VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

RECONSTRUCTION OF HUMAN HAND SURFACE FOR THE PURPOSE OF BIOMETRIC RECOGNITION OF PERSONS

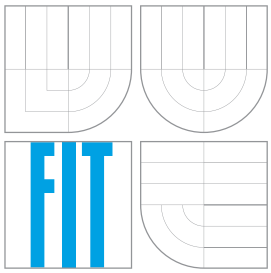
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN SVOBODA

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

REKONSTRUKCE POVRCHU LIDSKÉ RUKY PRO BIOMETRICKÉ ROZPOZNÁVÁNÍ OSOB

RECONSTRUCTION OF HUMAN HAND SURFACE FOR THE PURPOSE OF BIOMETRIC RECOGNITION OF PERSONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN SVOBODA
AUTHOR

VEDOUCÍ PRÁCE — Ing. DLUHOŠ ONDŘEJ
SUPERVISOR

BRNO 2012

Abstract

The goal of this work is to create a device and design an algorithm for the recostruction of human hand surface for the purpose of biometric recognition of persons. First of all existing approaches to human hand surface reconstruction are introduced. Subsequently the theoretical base of the methods used in the solution is shown. Also, the device construction is presented in more detail to the reader. After that the reconstruction algorithm implementation is described. In the end, experiments and results are presented.

Abstrakt

Cílem této práce je zkonstruovat zařízení a navrhnout algorimus pro rekonstrukci povrchu lidksé ruky pro účely biometrického rozpoznávání osob. Úvodem jsou krátce představeny existující přístupy k rekonstrukci povrchu lidské ruky. Následně je uveden teoretický základ metod užitých při řešení. Čtenář je rovněž blíže seznámen s konstrukcí zařízení. Poté je popsána implementace rekonstrukčního algoritmu . V závěru jsou prezentovány experimenty a dosažené výsledky.

Keywords

surface reconstruction, hand geometry, 3D hand, 3D scanner, laser scanner, hand biometrics, OpenCV

Klíčová slova

rekonstrukce povrchu, geometrie ruky, 3D ruka, 3D skener, laserový skener, biometrie ruky, OpenCV

Citace

Jan Svoboda: Reconstruction of Human Hand Surface for the Purpose of Biometric Recognition of Persons, bakalářská práce, Brno, FIT VUT v Brně, 2012

Reconstruction of Human Hand Surface for the Purpose of Biometric Recognition of Persons

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Ondřeje Dluhoše

.....
Jan Svoboda
May 16, 2012

Poděkování

Poděkování patří Ing. Ondřeji Dluhošovi za odborné vedení této práce, dále také Ing. Radimu Dvořákovi a dalším členům výzkumné skupiny STRaDe za velmi cenné rady a příspěvky.

Contents

Chapter 1

Introduction

Humand hand surface reconstruction in 3D space is problem from the field of computer vision and image processing. Nowadays there are many solutions, some of them are already being used in industry, mainly in medicine. Most of these devices are more general and consider any object instead of just human hand, which brings more problems and limitations to the algorithm design that need to be taken into account. The general approach and high requirements increase the cost usually into the values of 10^5 or 10^6 czk.

This solution is restricted to be used only for the human hand geometry, thus problems like reconstruction of general object can be neglected and some specific parts of the human hand can be of use. There is not even need to reconstruct the whole 3D surface of a human hand nor to obtain the volumetric data, only the ridge part is to be reconstructed. These model are often called 2.5D, because about half of the object surface is known.

Because the result is supposed to be used for the biometric recognition of people according to their hand geometry, important aspects are speed and precision of the reconstruction. The surface is constructed using only one image of the scene that is illuminated by structured light pattern, which increases the speed of verification. To achieve high precision of the resulting model, calibration of the device has to be done before it is used.

Another key aspect is the price. This is important especially when the device is about to be published. Because of that, only one camera and structured lighting is used. If the parts of the device are selected appropriately, this is usually cheaper than for example stereo reconstruction.

The theoretical basics of the approaches used in this solution are discussed in chapter 2. Camera calibration method is described, possibilities for the human hand detection in the image are shown. Reconstruction methods are introduced afterwards. Last but not least some theory regarding the device construction is mentioned.

Chapter 3 is focused on the solution concept. First, the device used for the reconstruction is described. Then the whole calibration process is presented. And hand detection solution is shown next. In connection to the hand detection, the structured light pattern detection is discussed in the following section. At the end of this chapter, reconstruction method is introduced.

4th chapter documents some implementation details. At the begining, used libraries are listed. The overall code structure follows. After that all the important parts of the application are described in particular sections.

Last chapter presents the experiments that have been used to evaluate precision of the reconstruction. Possible problems during the reconstruction are presented. At the end, possible improvements are proposed.

Chapter 2

Theory Basis

In order to create the 3D model of an object, it is usually needed to know more that just 3D reconstruction algorithms. The issue covers wider area of computer vision such as image segmentation, which is used to find an object of interest in the input 2D image or detect the stripes regarding some of the structured lighting methods. Another very important part is the calibration. Without camera calibration and without knowledge of the system parameters(e.g. position of the camera, position of the projector of laser) you are not able to get acceptable results. For example the precision of the resulting 3D model is always much influenced by how well the system is calibrated. Last but not least is the diffraction effect, which I used to get more parallel stripes from one laser projected line.

