Project Report
Depth Measuring
Digital Microscopy

Dorin Botan, 224187
Eriks Markevics, 224342
Dumitru Racicovschii, 224407

Supervisor:
Bo Brunsgaard Christensen

Hand in Date: 15th December 2017

Table of contents

List of Figures and Tables

Terminology and abbreviations

Digital Microscope[1] - a variation of a traditional optical microscope that uses optics and a digital camera to output a digital image to a monitor, by means of software.

Focus stack - a multiple of images taken from the same viewpoint but at different focus distances.

Depth map (aka. height map)[2] - an image containing information relating to the distance of the surfaces of scene objects from the viewpoint.

Meta data - data describing a resource.

Acutance[3] - perception of sharpness of an image.

[1] https://en.wikipedia.org/wiki/Digital_microscope
[2] https://en.wikipedia.org/wiki/Depth_map
[3] https://en.wikipedia.org/wiki/Acutance

Abstract

Digital microscopes are hastily replacing traditional optical microscopes worldwide. These not only improve the user experience, but also create new up-to-date possibilities and applications in the world of microscopy. One of the things that became possible with the advent of digital microscopy is performing high-precision measurements on microscopic objects, where traditional measuring methods simply won't work. With the use of image analysis algorithms, digital microscopes can perform precise size, area and angle measurements on X and Y axis of the scanned object. A highly desired feature is being able to perform measurements on Z axis (measuring height, volume and profile). This can only be done on three-dimensional objects.

Creating a 3D scan of an object can be done in several different ways. Most of them imply the use of various peripherals, such as laser projectors, stereo-cameras, light field cameras, rotating stands... Depth from defocus technique, proposed by P. Grossman in 1980 [4], makes it possible to get the 3D profile of an object by combining pictures taken with different focal lengths. By analyzing the luminance of each pixel in relation to its' neighbors, it is possible to stack a bunch of photos together and get the depth map of the scanned object, which can then be used to reconstruct the 3D profile. This technique makes it possible to perform precise 3D measurements by using a single digital camera, meaning that it can be software implemented in an existing digital microscope.

Following is a description of the proof of concept of a system that can perform measurements of microscopic objects on Z axis, based on an existing single camera digital microscope and not requiring any hardware changes.

[4] Pattern Recognition Letters, Volume 5, Issue 1, January 1987 P. Grossmann,. Depth from defocus

1. Introduction

Summary

With the digital revolution taking over the world of microscopy, now the most important task is to create new possibilities and applications to renew and optimize visual inspection and quality control procedures. One of the possible directions in the development of digital microscopy is creating a possibility to perform cheap, simple and accurate measurements on volumetric objects.

We do this by using a relatively old image analysis technique called *Depth from defocus*, and a full-HD digital microscope produced by TAGARNO Digital Camera Microscopes.

TAGARNO A/S is a Danish company located in Horsens, with more than 40 years of experience in developing solutions within digital camera microscopy. Over the years TAGARNO keeps developing and refining the line of user friendly digital microscopes and entering new markets such as agriculture, material science, medical inspection.
In 2013 TAGARNO introduced the first FULL HD digital microscope and thereby raised the bar for camera and microscopic solutions for industries around the world. Currently TAGARNO A/S is owned by Insero Horsens.

The developed project performs volumetric measurements on microscopic objects with a use of a single camera TAGARNO full-HD digital microscope, deduces the possible accuracy of the measurements and explores the feasibility of developing a robust client product.

Project is being developed as proof of concepts and for simplicity reasons targets mainly the electronics and manufacturing markets. Proceeding from this idea, system testing is carried out by measuring the height of various JEDEC standard SMT components. Accuracy is respectively calculated by taking into consideration the real height and deviation of the components according to their datasheets.

After the system was implemented, it went through a series of tests to measure the accuracy and see if it corresponds to the desired one.

System purpose

Purpose of the project is to create a proof of concept of a system that can perform precise measurements on Z axis based on TAGARNO A/S digital microscope, measure its accuracy and explore the feasibility of using it in an existing digital microscope.

Problem formulation

The project focus is to perform microscopic measurements on Z axis, using a digital microscope. To accelerate the development, image analysis algorithms are developed on a desktop PC by using Python 3 with scientific packages attached. Thereby, the project follows a two-tier architecture consisting of a digital microscope that takes the focus stack of the object and a desktop computer reconstructing the 3D profile and displaying it. Questions to be answered were the following:

1. How to perform Z axis measurements of microscopic SMT components
2. How to keep the system cheap but accurate
3. How to develop a preliminary version of the system in half a year

Problems to be solved in order to develop the system were:

1. How to organize the communication between the microscope and the PC
2. How to go from a flat image stack to a 3D profile of the object
3. How to display the output of the program
4. How to measure the accuracy of the performed measurements

Delimitations

- Image analysis algorithms are not optimized for speed
- System does not support scanning of transparent and glossy objects
- Project targets one lens, one camera model and one camera setup only
- Graphical user interface is developed to a level sufficient to display the output of the algorithm and is not the final version of it
- Components in the two-tier system are only being able to communicate over LAN
- No encryption or authentication is provided by the server

2. Analysis

In order to simplify and speed up the development of image analysis algorithms, we use several Python scientific libraries (SciPy[5] – ecosystem for mathematics and science, pillow[6] – imaging library, matplotlib[7] - 2D plotting library). Also, image analysis is being developed and ran on a desktop PC, because according to our preliminary estimations, this approach is much faster than developing the algorithms from scratch in C++. This led to the need to develop a 2-tier client-server system described by the image below.

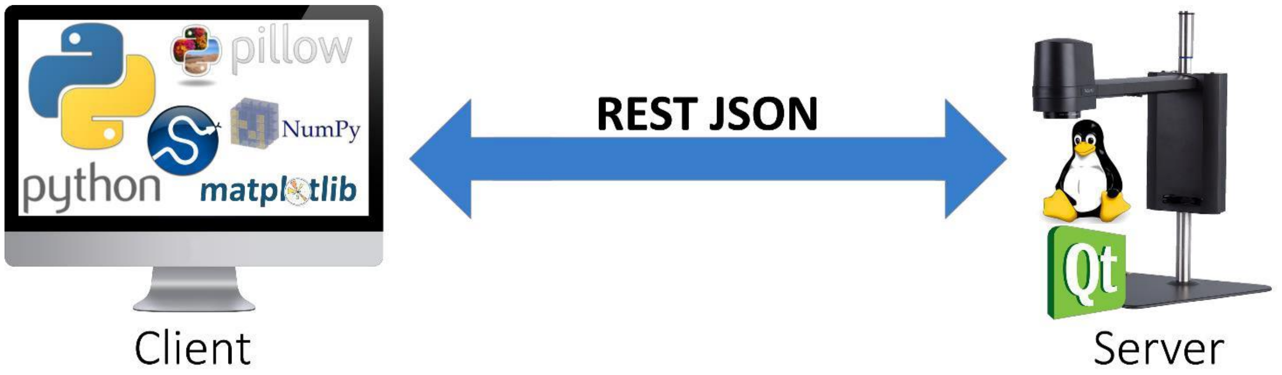

Figure 1 – Client-Server architecture overview

Main target of the project is electronics and manufacturing markets. Therefore, the minimum viable accuracy of the algorithm should be 0.05 mm, which corresponds to the minimal step in JEDEC standard for SMT components thickness.
The following requirements have been defined for the server and client parts, in order to determine the functionality of the final product. Items listed below were fully implemented without any changes.

[5] https://www.scipy.org/
[6] https://pillow.readthedocs.io
[7] https://matplotlib.org/

2.1 Server

The digital microscope acts as a server and is responsible for taking, storing and transmitting raw images to the client.

2.1.1 Functional and non-functional requirements

Functional requirements

What the system shall do:

1. Allow user to remotely control camera settings (ISO, Gain, Zoom, Iris)
2. Allow user to remotely control hardware settings (Light, IR filter, shutter)
3. Allow user to remotely take pictures and store them in internal memory
4. Allow user to remotely take focus stacks and store them in internal memory
5. Allow user to download taken images and focus stacks

Non-functional requirements

What the system shall be:

1. System shall target an existing embedded device (TAGARNO FHD Uno digital microscope)
2. System shall be accessible via LAN

2.2 Client

The desktop PC acts as a client and is responsible for preforming analysis and processing of the focus stacks and displaying the output to the user.

