION\_MOTOR\_F\_MIN": 1,**

**"MOTION\_MOTOR\_F\_MAX": 500**

Defines navigation motors minimal and maximal force value. Commands that lead to overshooting will not be sent to smoothie and executed.

ARCHITECTURE DESCRIPTION

Common logic and class/function diagram

Pieces of image to process are called fragments. Pieces of image that used as samples with which fragments are compared are called patterns.

**Main steps:**

**In any mode following steps are executed:**

1) Load settings files. Paths are specified in source code and located in config/ directory.

2) Create access-lockable thread-safe containers for corkscrew current coordinates.

3) If using smoothie is allowed - do calibration and go to the center of working area (the center of x and y axes of corkscrew)

4) Get the image. If using camera is allowed in settings - take single image from camera and work with it, otherwise load image by input image path specified in settings.

5) Get all fragments according to fragments settings from the area of interest which coordinates is specified in config file too.

**If running in database making mode:**

6) Save these fragments on hard drive in directory specified in config, use UUID as unique names for them. These fragments will be used as samples (patterns) for future comparison.

7) Try to load histograms database file. Loaded database is stored in RAM. If file is absent - create empty database structure.

8) Generate for all patterns color histograms, and add them to loaded database.

9) Overwrite existing database with updated database.

Creating new database requires removing existing database and all patterns fom the patterns directory. Adding new samples (patterns) to existing database requires temporary remove patterns from patterns directory, or their copies will be added to database too. After update with new samples (patterns) old used patterns should be placed back to the patterns directory. Don't worry about the images names - you won't get any "existing file errors" as each pattern always has unique name regardless on config, computes, OS, etc.

**If running in searching mode:**

6) Get the most unlike fragment and fragment’s data.

7) Get central pixels coordinates of this fragment.

8) Draw this fragment and his center on image copy and save it to debug directory.

Next steps are executed only if smoothie enabled:

9) Convert pixel coordinates into smoothie coordinates.

10) Check if these coordinates goes beyond acceptable range that is specified in config, display error information if so.

11) Generate g-code and send to smoothie to move corkscrew in position (X, Y axes).

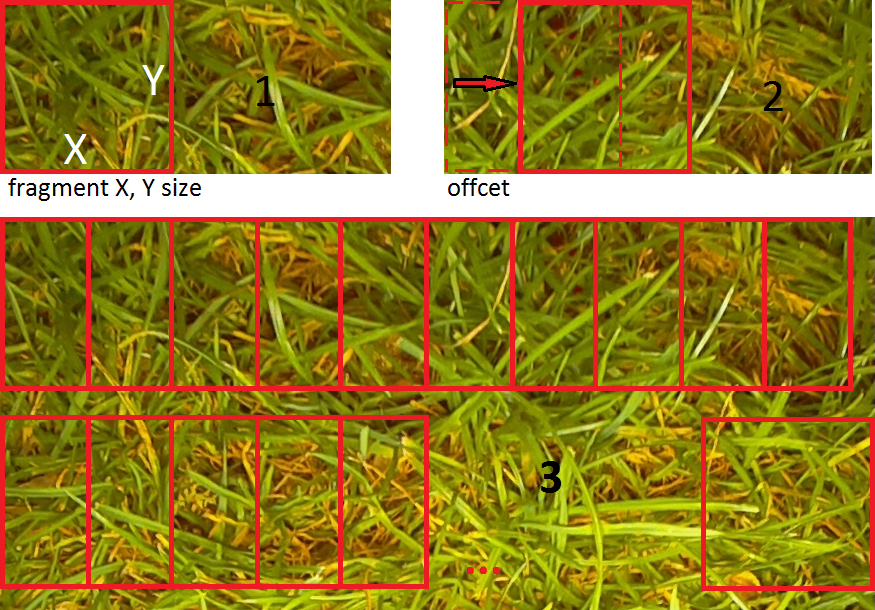
12) Generate g-code and send to smoothie to make extraction, then same actions to lift up corkscrew.