2.1 Biometric recognition of persons

Recognition of persons is based on the unique identity of each person. There are two basic types of identity, the physical one and the eletronic one. Each person has only one physical identity, which is determined by the appearance and the behavior. The eletronic identity is totally different. Amount of eletronic identities of a person is unlimited(e.g. email accounts, ID cards, etc.).

Related to identity, identification and verification has to be distinguished. Identification(1:N) is used in order to obtain identity of a person. Biometric property of the person is passed to the system, but not it's identity. On the other hand, verification(1:1) means that the person provides its biometric property to the system and based on this biometric property, the identity of the person is verified.

Biometry

Biometry has many meanings in different fields. In the field of IT, it means automated recognition of human beings based on their anatomical(e.g. fingerprints, iris, retina, hand geometry, etc.) and behavioral properties(e.g. speech, signature, etc.). Speaking about anatomical properties(also called static properties), it can be said that each property is fixed. Regarding the behavioral(or dynamic) properties, the previous statement is not true. Dynamic properties are easily influenced by condition of the person being recognized etc. [2]

Biometry in general is a problematic field also due to the problems like interclass and intraclass variability[2].

Biometric systems

Biometric system, in general, consists of 5 basic blocks. In particular, these are data capture, signal processing, data storage, comparison block and decision block. Detailed description of these blocks is in [2, 14].

A user can make a statement regarding his identity which falls into one of the 4 following groups:

- `positive statement about identity`;
- `negative statement about identity`;
- `explicit statement about identity`;
- `implicit statement about identity`.

as stated in [2, 14], where more about each particular group can be found.

After the data user provided are processed, biometric system comes to the phase of feature extraction, where the important biometric properties are extracted from the input biometric data. Extracted features are compared with a template and as a result, so called match score, which stands for rate of match, is returned.

Result of comparison in the biometric system is based on the threshold T, which defines, whether the match score should be interpreted as match or no match. The result of the comparison is either `accept` or `reject`.

Based on the threshold set and the match score, biometric systems can make mistakes. Behavior of the biometric system given input data that has to be identified/verified can according to the [2] be one of:

- `true accept`,
- `true reject`,
- `false accept`,
- `false reject`.

For further information, see [2].

The reliability of each biometric systems can be described by few error rates. These are:

- `FAR - False Acceptance Rate`,
- `FRR - False Rejection Rate`,
- `FMR - False Match Rate`,
- `FNMR - False Non-Match Rate`,
- `FTA - Failure to Acquire`,
- `FTE - Failure to Enroll`,
- `FTM - Failure to Match`.

Each of the presented rates can be visualized for better understanding. Examples can be found in [2]. Based on these rates, testing of the biometric systems can be performed.

When it is decided which biometric system to use in a particular case, some other properties of the biometric systems also has to be considered. Basic properties are universality, uniqueness, uniformity, performance, achievability, acceptance, resistance to falsification, cost. You can learn more about these properties in [2, 14].

Hand geometry biometric recognition

In case the geometry of the human hand is used for the biometric recognition, two basic approaches can be distinguished. It can be either 2D hand geometry recognition or 3D hand geometry recognition. These methods can be used especially in the cases where use of e.g. fingerprints is merely possible. For the recognition purposes, following features can be used:

- `finger length`,
- `finger width`,
- `finger height`,
- `curvature and local anomalies`.

In case of 2D hand geometry, methods based on various approaches such as the ones called `direct measurements` or `finger width analysis` can be used.

Regarding 3D hand geometry, more entropy is available since the 3D space provides more information than the 2D one. First, the 3D surface is reconstructed, which is the task of this thesis. Next, important features are extracted from this surface. Nowadays surfaces of all 5 fingers are used. At first, fingers has to be detected and their orientation in 3D space obtained. Then the widths and average curvatures of the fingers in various segments can be measured. These data serves as the features for the template describing the hand geometry.

All the information in this section are taken from [2], for detailed information about this and also other biometric recognition methods see the book.

2.2 Image processing

When the input 2D images are obtained, they must be processed to get the information, that are important and useful for the goal that is about to be achieved.

Color space conversions

The image captured by a color camera is usually represented in a RGB color space. Depending on the scene that is captured, an object of interest and an environment it is placed in, other color spaces may have better properties that can make it easier to distinguish between the target object and the rest of the scene.

There are many color spaces. The most common one is RGB. In hardware, another widely used is CMYK, e.g. in printers. More user friendly color spaces are HSV or HLS. For video and image processing YUV or YCrCb are used in many applications. Very well known one is also the grayscale color space.

RGB color space

RGB color space is based on the 3 basic color components - red(R), green(G) and blue(B). By composing these 3 components, other colors can be created. Thus RGB color space uses so called additive color composition.

HSV color space

In HSV color space, 3 main components are hue(H), saturation(S) and value(V). Hue represents the color and it is an angle from 0 to 360 degrees. Saturation is the amount of gray. It goes from 0(the color si gray) to 1(the color is the primary color). The last component called value stands for the brightness of the color.

Grayscale color space

In a grayscale image, every pixel has only one color information - the gray color intensity. These images are composed only from the tones of gray. The lowest intensity means black, the highest one means white. Range of the intensity values depends on the color depth.

Segmentation

Image segmentation is the process of dividing 2D image into multiple regions in order to extract an object from the image, get some more specific information and simplify or speed up further processing. Object of interest usually does not occupy the whole captured image so it is desired to mask out only the part of the image, that the object belongs to. In many cases, image is converted into grayscale color space before being segmented.