2.2.1 Functional and non-functional requirements

Functional requirements

What the system shall do:

1. System shall be able to connect to TAGARNO FHD Uno server and access its resources
2. System shall generate the depth map from the taken focus stack with maximum measurement deviation of 0.05 mm
3. System shall display the calculated depth map

Non-functional requirements

What the system shall be:

1. System shall be implemented entirely in software and not require any hardware changes or attachments to the server

2.3 Use case diagram

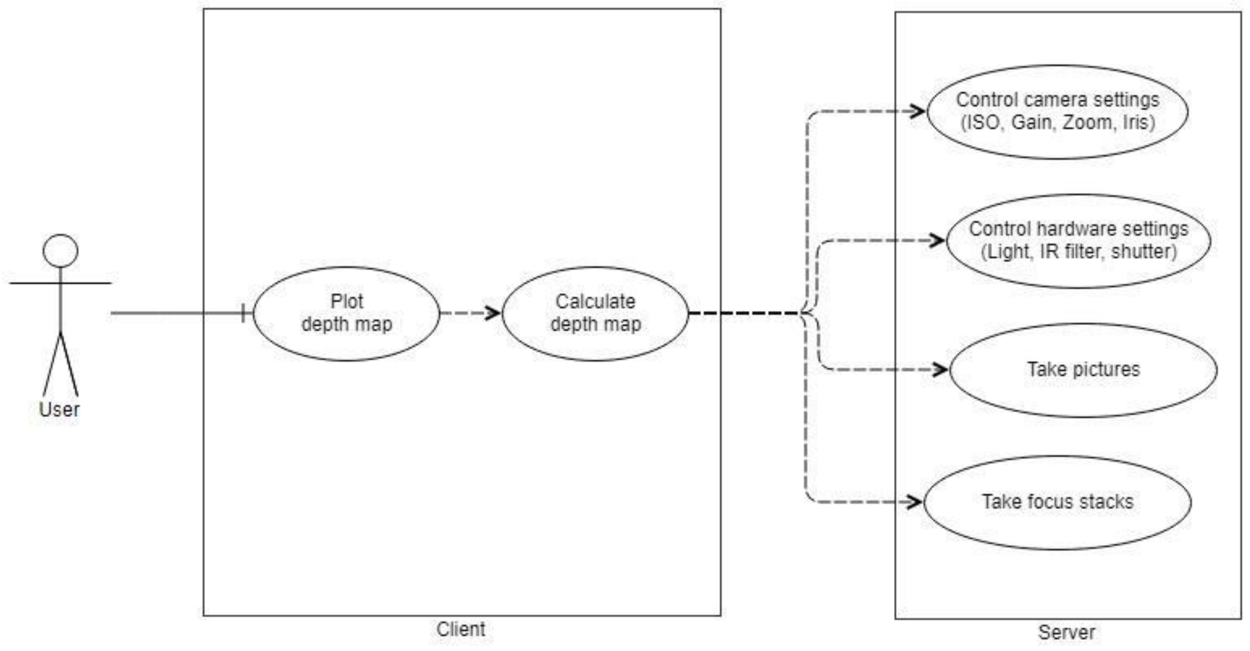
Control camera settings (ISO, Gain, Zoom, Iris)
Control hardware settings (Light, IR filter, shutter)
User
Plot depth map
Calculate depth map
Take pictures
Take focus stacks
Client
Server

Figure 2 - Use case diagram

The picture above displays the only possible use case of using the developed system.

2.3.1 Use case description

The client can perform one single action with the system – get the depth map of the object currently under the microscope lens. In order to facilitate the development, the server does not contain any software logic for image analysis and processing and acts as a remote interface to camera and hardware settings on the digital microscope.

Get depth map: Takes a focus stack of the object under the microscope, calculates the depth map and displays it to the user

Control camera settings: Sets and reads current camera settings (ISO, Gain, Zoom and Iris)

Control hardware settings: Sets and reads current hardware settings (Built-in lighting, Infrared filter and shutter)

Take pictures: Returns images from the digital camera and internal memory

Take focus stacks: Returns focus stacks with given camera settings

3. Design

The system is logically divided into 3 parts: hardware interface (partially implemented at TAGARNO A/S, and hence, is not being documented), server and client.

3.1 Server

The server implements an easy to use remote interface for the hardware device. Resources access to which is provided by the server are listed below:

/zoom	Control camera zoom
/magnification	Control camera magnification
/focus	Control camera iris
/led	Control built-in lighting
/laser	Control built-in laser
/ir	Control built-in infrared filter
/flip	Control hardware image flip functionality
/image	Take images from camera and internal storage
/meta	Take metadata for the images stored in internal storage
/focusstack	Take a new focus stack from camera

Communication to the server is being done via a RESTful web service with JSON encoded messages.

3.2 Client

The client should be able to query the microscope through a RESTful web service, get images and store them in a custom format. This format should have information about the conditions when the image was taken, i.e. focus value, zoom value, condition of the built-in LED etc.
This functionality, along with functionality for calculating the acutance, generating the depth map and visualizing it should be separated to make the system flexible. Below is the initial UML diagram for the client part:

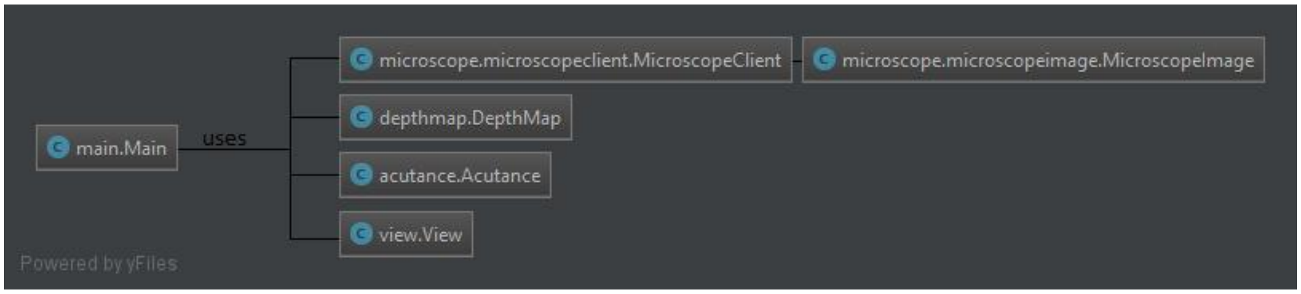
main.Main
uses
microscope.microscopeclient.MicroscopeClient
microscope.microscopeimage.MicroscopeImage
depthmap.DepthMap
acutance.Acutance
view.View
Powered by yFiles

Figure 3 - Initial UML diagram of the client

3.3 Image processing pipeline

In order to convert an image focus stack to a height map, images have to pass through the image processing pipeline shown in Figure 3. The pipeline includes the following steps:

0 – A new focus stack if shot by the microscope and sent to the client along with its corresponding metadata. Camera values (zoom, ISO, gain…), number of images in the stack, starting and ending iris values are sent by the client in the body of the request to the server. The resulting stack contains images taken with different iris values, from starting to ending iris value given by the client and with equal indentations between them.

1 – Images in the focus stack are greyscaled to reduce the amount of memory used and speed up the processing. Further measurements only work with image's luminance, meaning that shrinking 3 channels into one does not affect the quality of the processing.

2 – Images in the greyscaled focus stack are analyzed with various acutance metrics, in order to generate a robust acutance stack. Images in the acutance stack contain information about every pixel's acutance (edge sharpness), and respectively indicate which areas of the images are more in focus than the others. Detailed explanation of acutance calculation can be found in 4.3 Acutance metrics.

3 – Images in the acutance stack are combined to generate the relative height map, where every pixel contains the index number of the image where this exact pixel was the most in focus. This allows us to obtain a height map of the object with correct proportions of the measured objects, but incorrect height values.

4 – Each pixel in the relative height map is mapped to its real height according to the dependency line calculated during the calibration process. This gives us an absolute height map, where every pixel contains its height relative to the measurement table in mm. The absolute height map contains the real (absolute) height of the measured object at every pixel of the image.

5 – In order to nicely plot the calculated height map, it is converted in to a 3 channel, 8-bit RGB image, where reddish colors indicate the highest points in the image and bluish are the lowest. Due to quantization error, the accuracy of this representation is worse than for the absolute height map, respectively, the RGB height map is only used to plot the result to the user, while all the measurements are supposed to be performed on the grayscale absolute height map.

6 – The calculated RGB height map is shown to the user by the means of matplotlib.

In addition to the steps listed above, there may also be a need to perform smoothing of the calculated depth map, in order to reduce the effects of noise and generate a prettier output. This however may affect the accuracy of the algorithm, and therefore is not being done in the current version of the system.