Each smoothie response is displayed on the screen.

**How fragments searching works**

Area of interest (AOI) on the image processed iteratively by moving mask on this image. Each iteration, mask is moved by shift distance, part of the image under mask is called fragment. Mask size and shift distance is specified in configuration and can be changed if needed, but mask size should be lesser than AOI.

If there remains some space on the AOI that lesser than shift distance what means it will not be covered when last possible mask shifting with current offset distance - in that case mask will capture additional fragment from the end to prevent missed parts of image. It is applied to lines and columns, all is calculated automatically.



Functions description

**def dump\_database(database):**

Parameters:

*database* – python list containing database records

Returns:

*None*

Using [pickle](https://docs.python.org/3/library/pickle.html), saves received database on the hard drive in the directory specified in config.py.

**def load\_database():**

Parameters:

*None*

Returns:

*database* - python list containing database records

Using [pickle](https://docs.python.org/3/library/pickle.html), loads and returns database from the hard drive from the directory specified in config.py.

**def save\_frags\_to\_patterns\_dir(fragments):**

Parameters:

*fragments –* python list of image fragments, where each fragment is python dict with structure {"img", "start\_x", "start\_y", "end\_x", "end\_y"}, where "img" is key of the fragment, and the rest keys are containing coordinates of that fragment on image where it from.

Returns:

*None*

This function writes each fragment from received fragments list on hard drive in directory specified in config.py. Each fragment is saved using unique name which is generated unique [UUID](https://docs.python.org/3/library/uuid.html). Unique names are never being repeated regardless anything.

**def add\_patterns\_to\_database(database: list):**

Parameters:

*database* – python list containing database records or empty python list.

Returns:

*database* –python list containing updated database records

Loads every sample image called pattern from directory specified in config.py from hard drive, calculates color histogram according to settings in config.py, creates record and adds this record to the argument database list. Each record is a python dict that has {"path", "hist"} structure, where “path” is a key of full path of the current sample, “hist” is a key of sample’s color histogram. Function always adds to database all samples from samples directory, meaning if some samples in this directory were added earlier – they will be added again. To avoid this keep in directory only samples that should be added when launching function. Returns the argument database to which records were added during the function operation.

**def calc\_hist(image, mask=None):**

Parameters:

*image* – [numpy](https://www.numpy.org/) array which represents image for which color histogram should be created.

*mask* – numpy array mask which is used when making histogram for part of the argument image instead of whole. Default value is python None type (function creates histogram for whole received image). See details about mask [here](https://docs.opencv.org/3.4/d6/dc7/group__imgproc__hist.html#ga4b2b5fd75503ff9e6844cc4dcdaed35d).

Returns:

*histogram* – opencv’s [normalized](https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga87eef7ee3970f86906d69a92cbf064bd) color histogram for received image.

Creates and returns color histogram for argument image according to histogram settings in local.py config file.

**def find\_most\_unlike\_hist(query\_hist, database: list):**

Parameters:

*query\_hist* – OpenCV’s color histogram to compare with histograms in database.

*database* – (python list) which contains records with which will be compared query histogram.

Returns:

*most\_unlike* – python dict, contains information about sample in the database which has most big difference comparing to query histogram.

Function searches record in the argument database with most unlike image sample comparing to the query histogram. Most unlike sample is being found by comparing distances of the query histogram and histogram of each database sample. Comparison algorithm and which distance should be saved (smallest or biggest) are specified in local.py config file.

Returning value is python dict with structure {"path", "dist"}, where “path” is the key of the full path to the resulting sample in the database, “dist” is the key of the distance between query histogram and histogram of the sample in database.

**def find\_most\_unlike\_frag(fragments: list, database: list):**

Parameters:

*fragments* – python list of images which should be compared to database samples. Each list record is python dict with available image by key “img”.

*database* – (python list) which contains records with which will be compared each fragment from the argument fragments list.

Returns:

*most\_unlike\_fragment* – python dict which contains fragment and his coordinates on source image.