Basic techniques of image segmentation can be divided into 4 groups as it is said in [10]. First, pixel-based methods that use only the gray values of the individual pixels. Second, region-based methods analyze values in a larger area. Third, edge-based methods detect the edges first, then they try to follow them. The fourth can be used only if the shape of the target object is known, these methods are called model-based.

Thresholding

Thresholding is basic and simple method of image segmentation that belongs to the group of the pixel-based methods. Input image is converted from the grayscale color space into binary one, where only colors black and white take place. Thresholding is represented by the following equation

$$f(x,y) = \begin{cases} 0 & \text{if } f(x,y) < T \\ 255 & \text{otherwise} \end{cases}$$

, where $f(x,y)$ is a function that represents the 2D image and T stands for the threshold value.

Selection of the threshold value is the crucial part of the thresholding method. When the selection needs to be done automatically, so called iterative algorithm that is a special case of the k-means clustering method, which produces good threshold values.

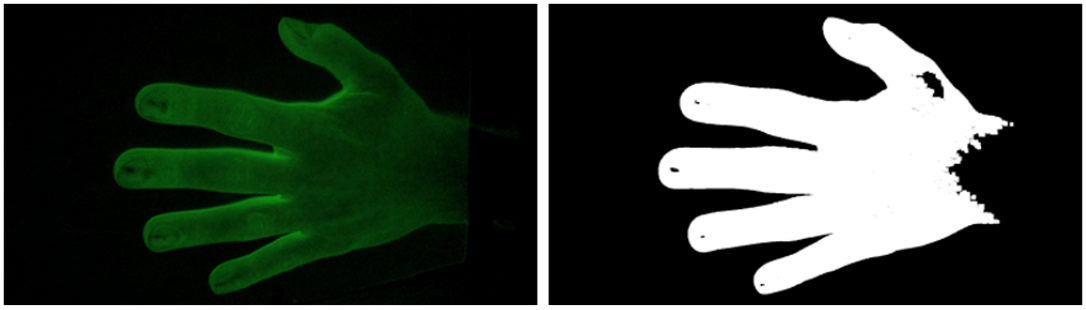


Figure 2.1: The original one is on the left. Thresholding result is shown on the right.

Flood Fill

Another basic segmentation algorithm is called `flood fill`. It is one of the region-based methods based on region growing approach. As an input, the algorithm receives a seed, which is a point in the 2D image. Given the seed, it recursively fills the whole area that this seed belongs to.

Detailed description and possible improvements of this algorithm can be found in [5].

2.3 Camera calibration

For the purpose of 3D reconstruction, we assume pin-hole camera model [12, 15]. Its essential parameters are focal length, the size of the image plane and the principal point. These parameters are often provided by the manufacturer, but they are not precise enough for the 3D reconstruction as stated in [12]. Also, some distortions(e.g. radial distortion) may appear. This is because of the camera manufacturing process imperfections. Due to these, we have to perform the calibration process to achieve higher precision.

The goal of the camera calibration is to obtain the camera `intrinsic parameters`:

- the focal length f;
- the transformation between camera frame coordinates and the pixel coordinates;
- the geometric distortion of the optical system;

and the `extrinsic parameters` that describe transformation between camera and world frame using:

- 3D translation vector T;
- 3 x 3 rotatin matrix R;

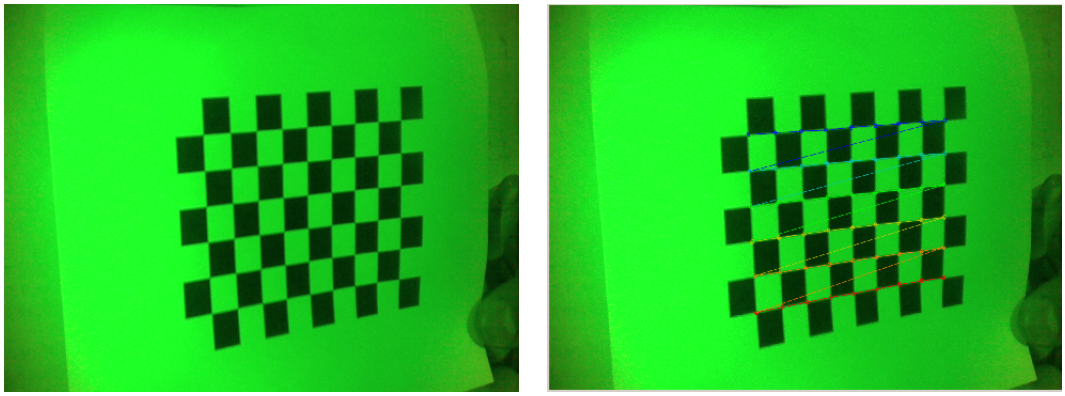
as written in [12].

Calibration methods

Nowadays many approaches to camera calibration such as DLT(Direct Linear Transformation), Tsai's Algorithm, Zhang's method, etc. have been proposed.

Zhang's method

In this method, a planar chessboard pattern is captured in two or more orientations in the scene. From this group of captured images, intrinsic parameters can be solved. The extrinsic parameters can be then solved using only one image of the particular chessboard orientation. As this method is implemented in OpenCV[12] and in the Matlab calibration toolbox[9], it is widely used in the computer vision applications. For deep theoretical desctription of this method, see [16].