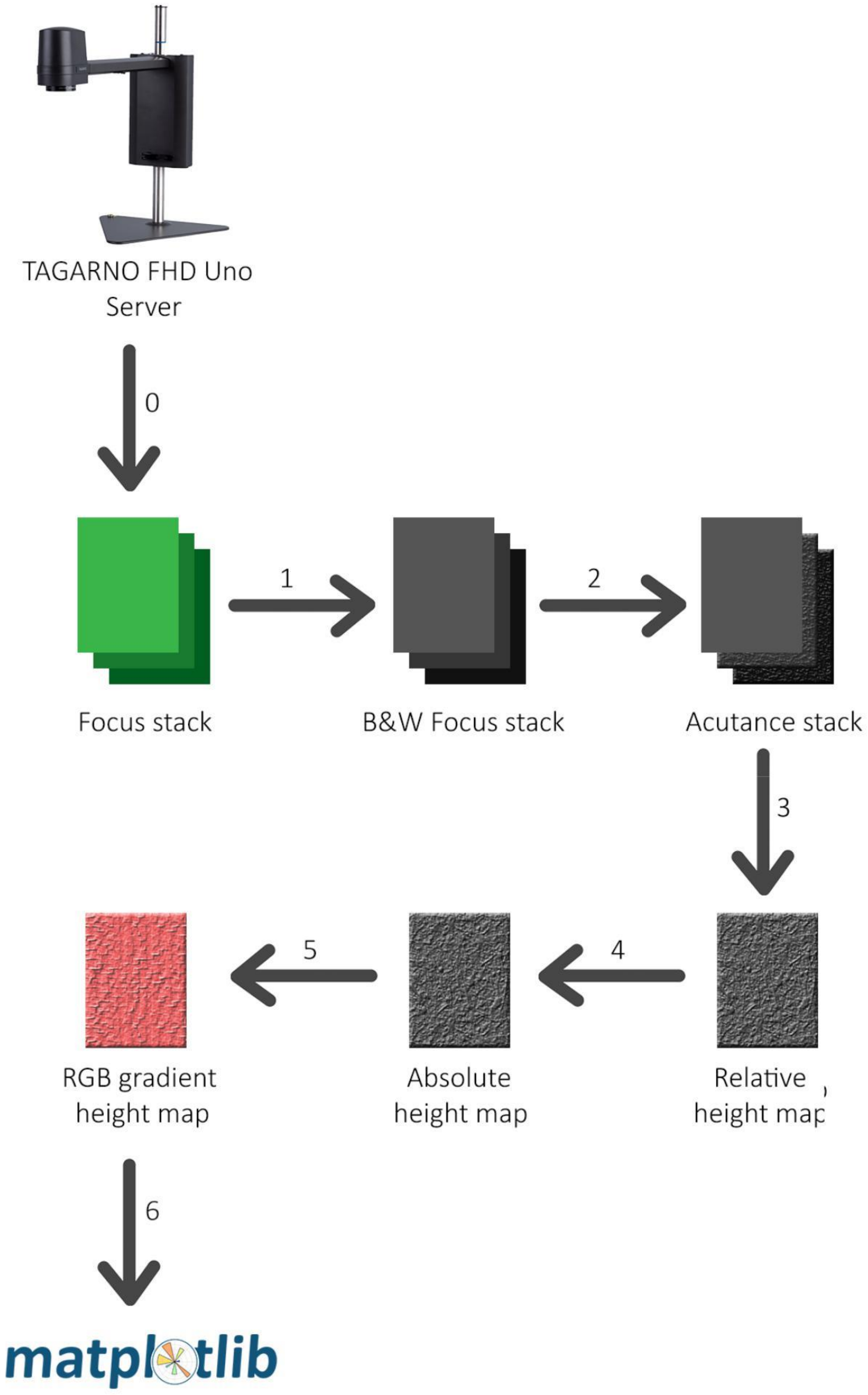

Figure 4 - Image processing pipeline

3.4 GUI Design

Graphical user interface is developed to an extend that is enough to demonstrate the system operability and is not designed to replenish the needs of a commercial product. Program output consists of a single window and does not involve any user interaction. The window contains a colored height map of the scanned object (1), a legend (2) and a one-dimensional plot of the height of the object (3).

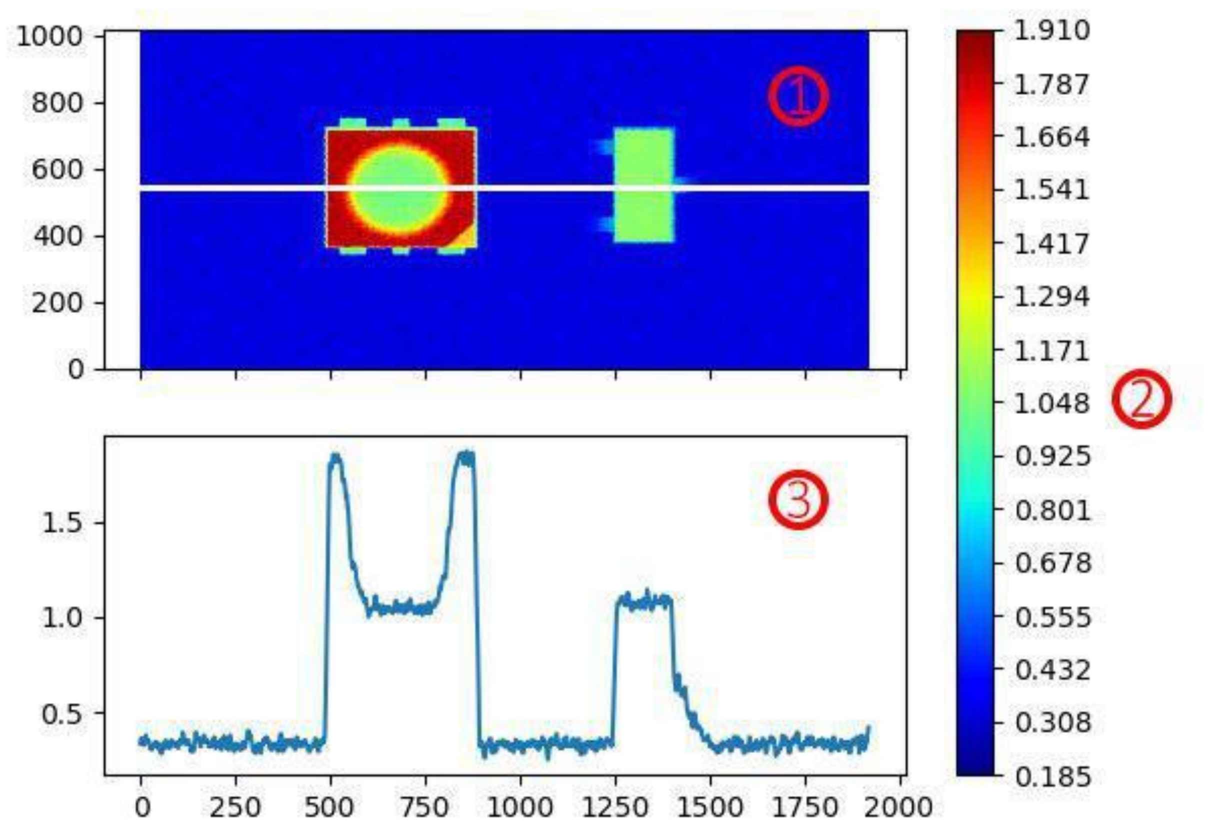

Figure 5 – Graphical output of the program

The height map (1) displays the shape and height of the objects under the microscope, according to the legend (2). Horizontal white line on the height map indicates the line segment plotted in (3), which simplifies the understanding of object's size by showing the height on the vertical axis (instead of showing it with color) and width on the vertical axis.

3.5 Technologies

To ease and speed up the development process, the following technologies were used:

3.5.1 Qt Embedded 4.8

Qt is a cross-platform application framework used for developing application software that can be run on various software and hardware platforms, including desktop PCs, smartphones and embedded devices. It uses standard C++ with several extensions including signals and slots that simplify the development of GUI or server applications which receive their own set of event information and should process them accordingly.
Qt Embedded is an assembly of Qt which has no need for a window system in order to run, lowering the amount of memory required by the framework and making it more suitable for Embedded Devices running f.ex. Embedded Linux.
In out project, Qt (together with C++) is being used on the server side and is responsible for implementing the web service interface, as well as all the interaction to the hardware. The reason for choosing Qt is that it is already configured for our embedded system, relieving the need to set up a new technology. Also, it is easier to use than pure C++. Qt 4.8 is being used under a student license.