*most\_unlike\_value* – python dict with info about returned fragment. Result of find\_most\_unlike\_hist function, see function description for details.

This function returns most unlike fragment of the argument fragments list comparing to each record of argument database. Each fragment is compared to each fragment of the database, the most unlike is the result. See find\_most\_unlike\_hist function for comparison details. As find\_most\_unlike\_hist function this function uses local.py settings file to define lower or greater distance value means more similar images.

Returns fragment which has biggest difference comparing to the one of records of the database.

**def \_frag\_process\_columns(fragments: list, image, cur\_end\_y, last\_col\_processing):**

Parameters:

*fragments* – python list to save found fragments.

*image* – source image to process.

*cur\_end\_y* – end pixel number of current line to process on source image.

*last\_col\_processing* – python bool type flag that indicates if there will be a piece of an image that cannot be covered by mask with current mask shift distance setting.

Returns:

*fragments* – list of found fragments during function operation.

This function is for inner usage. Processes specified single line on the received image in range of AOI that specified in local.py config file. Line end point is specified as *cur\_end\_y* parameter. Line start point is calculated from end point and fragment (mask) size that specified in local.py config file. Returns list of found fragments with their coordinates on the source image. Each fragment is saved into python dict with structure

{

"img" – fragment itself

"start\_x": fragment’s X starting pixel

"start\_y": fragment’s Y starting pixel

"end\_x": fragment’s X end pixel

"end\_y": fragment’s Y end pixel

}

And then added to fragments list. There is no deep copy operation when creating fragments, that means if source image changed – fragments are changed too. Last fragment will be taken from the end of image if there is a piece of image that lesser than mask shift distance (see details in architecture description – how fragments searching works).

**def get\_aoi\_fragments(image):**

Parameters:

*image* – image to process.

Returns:

*fragments* – list of found fragments on the argument image.

This function processes AOI on query image making fragments and saving them into list of fragment, which will be returned. See details in “architecture description – how fragments searching works” section and \_frag\_process\_columns function description for algorithm main idea and features.

**def debug\_mark\_frag\_on\_img(image, fragment):**

Parameters:

*image* – image on which fragment’s edges will be drawn.

*fragment* – fragment which edges will be drawn on the argument image.

Returns:

*image* – image on which fragment’s edges were drawn.

This function is for logging and debugging. Draws argument fragment edges on the argument image and returns this image.

**def debug\_draw\_frag\_on\_img(image, fragment):**

Parameters:

*image* – image on which the fragment will be drawn.

*fragment* – fragment which will be drawn on the argument image.

Returns:

*image* – image with fragment drawn on it.

This function is for logging and debugging. Draws argument fragment over the argument image and returns this image.

**def get\_fragment\_center\_coords(fragment):**

Parameters:

*fragment* – python dict structure, contains fragment for which center pixel coordinates on source image will be calculated.

Returns:

*x, y* – fragments center pixel coordinates (int, int) on the source image where fragment from.

This function calculates fragment’s center pixel coordinates on source image. For example: fragment 100\*100 px was taken on source image 1000\*1000 px size. Fragment starts from 500th px to 600th px on X axis and from 0th to 100th pixel on Y axis. Then fragment’s center on source image will be:

xc = 500 + 100 / 2 = 550

yc = 0 + 100 / 2 = 50

This coordinates will be returned.

**def debug\_image(image, fragment, x, y):**

Parameters:

*image* – image on which debug info will be drawn.

*fragment* – fragment which edges will be drawn on image.

*x* – fragment center x axis coordinate of the point to draw.

*y* - fragment center y axis coordinate of the point to draw.

Returns:

*None*

Function for loggings and debug. Draws fragment’s edges and center on the image and saves that image on the hard drive. Image will be saved in directory path which specified in local.py settings file (output\_image\_dir).

**def read\_until\_contains(pattern, smc: SmoothieConnector):**

Parameters:

*pattern* – python str pattern which searched in smoothie output text.

*smc* – SmoothieConnector object which already successfully connected to smoothie.