(a) (b)

Figure 2.2: Example of the chessboard orientations in the calibration images is shown on image (a). Image (b) shows extracted corners of the chessboard.

2.4 3D reconstruction

According to [11], the reconstruction of object surfaces is a special discipline in computer vision. This discipline is directed towards the recovery of object shapes or towards the calculation of distances between the sensor, i.e. the camera, and objects in the scene. Data acquired with one or more cameras constitute the initial information.

In computer vision shape reconstruction of 3D objects is treated on the basis of visual data obtained by one or more cameras, which reproduce a static or dynamic scene [11] depending on the fact that the image acquisition is either static(nor the objects in the scene, neither the camera are moving) or dynamic(the objects in the scene or the camera are being moved).

Several active and passive image acquisition techniques exist, which are directly oriented towards range determination, or which at least allow for distance determinations. Range relevant or surface relevant data can be specifically embedded in the generated image information by menas of active image acquisition techniques as mentioned in [11].

As stated in [11], image acquisition techniques can be also distinguished into `monocular`(one camera), `binocular`(two cameras), or `polyocular`(several cameras) depending on the number of cameras that are being used.

Methods

One possibility is to use method called stereo analysis, which takes an advantage of multiple cameras observing the same scene from a different viewpoints. Relations between the cam-

eras are evaluated and the distance measurements are performed by means of triangulation that is described in the following section. Second choise can be to use mono or binocular image acquisition and the application of structured lighting. In this case, for example line or grid patterns are projected into the scene and the distance measurements are done using already mentioned triangulation. Another possibility is to use so called photometric stereo analysis that is based on variations of the illumination in the scene. Controlled object or camera movements can be used also. These methods are called shape from motion or shape from occluding boundaries.

More methods and their detailed description is provided in [11].

Triangulation

Contained in [15], the basic geometry for a triangulation system is shown in Figure 2.3. The system consists of a light projector that is placed at a distance b, so called baseline, from the center of projection of a pinhole camera. The center of projection is at the origin of the reference frame XYZ, in which are all the sensor's measurements expressed. f is the focal length. This and other intrinsic parameters can be obtained by performing calibration process. The projector emits a plane of light that is perpendicular to the plane XZ and forming a controlled angle ϕ with the XY plane. The intersection of the plane of light with the scene surfaces is a planar curve called the `stripe`, which is observed by the camera. Coordinates of a stripe point $P = [X, Y, Z]^T$ are given by

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \frac{b}{f \cot \phi - x} \begin{pmatrix} x \\ y \\ f \end{pmatrix} \tag{2.1}$$

.

If we apply this equation to all the visible stripe points in the image, 3D profile of the surface points belonging to the particular stripe is obtained. The same approach can be applied to each stripe visible in the image to get as many object surface information as possible.

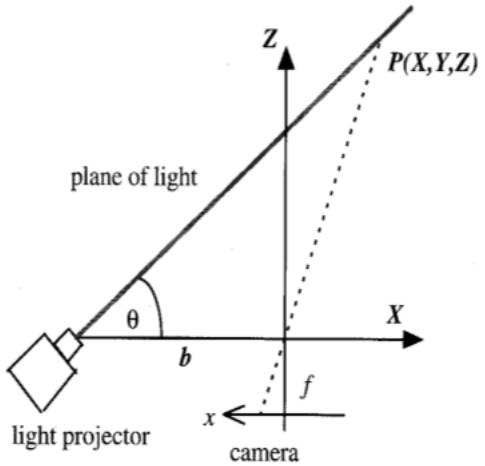


Figure 2.3: Geometry of the basic triangulation system. Planar view XZ is shown. Y and y axes are perpendicular to the plane of this figure.

Structured lighting methods

Structured lighting stands for projection of light patterns into a scene. Patterns are projected onto the objects which lie in the filed of view of the camera. By analyzing the light patterns in the images, distance of an object to the camera or the location of an object in space can be determined. This technique simplifies the general 3D reconstruction task enormously.

In [11] says that these methods can be regarded as a modification of static binocular stereo, where one of the cameras is replaced by a light source which projects the light pattern into the scene. Then the triangulation is carried out by intersecting the projection ray(from the camera) and the light ray or plane(from the light source). The approach of 3D object acquisition using structured lighting can be divided into methods which use simple geometric light patterns such as light spots or light stripes and methods which are based on spatial and/or temporal coding of the light patterns. You can find much more about each particular method in [11].

Static light pattern projection

According to the expalnation in [11], it is possible to project serveral rays or planes at the same time on the object surface instead of just one. This reduces the number of images needed for the reconstruction. Principle of this method is still based on triangulation that was described previously. Usually, dotted lines, parallel lines or dot matrices are used as a pattern.

In this case, an approach to indexing the stripes on the object has to be proposed. After the stripes are indexed, each of them can be processed independently.

2.5 Diffraction

As mentioned in [3], when light is passing an edge, it is deviated from rectilinear propagation. This phenomenon is called diffraction. It is based on the Huygen's principle and can be observed by passing light through a narrow single slit. Also multiple slits can be used. Then the phenomenon is called diffraction from a grating.

Diffraction from a single slit

In this case, the light is passing through a narrow single slit. The intensity in the pattern generated on the screen can be described using following equation:

$$I = I_{max} \cdot \left(\frac{\sin \alpha}{\alpha} \right)^2 \tag{2.2}$$

, where

$$\alpha = \frac{\pi a}{\lambda} \cdot \sin \phi \tag{2.3}$$

.