3.5.2 Python 3

Python is a widely used, modern, general-purpose and multi-paradigm high-level programming language, who's design philosophy is to emphasize code readability and allow programmers to express concepts in fewer lines of codes than in languages such as C++ or Java. As a result, Python comes very handy in cases when it is necessary to increase developer's productivity and develop a product quickly. This is also facilitated by a wide range of modern features that come with Python, such as dynamic type system, automatic memory management, exception handling and a rich standard library.
In addition to this, being one of the most popular programming languages, Python supports a huge number of third-party libraries, some of which were very useful for our project.
Python 3 is a new version of the language (incompatible with the previous ones), which is mostly the same as older versions, but with a slightly redesigned way of working with the built-in data types and a lot of deprecated methods removed.
In our project, Python is being used on the client side and is responsible for image analysis and processing. The reasons for choosing Python as the programming language for the client were the following:

- It is free and Open Source. There is no need for licensing. The same applies to most of the third-party libraries available for Python.
- Project developers are familiar with the language, thanks to PCLI1 course at VIA UC.
- It has a great selection of third-party libraries, including libraries for scientific computing, image processing and web.

As an alternative, we also considered using MATLAB and C#/.NET, but preferred Python due to its nicer syntax and higher prevalence in the area of image analysis and scientific computing.

3.5.3 NumPy (Numerical Python)

NumPy is a library for Python, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy is open-source and widely used for various scientific calculations. In our project, NumPy is being used for image processing on the client side. Possible alternatives to NumPy were MATLAB, R, Julia, Spyder and Scilab. NumPy was preferred due to its close integration into Python.

3.5.4 SciPy (Scientific Python)

SciPy is a library for Python used for scientific and technical computing. It builds on the NumPy array object, being a part of NumPy stack, and adds modules for linear algebra, FFT, signal and image processing and other common tasks in science and engineering. In our project, SciPy is being used for image analysis and processing on the client side. As well as NumPy, SciPy is open-source.

3.5.5 Pillow

Pillow is a friendly fork of PIL (Python Imaging Library), which adds image processing capabilities to Python. In out project, Pillow is used for accessing, storing and editing image files. The library is open-source.

3.5.6 matplotlib

matplotlib is a 2D plotting library for Python and NumPy. matplotlib allows generating plots, histograms, chrats, etc. with just a few lines of code. In out project matplotlib is used to plot the calculated depth map to the user. A possible alternative to matplotlib was Gnuplot, but mutplotlib was preferred due to its close integration into Python. matplotlib is open-source.

3.5.7 Doxygen

Doxygen is a tool for generating documentation form annotated sources, with support for multiple programming languages including C/C++, PHP, Java, Python… Doxygen is open-source and de facto standard tool for C++ code documentation. In our project, Doxygen was used to generate the HTML formatted documentation for the server source code.

4. Implementation

4.1 Server

Below is the UML Class diagram of the server. The diagram only contains the classes related to the web service and does not contain any of the classes used for interacting with hardware (for privacy reasons).

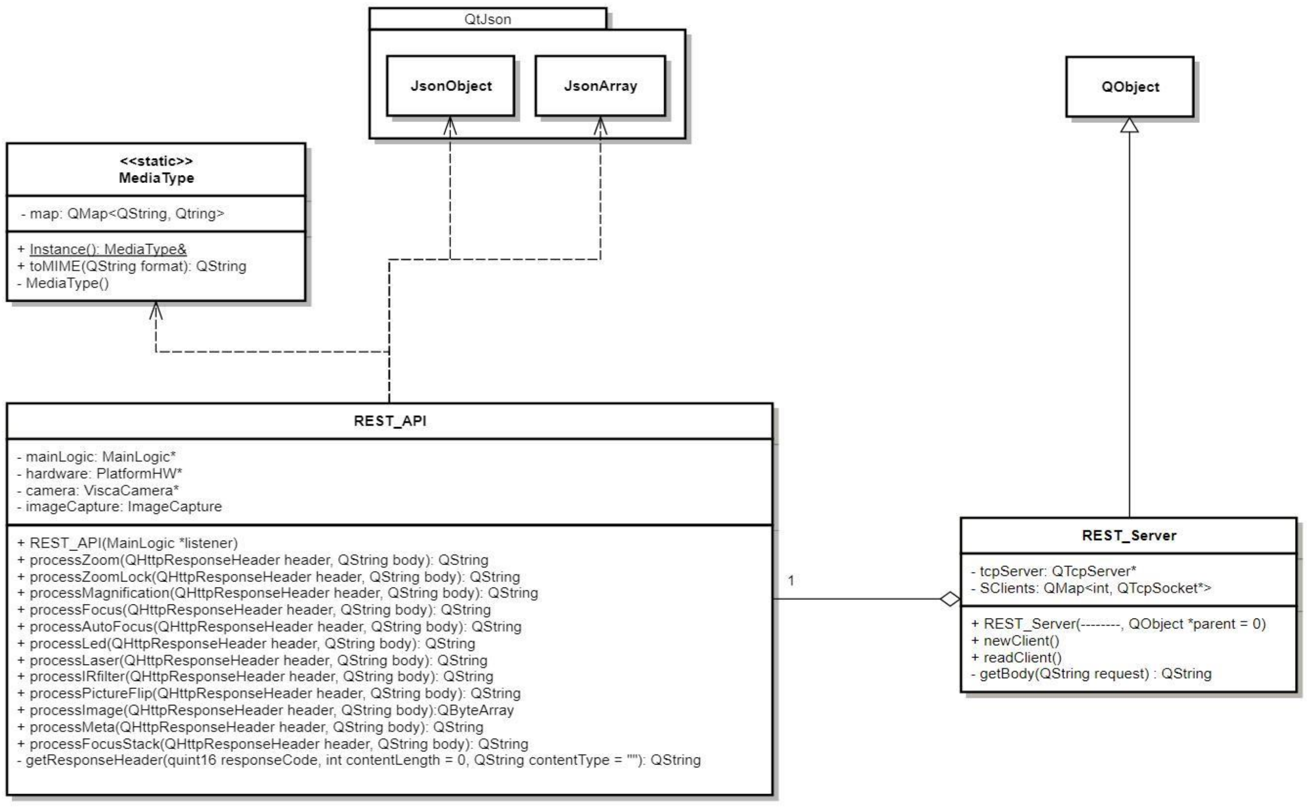
QtJson
JsonObject
JsonArray
QObject
<<static>>
MediaType
- map: QMap<QString, Qtring>
+ Instance(): MediaType&
+ toMIME(QString format): QString
- MediaType()
REST_API
- mainLogic: MainLogic*
- hardware: PlatformHW*
- camera: ViscaCamera*
- imageCapture: ImageCapture
+ REST_API(MainLogic *listener)
+ processZoom(QHttpResponseHeader header, QString body): QString
+ processZoomLock(QHttpResponseHeader header, QString body): QString
+ processMagnification(QHttpResponseHeader header, QString body): QString
+ processFocus(QHttpResponseHeader header, QString body): QString
+ processAutoFocus(QHttpResponseHeader header, QString body): QString
+ processLed(QHttpResponseHeader header, QString body): QString
+ processLaser(QHttpResponseHeader header, QString body): QString
+ processIRfilter(QHttpResponseHeader header, QString body): QString
+ processPictureFlip(QHttpResponseHeader header, QString body): QString
+ processImage(QHttpResponseHeader header, QString body):QByteArray
+ processMeta(QHttpResponseHeader header, QString body): QString
+ processFocusStack(QHttpResponseHeader header, QString body): QString
- getResponseHeader(quint16 responseCode, int contentLength = 0, QString contentType = ""): QString
1
REST_Server
- tcpServer: QTcpServer*
- SClients: QMap<int, QTcpSocket*>
+ REST_Server(--------, QObject *parent = 0)
+ newClient()
+ readClient()
- getBody(QString request) : QString

Figure 6 - Server UML Class diagram

The system functionality is mostly contained in the classes “REST_Server” and “REST_API”.
A REST_Server object is used for initializing a web-server at port 80, which can receive / process requests from multiple clients and return responses. The class heavily relies on Qt Network C++ classes, such as QTcpServer and QNetworkInterface, which implement most of the functionality, but also keeps track of multiple parallel connections and processes incoming requests. REST_Server is the only way of interaction with the server part of the system and the entry point to the program.
A REST_API object is used to implement the access to all of the server resources and format the response message.
In addition to this, the classes in QtJson namespace implement the processing of JSON encoded messages, while MediaType lists all of the used response data types and comes in handy in the formation of the response message.