Returns:

*response* – python str which contains smoothie’s response message.

This function reads smoothie’s responses until *pattern* isn’t found in them. Response that contains *pattern* is returned.

**def read\_until\_not(value, smc: SmoothieConnector):**

Parameters:

*value* – python str to search in smoothie responses.

*smc* – SmoothieConnector object which already successfully connected to smoothie.

Returns:

*response* – python str which is not equal to *value*.

This function reads smoothie’s output messages until message is not equal *value* argument. First smoothie’s response that is not equal to *value* is returned.

**def calibrate\_axis(axis\_current: Value, axis\_label, axis\_min\_key, axis\_max\_key, smc: SmoothieConnector):**

Parameters:

*axis\_current* – python multiprocessing.Value object which contains current coordinate value of axis to work with.

*axis\_label* – python str which contains label of axis to work with. This label is used to determinate constant key in the settings by which settings are available.

*axis\_min\_key* – minimal permissible value of the current axis.

*axis\_max\_key* – maximal permissible value of the current axis.

*smc* – connectors.SmoothieConnector object, should be already connected to smoothie board (may be None type if smoothie connection is not allowed in settings).

Returns:

*None*

This function is doing corkscrew calibration on a given axis if smoothie connection is allowed in the configuration, otherwise will set axis’s current value to default (side’s extreme value + or – offset, depends on settings). Corkscrew moving direction, offset from the axis border and smoothie usage is specified in config files. After function operation corkscrew will be placed in the specified in settings position, coordinates contained in Value object will respond real corkscrew position.

def corkscrew\_to\_start\_pos(smc: SmoothieConnector):

Parameters:

*smc* – connectors.SmoothieConnector object, should be already connected to smoothie board (may be None type if smoothie connection is not allowed in settings).

Returns:

*None*

This function is doing calibration for each permitted in settings axis (X, Y, Z). This function is caller of calibrate\_axis function, giving parameters for each permitted to calibration axis and shows process information on the console.

**def switch\_to\_relative(smc: SmoothieConnector):**

Parameters:

*smc* – connectors.SmoothieConnector object, should be already connected to smoothie board.

Returns:

*None*

This function sends “G91” code to smoothie enabling relative mode on smoothie. Also reads smoothie responses until smoothie is ready for new command.

**def validate\_moving\_key(value, key\_name, key\_min, key\_max, key\_min\_name, key\_max\_name, current\_value):**

Parameters:

*value* – int value of moving distance or force, can be positive or negative.

*key\_name* – str axis label or force char for which validation is doing.

*key\_min* – int axis minimal permitted value.

*key\_max* – int axis maximal permitted value.

*key\_min\_name* – str name of minimal value (e.g. “MIN\_X”), used to display message if value wasn’t validated.

*key\_max\_name* – str name of maximal value (e.g. “MAX\_X”), used to display message if value wasn’t validated.

*current\_value* – int, current axis value.

Returns:

*fail\_message* - str validation fail message if validation was failed,

*None* – if validation was successful.

This function is doing checking of move or force value is it in acceptable range (min and max values is specified in config file) and if moving won’t go beyond hardware posibilities, and returns python *None* if value is ok, otherwise str error message will be returned.

**def move\_cork\_to\_center(smc: SmoothieConnector):**

Parameters:

*smc* – connectors.SmoothieConnector object, should be already connected to smoothie board.

Returns:

*None*

This function is moving corkscrew to central position of X and Y coordinates.

**def run\_database\_mode():**

Parameters:

*None*

Returns:

*None*

This function is doing database mode actions. See actions, details and algorithm steps in architecture description (main steps, then database mode steps).

def run\_searching\_mode():

Parameters:

*None*

Returns:

*None*

This function is doing searching mode actions. See actions, details and algorithm steps in architecture description (main steps, then searching mode steps).

**def main():**

Parameters:

*None*

Returns:

*None*

This function runs script in the specified in config mode. This function is being executed only if script was started as standalone application.