In equation 2.3, a is the width of the slit, λ is the wavelength of the light and ϕ determines the position of the point on the screen. As we can see from the equations 2.2 and 2.3, in order to have the intensity of the pattern on the screen as homogenous as possible event further away from the center, width of the slit a should be equal to the wavelength of the light λ.

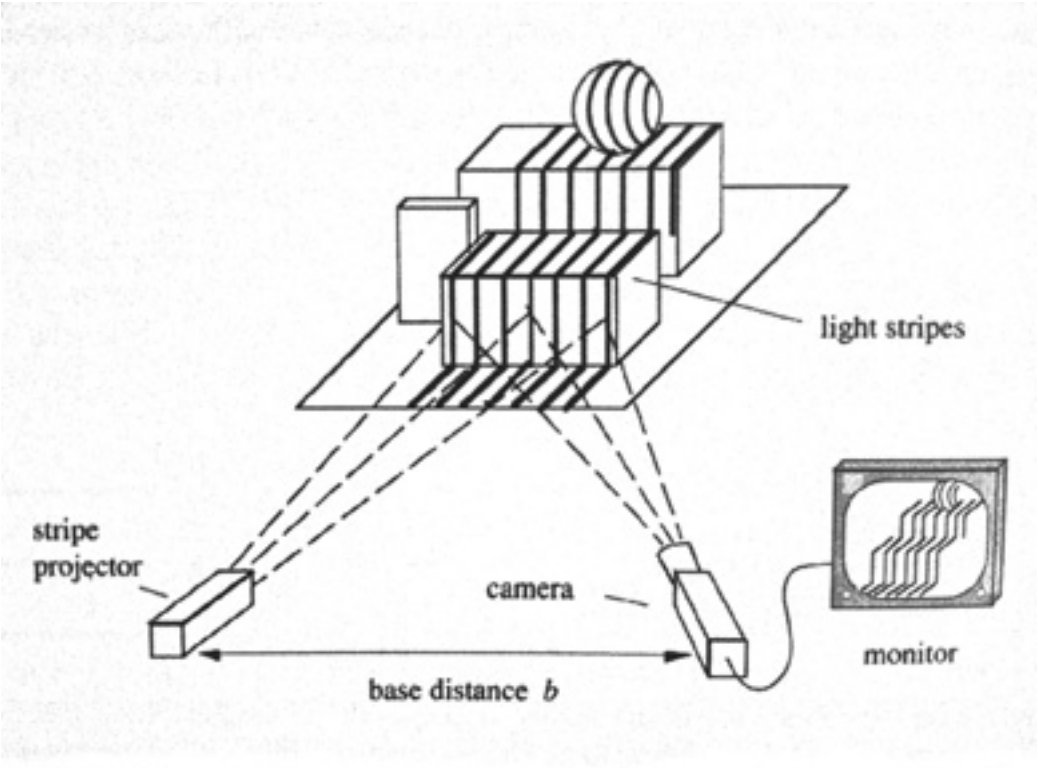


Figure 2.4: Example of the static light pattern projection system. [11]

Diffraction from a double slit

When the light is passing through double slit, there is another phenomenon that needs to be taken into account called interference [8]. In this case, the intensity in the pattern generated on the screen is described by the equation

$$I = I_{max} \cdot \left(\cos^2 \beta\right) \cdot \left(\frac{\sin \alpha}{\alpha}\right)^2 \tag{2.4}$$

, where

$$\beta = \frac{\pi d}{\lambda} \cdot \sin \phi \tag{2.5}$$

.

The λ and ϕ have the same meaning as in the equation 2.3. Finally d is the distance between the centers of the slits. The $\cos(2\beta)$ is so called interference factor [8] and $(sin(\alpha)/\alpha)^2$ is the diffraction factor [8].

Diffraction from a grating

As stated in [8], since diffraction grating is a multiple-slit plate, the maxima occur at exactly the same position as in the case of a double slit interference. However, the bright fringes are sharper and brighter.

When it comes to a diffraction gratings, the principles introduced above are followed. So the diffraction pattern on the screen is again the result of the combined effects of interference

and diffraction as it is for the double slit. Each slit causes diffraction, and the diffracted beams in turn interfere with one another to produce the pattern. The path difference between the waves from any two adjacent slits can be found by dropping a perpendicular line between the parallel waves. According to the rules of geometry, the path difference is $d \sin \phi$. In the case this path difference is equal to a wavelength or an integral multiple of a wavelength, waves from all the slits are in a phase and a bright line is observed at that point. Therefore, the condition for maxima in the interference pattern at the angle ϕ is:

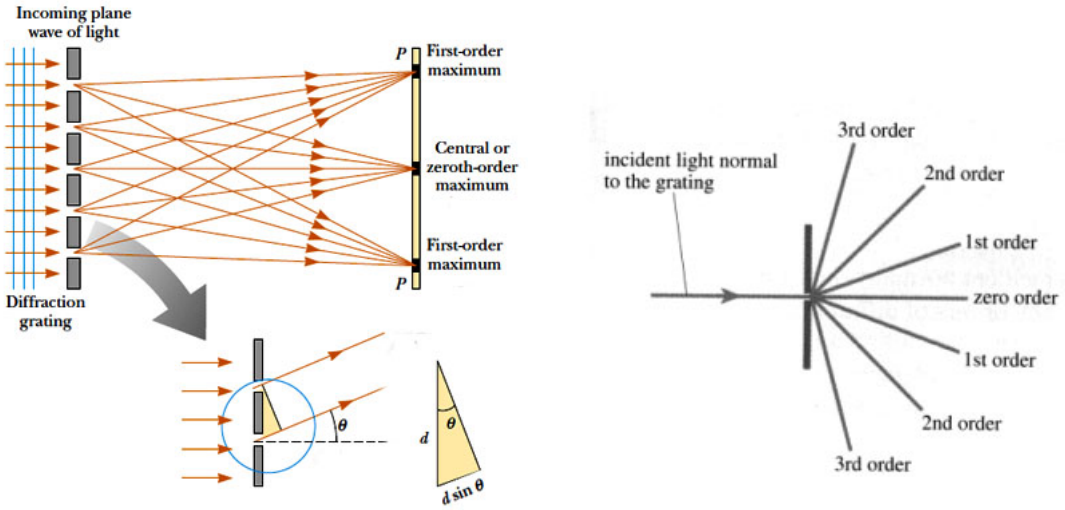
$$m\lambda = d \sin \phi \tag{2.6}$$

,where m is the order number, a positive integer representing the repetition of the spectrum. The rest of the coefficients is already known from the above equations. Another important parameter is the grating constant, which stands for the number of slits per unit length:

$$N = \frac{1}{d} \tag{2.7}$$

For more details about the diffraction, please read [7].

Using the equations 2.2 - 2.7 and considering the application the grating will be used for, all the important parameters can be computed so that the most suitable diffraction grating can be designed and manufactured.



(a) (b)

Figure 2.5: Diffraction from a grating is shown on the image (a). Image (b) shows an example of the interference maxima for particular orders. [7]

Chapter 3

Solution Concept

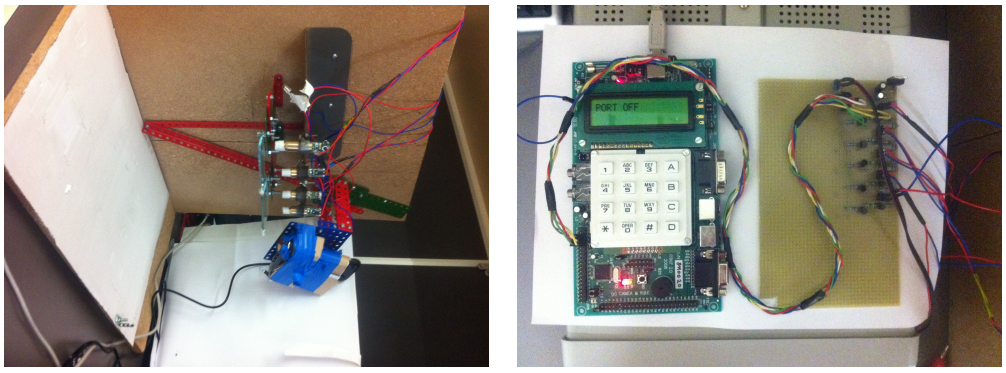
Whole reconstruction process divided into few sections is described in this chapter. The rough overview of the methods that can be used for each particular task has been done in the previous chapter. I will now focus only on the solution specific methods. In the solution, 4 lasers were used. From section 3.4 on, the approaches are the same for all the lasers, so these will be described considering only one of them.

3.1 Device design

Many devices for the 3D reconstruction are already on the market, but I'm trying to desing a new approach and that means that I also need to create a new device, that will fit my requirements the best. Because structured lighting method is used, an emitter of structured light is needed. In this case, I have chosen a line laser module. From this single line, mulitple lines are created with the help of diffraction grating. The emitted pattern is projected onto the screen which is captured by a camera.

Because of the low price requirement, each part was chosen very carefully taking its price into account but also involving the product quality in the decision process.

The device can be seen on the images below. Particular parts are then described in the following sections.



(a) (b)

Figure 3.1: The focused shot of the laser-camera system is shown on image (a). Rest of the device is then presented on image (b) where you can see the FITkit and electrical circuit used as a controller.

LED diodes

For easier hand masking process described later in section 3.4, the scene is illuminated by 3 green LEDs that are directed so that they does not point directly on the hand but they rather distributes the light in the scene, so that it's whole illuminated as homogenously as possible.

The LEDs are powered by 4V source and they require current of about 200mA.

Laser

In the case of laser, I have decided to go for a green laser module. It has good properties when projected onto the human hand even in the normal(not direct) daylight. It is a line module, so it generates a straight line instead of just a point. This effect is achieved by simple optics(lens) attached to the laser module.

The light beam generated by the module has wavelength of 532nm. It is 10mW module which needs 3V power and the current is recommended to be 250mA.

Diffraction grating

As mentioned in the previous section, laser generates single straight line, which is not enough for the method proposed here. Because multiple parallel lines are required, diffraction grating is used to create the desired pattern on the screen.

All the professional or special order gratings are expensive, so I have tried to use a simple one, which is primarily sold for the educational purposes. It has 1000 slits per milimeter and is of a 50x50mm size. This grating is two dimensional, which means that it diffracts the light in two directions, which is of an advantage since I was able to increase the number of the stripes that the diffraction effect produces by rotating the grating 45 degrees around the z-axis.