Server part is entirely implemented in C++ and Qt. The web service is built REST[8] style and implements four HTTP methods - GET, POST, PUT and DELETE. The following resources/methods are accessible via the web service:

	GET	POST
/zoom	get zoom level *raw value [0 - 16384] as utf-8 text	-
/zoom/lock	get zoom lock value *binary value [0 - 1] as utf-8 text	lock zoom
/magnification	get magnification level *decimal value[1.66745 - 33.889] as utf-8 text	-
/focus	get focus level *raw value [4096 - 59392] as utf-8 text	-
/focus/auto	get autofocus value *binary value [0 - 1] as utf-8 text	enable autofocus
/led	get LED state *binary value [0 - 1] as utf-8 text	turn on LED
/laser	get laser state *binary value [0 - 1] as utf-8 text	turn on laser
/ir	-	enable IR filter
/flip	-	enable image flip
/image	get an image from the camera *image as image/png	-
/image/*123.png*	get given image from internal memory *image as image/png	take an image with the given name and store on internal memory
/meta	get meta data *text as utf-8 text	-
/focusstack	-	take image focusstack *{"zoom": [0 - 16384], "minfocus": [4096 - 59392], "maxfocus": [4096 - 59392], "steps" : [2 - ..]} *metadata as utf-8 text

Table 1 - REST resources (GET, POST)

8 Fielding, Roy Thomas (2000). "Architectural Styles and the Design of Network-based Software Architectures". Dissertation. University of California, Irvine.

	PUT	DELETE
/zoom	set zoom to the requested level *{"value" : [0 - 16384]} as utf-8 text	-
/zoom/lock	toggle zoom lock	unlock zoom
/magnification	set magnification to the requested level *{"value" : [1.66745 - 33.889]} as utf-8 text	-
/focus	set magnification to the requested level *{"value" : [1.66745 - 33.889]} as utf-8 text	-
/focus/auto	toggle autofocus	disable autofocus
/led	toggle LED state	turn off LED
/laser	toggle laser state	turn off laser
/ir	-	disable IR filter
/flip	-	disable image flip
/image	-	-
/image/*123.png*	-	delete image with the given name from internal memory
/meta	-	-
/focusstack	-	-

Table 2 - REST resources (PUT, DELETE)

During the development period, source code for the server part was documented by following an iterative approach and using Doxygen to auto-generate the documentation files from the comments in the source code. The intent of this documentation was to simplify the collaborative work on the server’s code base.
The documentation is available in form of HTML files in Appendix 1.

4.2 Client

Client part is entirely implemented in Python 3, with abundant use of various scientific libraries, such as pillow, NumPy, SciPy and matplotlib. This allowed us to operate with the already implemented image processing methods, instead of writing them from scratch. In this regard, client code base is very small and concise.
Below is the UML Class diagram of the client:

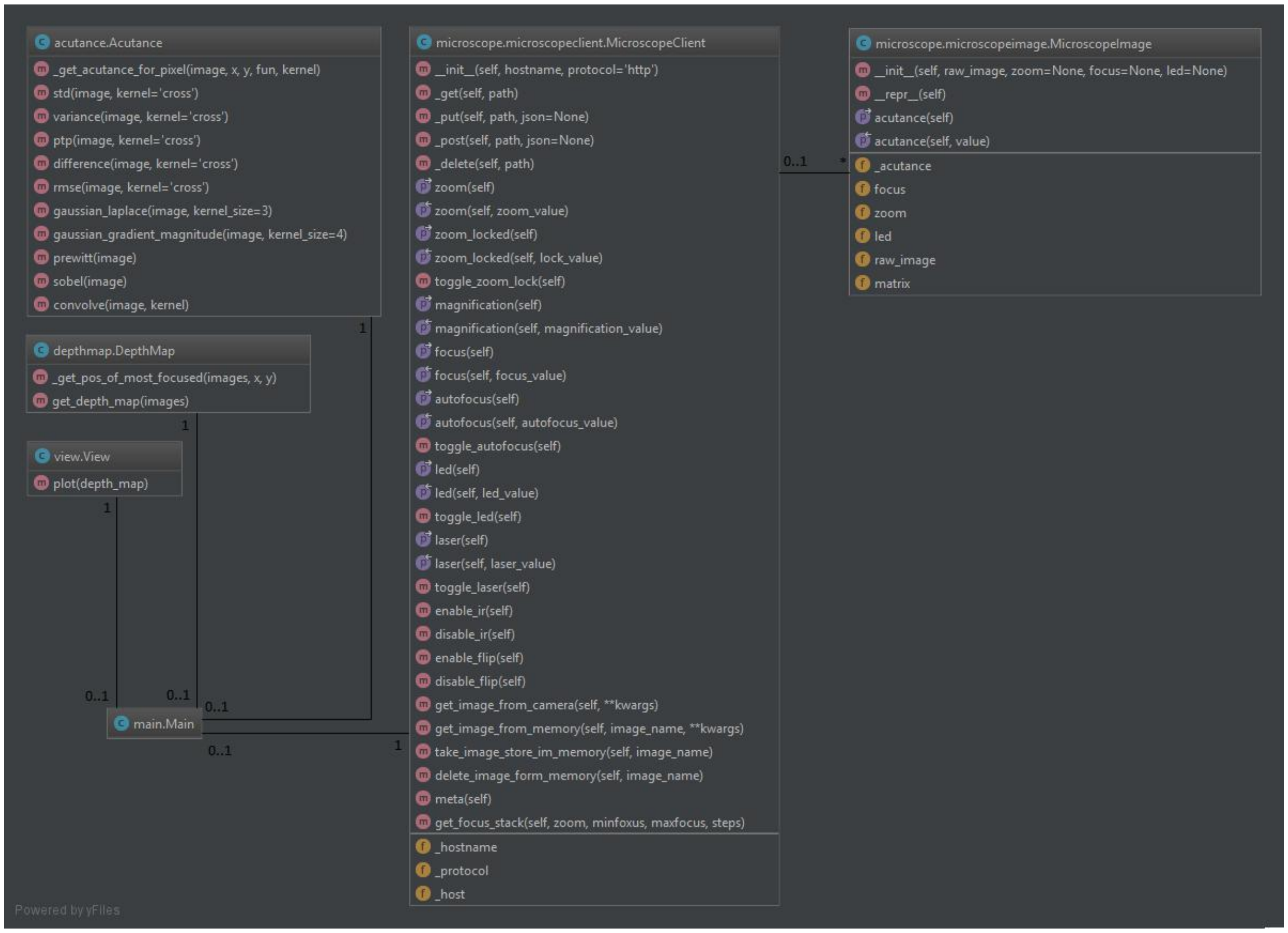
acutance.Acutance
_get_acutance_for_pixel(image, x, y, fun, kernel)
std(image, kernel='cross')
variance(image, kernel='cross')
ptp(image, kernel='cross')
difference(image, kernel='cross')
rmse(image, kernel='cross')
gaussian_laplace(image, kernel_size=3)
gaussian_gradient_magnitude(image, kernel_size=4)
prewitt(image)
sobel(image)
convolve(image, kernel)
depthmap.DepthMap
_get_pos_of_most_focused(images, x, y)
get_depth_map(images)
view.View
plot(depth_map)
main.Main
microscope.microscopeclient.MicroscopeClient
__init__(self, hostname, protocol='http')
_get(self, path)
_put(self, path, json=None)
_post(self, path, json=None)
_delete(self, path)
zoom(self)
zoom(self, zoom_value)
zoom_locked(self)
zoom_locked(self, lock_value)
toggle_zoom_lock(self)
magnification(self)
magnification(self, magnification_value)
focus(self)
focus(self, focus_value)
autofocus(self)
autofocus(self, autofocus_value)
toggle_autofocus(self)
led(self)
led(self, led_value)
toggle_led(self)
laser(self)
laser(self, laser_value)
toggle_laser(self)
enable_ir(self)
disable_ir(self)
enable_flip(self)
disable_flip(self)
get_image_from_camera(self, **kwargs)
get_image_from_memory(self, image_name, **kwargs)
take_image_store_im_memory(self, image_name)
delete_image_form_memory(self, image_name)
meta(self)
get_focus_stack(self, zoom, minfoxus, maxfocus, steps)
_hostname
_protocol
_host
microscope.microscopeimage.MicroscopeImage
__init__(self, raw_image, zoom=None, focus=None, led=None)
__repr__(self)
acutance(self)
acutance(self, value)
_acutance
focus
zoom
led
raw_image
matrix
0..1
*
1
1
1
0..1
0..1
0..1
0..1
1
Powered by yFiles

Figure 7 - UML class diagram of the client

OOP approach was used to implement client part of the system.
MicroscopeImage class contains the image itself in PIL.Image format along with the image properties, numpy.array representation and is also capable of storing acutance map as a property.
MicroscopeClient provides an easy to use interface for the microscope, encapsulating all the REST methods. It uses MicroscopeImage format both when returning individual images and the focus stack at once.
Acutance class provides ten different algorithms to calculate an acutance map for an image. The algorithms are described in detail in the next section (4.3).
Depth map class analyses the acutance stack, generating a depth map. Each value in the depth map is showing height of the given pixel relatively to the height of the lowest focus used. The height is

only valid for one microscope setup, for which the calibration was performed and a dependency between focus distance and actual height was computed.
View class is visualizing the depth map according to designed GUI, described in 3.4.
Main, which is an entry point of the client, is a script, operating with all the available classes to get focus stack from the microscope, calculate acutance metric for every image, create a depth map and visualize it. It is not currently possible to specify any parameters when running the client because the system is configured for one single setup of the microscope, for which the calibration was performed.

4.3 Acutance metrics

An acutance map shows which areas of the image are in focus, and correspondingly what is the focal length of the image. This is a crucial part of the algorithm and also the main source of errors. Various acutance metrics have been tested in order to find a combination that best fits our needs. A good acutance metric should generate an acutance map with biggest range and a maximally linear gradient between the points that are most and least in focus.
Below are the outputs from several unsuccessful acutance metrics that we have tested:

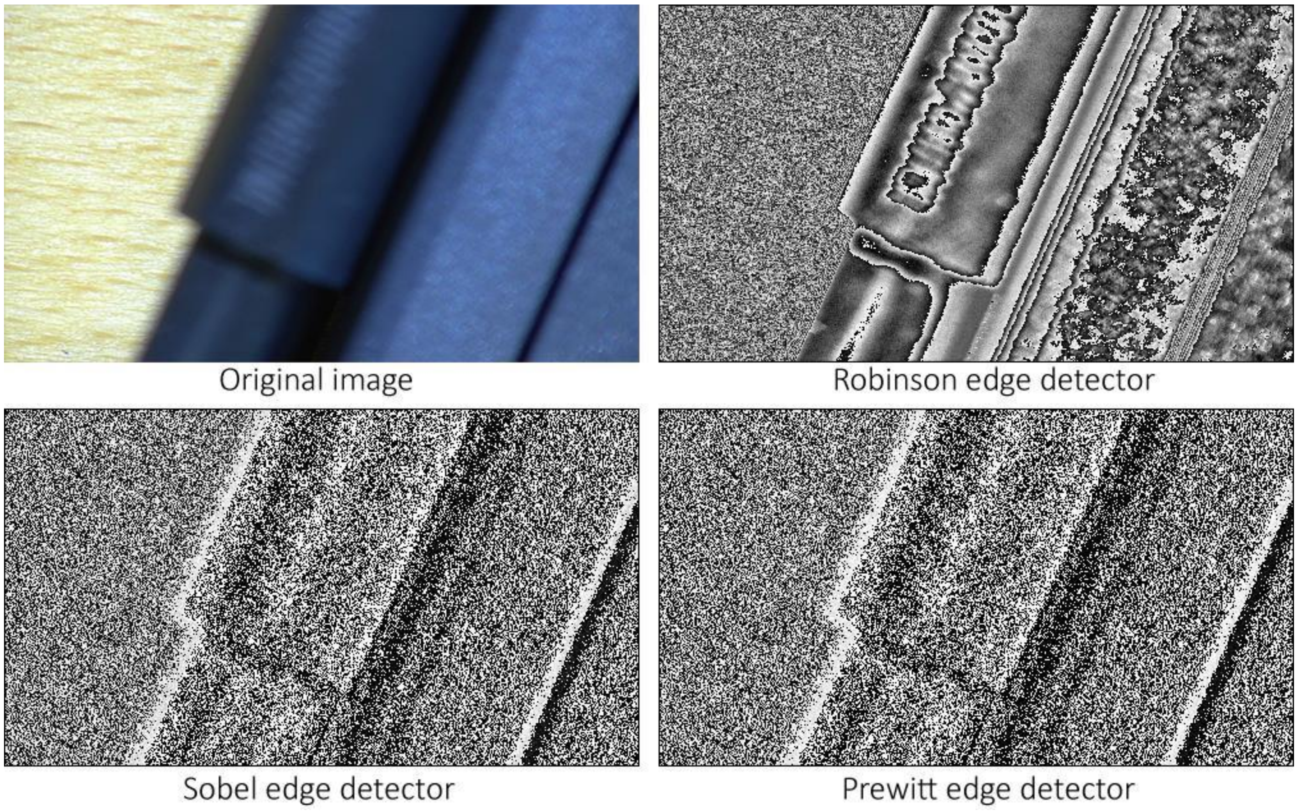

Figure 8 - Unsuccessful acutance metrics

In the original image, most in-focus area is the table on the left of the image. Several edge detection techniques have been applied on raw images, in order to generate the acutance map. As can be seen from the images above, there is no big difference between the area in and out of focus. Presumably this is caused by a high level of noise in the images.

Below are several successful acutance metrics that we have tested:

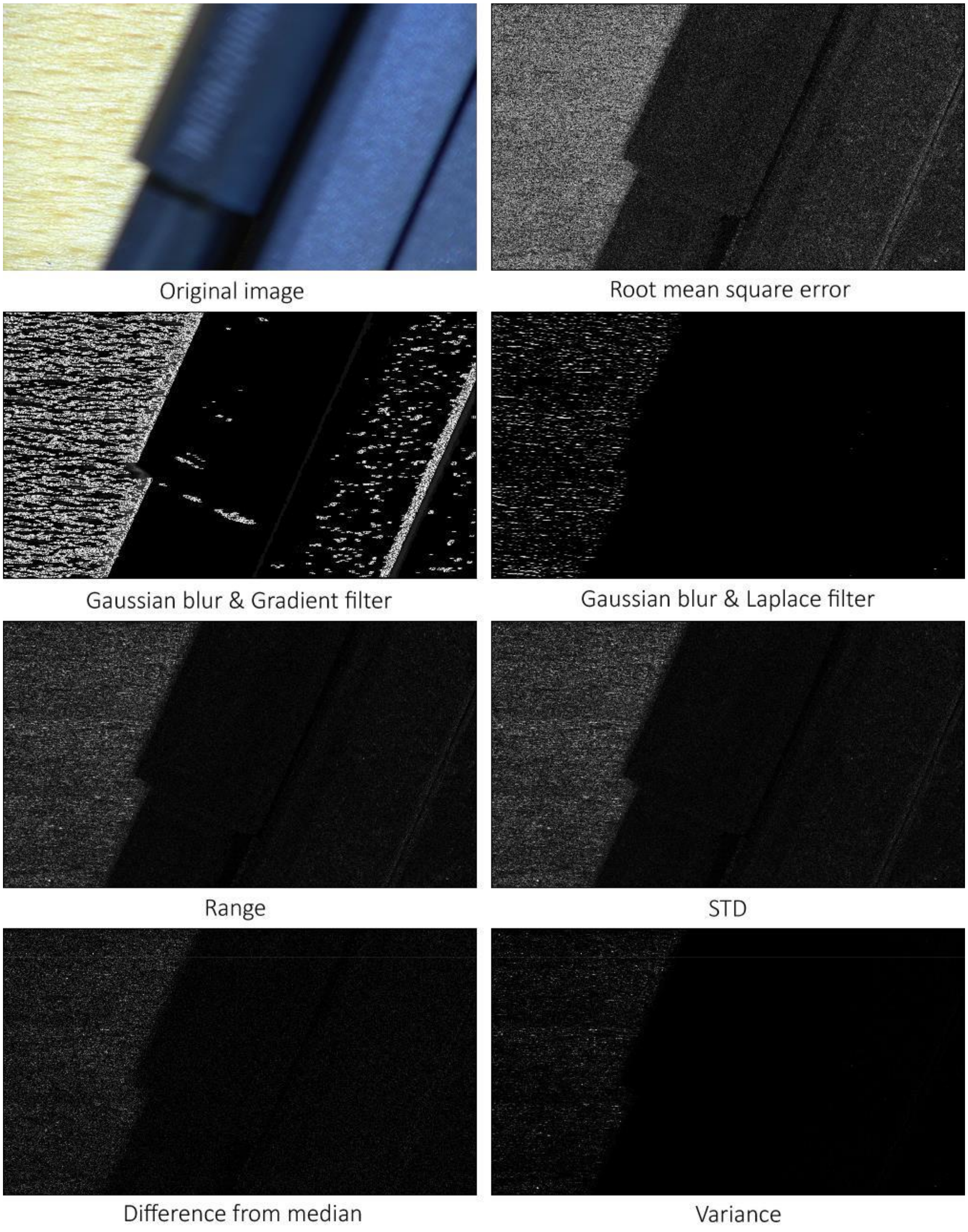

Figure 9 - Successful acutance metrics

Various edge detection techniques in combination with Gaussian blur and different kernel sizes have been tried and gave satisfactory results (Gaussian blur & Gradient filter, Gaussian blur & Laplace filter from the image above). However, in addition to eliminating the noise, Gaussian blur also smoothes the real edges, introducing a new source of error in to the algorithm.
Several acutance metrics based on deviation in luminosity levels between the pixels have been tried and also gave satisfactory results (Root mean square error, Range, Standard deviation and Variance from the image above). Visually, RMSE, STD and Range based metrics give similar results, with only the brightness being slightly different. Range based method however is much faster to compute because it does not require the square root operator. The weak part of all the 3 above discussed metrics is that pronounced edges are being detected even when those are not in focus (can be seen by the oblique line on the right part of the images). Variance based metric does not have this problem but it is also bad at detecting areas in focus.
In the end, we derived our own metric based on difference between the pixel's luminosity level and mean luminosity of the pixels around it. This gave us the ability to detect areas in focus comparable to Range and STD based methods but not detect false in-focus areas at highly pronounced edges. The resulting acutance map generated with this method can be seen in the Figure 9 (Difference from median).
Below is the implementation of the Difference from median acutance metric:

```
@staticmethod
def _get_acutance_for_pixel(image, x, y, fun, kernel):
    if kernel == 'round':
        return fun([image[x - 1][y], image[x][y - 1], image[x +
1][y], image[x][y + 1]])
    elif kernel == 'cross':
        return fun([image[x][y], image[x - 1][y], image[x][y -
1], image[x + 1][y], image[x][y + 1]])
    elif kernel == 'square':
        return fun([image[x - 1][y - 1], image[x - 1][y], image[x -
1][y + 1],
                    image[x][y - 1], image[x][y], image[x][y + 1],
                    image[x + 1][y - 1], image[x + 1][y], image[x +
1][y + 1]])
    else:
        raise NotImplementedError()

@staticmethod
def difference(image, kernel='cross'):
    return [[abs(Acutance._get_acutance_for_pixel(image.matrix,
x, y, numpy.mean, kernel) - image.matrix[x][y])
             for y in range(1, image.matrix.shape[1] - 1)]
            for x in range(1, image.matrix.shape[0] - 1)]
```

Implementation of the other metrics that we have tried can be found in the client's source code (acutance.py).

4.4 Calibration

In order to convert the relative height of the objects to their real height, the algorithm was calibrated on objects with known heights. Calibration of our microscope was held on various SMT components with heights taken from their datasheets. In comparison to using commercial *etalon* heights for calibration, this approach turned out much cheaper, easier and more precise (considering that with cheaper components we can get a much bigger sample space). Below is the table of the SMT components used for calibration:

Component	Height (mm)	Deviation (%)	Datasheet
Table	0	-	-
SOT 23 (MFE Transistor)	1.025	± 9.5	datasheet
TXB0104 (Voltage-level Transistor)	1.27	-	datasheet
LMV358M (Operational Amplifier)	1.75	-	datasheet
Osram Opto LRTG GFTG (RGB LED)	2	-	datasheet
Capacitator in **1825*** package	2.54	-	datasheet

Table 3 - SMT components used for calibration

Below is the graph showing the dependency between the height of the object and camera iris value. Horizontal axis indicates component heights. Vertical axis contains raw iris values.

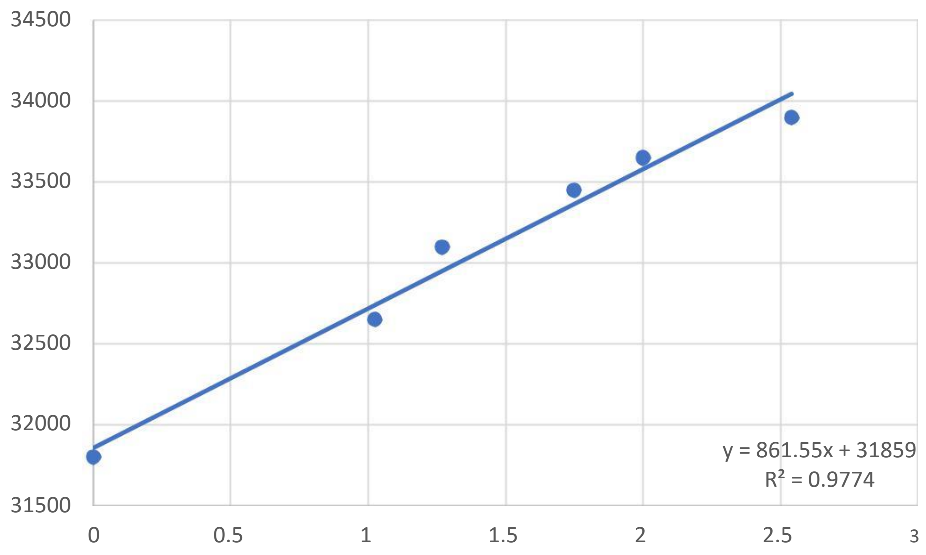

Figure 10 - Height vs Iris value dependency

The relationship was calculated as a linear regression. Considering quite big coefficient of determination (R^2 = 0.98), we conclude that the linear model is well suited to describe the dependence.

It is important to consider that the dependency is only valid for our particular microscope, in one single setup. If any of the physical components of the device is being changed (camera height, camera angle, lens, zoom…), the calibration must be re-done.

5. Testing

This section defines test strategies and procedures in relation to the functional and non-functional requirements. Two tires of the client-server system are tested separately.

5.1 Server

5.1.1 Remotely control camera settings

One of the essential functions of the server part of the system is to allow the client to remotely control Zoom and Iris settings.

Testing procedure: The test was done by performing remote calls to the /zoom, /zoom/lock, /magnification, /focus, /focus/auto and a dummy resource, with 5 different HTTP methods – GET, POST, PUT, DELETE and CONNECT (dummy method). The list of all the tested paths, as well as the expected results can be found in Appendix 2. In addition to visual observations, we also tracked system messages on the server, to get sure that camera settings are set correctly to the values given with the request.

Testing results: After testing all the functionalities, everything works as it should. Response codes, response body and system actions to the requests meet the expectations.

Testing exceptions or errors:

- If the client requests a nonexistent resource, a response with code 404 and empty body is returned.
- If the client requests a nonexistent method of an existing resource, a response with code 404 and empty body is returned.
- If the client puts an incorrectly formed message in the body of the request, where there is a need for an input parameter, a response with code 400 and empty body is returned.
- If the client puts an out of bound parameter in the body of the request, where there is a need for an input parameter, the requests is successfully processed with bound values instead.

5.1.2 Remotely control hardware settings

Another essential function of the server part of the system is to allow the client to remotely control LED, Laser, IR-filter and hardware image flip settings.

Testing procedure: The test was done by performing remote calls to the /led, /laser, /ir and /flip, with 5 different HTTP methods – GET, POST, PUT, DELETE and CONNECT (dummy method). The list of all the tested paths, as well as the expected results can be found in Appendix 2. All of the accessible hardware resources are binary (can only be switched on or off) and visually observing the changes on client's request on was sufficient for testing.

Testing results: After testing all the functionalities, everything works as it should. Response codes, response body and system actions to the requests meet the expectations.

Testing exceptions or errors:

- If the client requests a nonexistent method of an existing resource, a response with code 404 and empty body is returned.

5.1.3 Remotely capture pictures and store them in internal memory

Another essential function of the server part of the system is to allow the client to remotely capture pictures and store them in internal memory of the server.

Testing procedure: The test was done by performing remote calls to the /image resource, with 5 different HTTP methods – GET, POST, PUT, DELETE and CONNECT (dummy method). The list of all the tested paths, as well as the expected results can be found in Appendix 2. We also tracked system messages on the server and visually checked the captured images, to get sure that images are properly taken and stored.