Another thing is that these simple gratings does not create enough lines. Because of that, 4 lasers and 2 gratings were used(1 grating for 2 lasers is enough).

Device controller

In order to be able to control the lasers and LEDs via computer, FITkit was used. Program that sets/resets specific pins on the FITkit interface using COM port for the communication was designed.

Because these FITkit pins do not provide enough voltage for the lasers and diodes, a specific eletrical circuit was designed that fulfils the task, which require 7V. For each of the lasers and for the LEDs, so called Darlington pair(a pair of transistors) was used to supply the particular laser/LEDs with enough power from an additional power supply. The circuit is based on the inverted logic, which means that when the logical value on the particular FITkit pin is 0, the corresponding laser/LEDs is on, if there's a logical value of 1, it is switched off. For more information regarding the eletric circuit, please see the scheme on the Figure 3.2.

Camera

To keep the price low, I have decided to use an ordinary web camera Logitech QuickCam 9000 Pro that can be found in many shops all over the world. It is not the cheapest of the

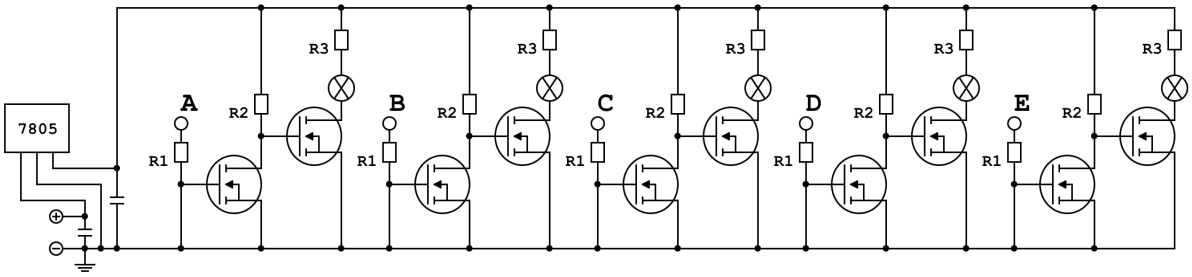


Figure 3.2: Scheme designed in order to provide enough power for the laser and LEDs. The values are $R1 = 560\Omega, R2 = 4700\Omega, R3 = 10\Omega$. Pins that triggers particular lasers/LEDs are connected to the inputs A - E.

kind, but it is higher quality one. My decision was mostly influenced by the fact that this web camera has already been successfuly used in many computer vision projects before.

This color CCD camera is equipped with a high quality Calr-Zeiss optics and provides resolution up to 1600x1200 pixels. It does not have any official Linux driver which is of a little disadvantage, but it does not matter since there are already tested 3rd party Linux drivers for the camera. Connection to the PC is done via USB cable.

3.2 Calibration

Calibration in this case does not mean only camera calibration. Because structured lighting method is used, the light emitter(laser in this case) also has to be calibrated for the purpose of triangulation. Unfortunately since high precision is required, the parameters cannot be obtained by measurements, but they have to be computed via computer.

Camera intrinsic and extrinsic parameters

For the camera parameters, OpenCV library calibration functions can be used. The OpenCV library uses Zhang's calibration method mentioned in section 2.3. Twelve images are used for the calibration in this case, so the planar chessboard is captured in twelve different orientations in the scene. From these twelve images, taking an advantage of OpenCV, corners on the chessboard are detected first and these data are then passed to another OpenCV function that does the calibration. The calibration function returns both intrinsic and extrinsic camera parameters.

As the extrinsic parameters are returned for each particular chessboard orientation, the one that is coincident with the reference plane has to be used. This is always the orientation corresponding to the first image that is captured. Assuming this, the first T vector and R matrix returned by the OpenCV calibration function are taken as the camera extrinsic paameters.

Having T vector and R matrix, it is now possible to compute the 3D orientation of a chessboard points. As the chessboard on the first image is always centered with respect to the line made by the diffraction grating 0th order of diffraction(which corresponds to the laser center), the chessboard central point can be chosen and its 3D world coordinates can be computed. By doing this, a 3D point lying on the backplane is obtained. Camera is

assumed to be at point $C = [0, 0, 0]$ in the world coordinate system, hence both points can be aligned so that the central backplane point will be at $P = [0, 0, 0]$ by performing the following

$$P = P - P = [0, 0, 0] \tag{3.1}$$

$$C = C - P = -P \tag{3.2}$$

.

Using this alignment, the diffraction grating 3D position L can be computed very easily later on by obtaining only its z-coordinate $L.z$, which is the only important one in this case, as

$$L = P - L = [0, 0, -L.z] \tag{3.3}$$

, where the subtraction is made due to the fact, that the camera position C z-coordinate is also negative value and both C and L has to be on the same side of the backplane. The computation of the $L.z$ value is described in the following subsection.