Testing results: After testing the functionality, everything works as it should. Response codes, response body and system actions to the requests meet the expectations.

Testing exceptions or errors:

- If the client requests a nonexistent method, a response with code 404 and empty body is returned.
- If the client captures a new image with an existing name, the existing image is overwritten.
- If the server internal memory is full, pictures cease to be saved, without warning the client about this. Made on purpose, to simplify parsing the response messages on the client side.

5.1.4 Remotely take focus stacks and store them in internal memory

Another essential function of the server part of the system is to allow the client to remotely capture focus stacks and store them in internal memory.

Testing procedure: The test was done by performing remote calls to the /focusstack resource, with 5 different HTTP methods – GET, POST, PUT, DELETE and CONNECT (dummy method). The list of all the tested paths, as well as the expected results can be found in Appendix 2. We also tracked system messages on the server and visually checked the captured focus stacks, to get sure that all the images and meta data are properly taken and stored.

Testing results: After testing all the functionalities, everything works as it should. Response codes, response body and system actions to the requests meet the expectations.

Testing exceptions or errors:

- If the client requests a nonexistent method of an existing resource, a response with code 404 and empty body is returned.
- If the client puts an incorrectly formed message in the body of the request, a response with code 400 and empty body is returned.
- If the client puts an out of bound parameter in the body of the request, the requests is successfully processed with bound values instead.
- If the server internal memory is full, pictures cease to be saved, without warning the client about this. Made on purpose, to simplify parsing the response messages on the client side.

5.1.5 Access images and focus stacks from the internal memory

Server must provide the client with remote access to the images and focus stacks stored in internal memory.

Testing procedure: The test was done by performing remote calls to the /image/*imageName* and /meta resources, with 5 different HTTP methods – GET, POST, PUT, DELETE and CONNECT (dummy method). The list of all the tested paths, as well as the expected results can be found in Appendix 2. In addition to the response codes, we also tracked system messages on the server and visually checked the returned images and data, to get sure that all the images and meta data are properly found and returned.

Testing results: After testing all the functionalities, everything works as it should. Response codes, response body and system actions to the requests meet the expectations.

Testing exceptions or errors:

- If the client requests a nonexistent method of an existing resource, a response with code 404 and empty body is returned.
- If the client requests a nonexistent image or meta file, a response with code 404 and empty body is returned.
- If the client tries to delete a nonexistent image, a response with code 404 and empty body is returned.

5.2 Client

5.2.1 Connect to TAGARNO FHD Uno server and access its resources

Connection to TAGARNO FHD Uno was tested at the same time as the test of server side since actual tests were performed using the python library which is part of the system (see 5.1).

5.2.2 Generate the depth map from the taken focus stack

Testing procedure: System was tested on the same components which were used to calibrate the microscope, because those were the only components available during development. Below is a table showing real size of the components compared to the measured size:

Component	Real height (mm)	Measured height (mm)	Deviation (mm)
Table	0	0.045	0.045
SOT 23 (MFE Transistor)	1.025	0.993	0.032
TXB0104 (Voltage-level Transistor)	1.27	1.239	0.031
LMV358M (Operational Amplifier)	1.75	1.777	0.027
Osram Opto LRTG GFTG (RGB LED)	2	1.971	0.029
Capacitator in **1825*** package	2.54	2.562	0.022

Table 4 - Comparison of real and measured size of components

Testing results: Mean for the deviation is 0.031, but five objects are not enough for computing an average deviation. Therefore, we need to compute confidence intervals for deviation mean to be sure that the deviation will not be higher than expected (0.05 mm). MATLAB was used to perform confidence intervals calculations.
With 95% certainty measurement deviation will be in the range of [0.022898; 0.039102] mm, which is below the expected maximum of 0.05 mm and passes the functional requirement.
Since maximal deviation in 95% cases is 0.039 mm and it is much lower than expected 0.05 mm, we also computed the interval for 99% certainty. With 99% certainty deviation mean will be in the range of [0.018292; 0.043708] mm, which is also below the expected maximum of 0.05 mm and still passes the functional requirement.
This amount of testing is not enough for a commercial project. Its main purpose is to show that that the system is working and to prove the concept of depth measuring based on a single camera. System should be tested on a bigger quantity of components to get a wider and more precise average deviation. However, the components were not available and therefore limited the amount of testing we could perform.

5.2.3 Shall display the calculated depth map

Testing procedure: GUI was tested by running the program and visual comparing of the GUI output to the actual content of the depth map.
Testing results: After running the program a various number of times, everything works as it should. Both colors and values are shown correctly and match the content of the depth map.

6. Results

In the above described project, we developed a system which is capable of performing height measurements on microscopic objects by running on an existing, single camera digital microscope, with no hardware changes/attachments required. The original goal of the project – try to reach an accuracy of 0.05 mm was met. This proved system's operability and applicability to a real-world task – measuring JEDEC standard SMT electric components.
The system was developed as a proof of concept and revealed the strong and weak parts of our approach. The system is very accurate, despite the fact that it is entirely implemented in software. Also, this makes it much cheaper to embody it in an existing product, than by any other methods currently known. However, this approach has a very high computational complexity and is slow. It may require sacrificing accuracy in order to decrease computation time. Other possible ways of improving the system are listed in 9. Future development & Recommendations.

7. Discussion

The project was different from what we were used to in our semester projects. It included a lot of self-studying, research, discussing and trying out things. As a result, there were cases, when a big amount of work resulted in just a couple of lines of working code.
The resulting system we consider to be satisfying, even though there are things we would do differently the next time. It could be a good idea to start implementing the most important part of the system, which is image processing, straight away rather than using horizontal approach and implementing system parts in the order they will be used. This approach we have chosen resulted in using part of TAGARNO code to control hardware settings.
However, there were also some decisions taken, which turned out to be very helpful. Using python and python scientific libraries instead of writing everything in C++ allowed us to try many options in a limited amount of time. Using SMT components for calibration and system testing helped to avoid the need of expensive commercial height etalons and also increased accuracy by allowing us to perform measurements on a higher number of measurements. It also brought the project closer to the real-world demands.

8. Conclusion

Depth from defocused technique proved to be applicable for the field of digital microscopy having several significant advantages over the other known and widely used techniques. Main advantage is that it only requires a single camera with motorized IRIS (about every advanced digital camera nowadays), meaning that it can be implemented in a wide range of already existing hardware devices. Digital microscopy is one of the examples of it.
Also, this technique appeared to be very accurate (even for the world of microscopy), which means, that its accuracy will be more than enough for other not so demanding fields.

9. Future development & Recommendations

For the system to grow in to a robust commercial product, there are some more features that need to be done.

- An advanced and easy to use graphical user interface, allowing the user to perform various types of measurements on the scanned object is required, in order to get the best use out of the system.
- A faster way to calculate the depth map should be found, in order to reduce the processing time.
- All processing, analysis and plotting parts should be transferred to the embedded device (server) in order to get rid of the need of a PC.

renere

- A better external light source is required, in order to take images with lower ISO values and avoid shadows/glare, thus reduce the noise levels and simplify processing of the image stack.
- A motorized height adjustment stem for the microscope camera is required in order to avoid the need to recalibrate the whole system when camera height changes.
- System accuracy should be recalculated on a bigger number of components, to increase its correctness.

10. References

Pattern Recognition Letters, Volume 5, Issue 1, January 1987 P. Grossmann,. Depth from defocus

Wikipedia 2017. Acutance. [online] Available at: https://en.wikipedia.org/wiki/Digital_microscope

Wikipedia 2017. Depth map. [online] Available at: https://en.wikipedia.org/wiki/Depth_map

Wikipedia 2017. Acutance. [online] Available at: https://en.wikipedia.org/wiki/Acutance

Fielding, Roy Thomas (2000). "Architectural Styles and the Design of Network-based Software Architectures". Dissertation. University of California, Irvine.

US National Library of Medicine, 2011 Aug 25, Xin Xu, Yinglin Wang, Jinshan Tang, Xiaolong Zhang and Xiaoming Liu. Robust Automatic Focus Algorithm for Low Contrast Images Using a New Contrast Measure

Scientific Imaging Group, Cambridge University, May 2001, C.F. Batten, D.M. Holburn, B.C. Breton, N.H.M. Caldwell. Sharpness Search Algorithms for Automatic Focusing in the Scanning Electron Microscope

11. Appendix

Appendix 1 – Server source code documentation, in form of HTML files. Generated by Doxygen.
Appendix 2 – Test cases for the server part in form on an .xlsx table.
Appendix 3 – Software requirements specification.