Diffraction grating and lasers

Not only the camera parameters has to be computed. The diffraction grating position is needed too. A method for the estimation of the diffraction grating position is based on observation of the projected laser stripes from two different distances. First distance is the backplane distance, hence the stripes are only projected on the backplane. For the second distance, a rectangular calibration object with known parameters that occupies the whole scene is used. The object is placed in the scene and the laser stripes are projected onto that object. It is important to say that since this calibration is done using camera, the camera calibration has to be done first. Summary of the laser calibration images is the following:

- `laser1CalCloser` - image with the stripes projected on the calibration object;
- `laserCalCloserBg` - image with the calibration object without projected stripes;
- `laser1CalCloserChess` - chessboard placed on the calibration object with the central projected laser line aligned with the center of the chessboard;
- `laser1CalFurther` - image with the stripes projected on the backplane;
- `laserCalFurtherBg` - image of the backplane without projected stripes;
- `laser1CalFurtherChess` - chessboard placed on the backplane with the central projected laser line aligned with the center of the chessboard.

The images with the suffix `Chess` allow to compute the pixels to milimeters ratio and the perspective projection homography of the camera view, as the camera is viewing the scene from a certain angle. It has to be computed separately for each calibration image because these values are changing with the changes of the distance to the camera.

Calibration images preprocessing

Images with the suffix `Bg` are then used for easier line extraction when the input images are processed. Before the other laser calibration images are processed, their inverse perspective transformation using the computed homography is done for each of them so that the trapezoidal distortion due to the perspective projection is corrected.

Grating position estimation

The following steps are done for both calibration images. At first the images are filtered using stripe masking technique described later in section 3.4. Second, line detection algorithm is used on the calibration image in order to detect the projected stripes. Distance between each two neighbor lines is then computed. From these distances, the average is computed. The corresponding pixels to milimeters ration is used to convert the average distance into milimeters.

As the average neighbor line distance in milimeters for each calibration image is now obtained, the distance from the diffractin grating to the screen can be computed easily using this equation

$$l = \frac{h}{1 - \frac{y}{x}} \tag{3.4}$$

, where h is the height of the calibration object, y is the average distance between neighbour lines projected on the calibration object and x is the average distance between neighbour lines projected on the backplane. The laser calibration setup is shown in Figure 3.3.

After the distance from the diffraction grating to the backplane is known, the diffraction grating position is set to point $G = [0, 0, l]$.

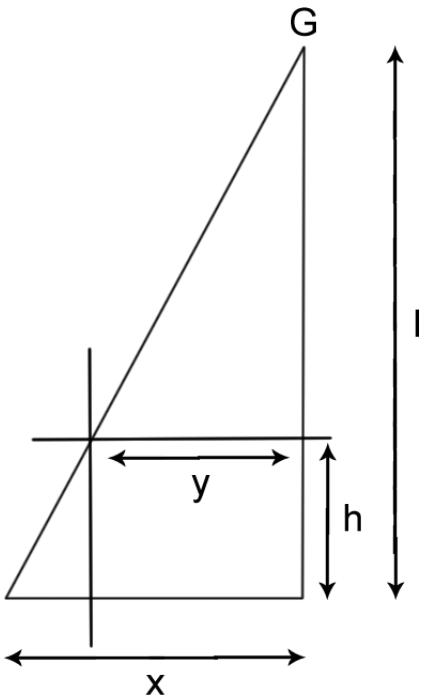


Figure 3.3: Laser calibration system setup. G is the diffraction grating position, l is the distance from the diffraction grating to the backplane and h is the height of the calibration object. Parameters x and y are distances between two neighbour projected stripes with respect to the distance from the grating.

Laser central stripes and x-coordinate offsets

As the last thing during the calibration process, the central stripe of each laser has to be detected. To make this task easier and more precise, laser calibration plane is created,

which has 4 lines drawn on it. These lines represents the first estimates of the central stripe position for each laser.

So it is important to take 5 images in this case, namely:

- `laserFirstEstimates` - image with the created laser calibration plane;
- `laser0CalFurther` - image with the stripes projected by the 1st laser on the laser calibration plane;
- `laser1CalFurther` - image with the stripes projected by the 2nd laser on the laser calibration plane;
- `laser2CalFurther` - image with the stripes projected by the 3rd laser on the laser calibration plane;
- `laser3CalFurther` - image with the stripes projected by the 4th laser on the laser calibration plane.

Having these images, the first estimate of the laser central stripe is obtained for all 4 lasers from image `laserFirstEstimates`. Then the following is done for all 4 lasers. Projected stripes are extracted from the image `laserXCalFurther`, where `X` stands for the index of the laser. Stripe that is closest to the first estimate of the particular laser is taken as the central one.

Now since the central stripe position for each of the lasers is known, the x-coordinate offset of each laser can be computed straightforwardly, assuming that the laser with the index 1 is the one with x-coordinate of 0 as in the geometry of the whole system, this laser is considered the central one.

Reference images capture

Acquiring the system geometrical parameters is not the only task. Important are also the reference images that are used later on during the reconstruction phase. So their capture is also desired, because as they are captured and saved only once during calibration, there is no need to loose time by their repetitive capture during each particular hand input. This makes the reconstruction process faster.

Two reference images are captured, namely it is:

- `bgNoHand` - image of the plain background without hand input;
- `stripesBg0` - image of the background illuminated by the structured light pattern made by the 1st laser without hand input;
- `stripesBg1` - image of the background illuminated by the structured light pattern made by the 2nd laser without hand input;
- `stripesBg2` - image of the background illuminated by the structured light pattern made by the 3rd laser without hand input;
- `stripesBg3` - image of the background illuminated by the structured light pattern made by the 4th laser without hand input.

The first one, `bgNoHand`, is used during the hand detection phase that is described later. The other ones, `stripesBgX`, are important for the reconstruction and are also presented later, specificaly in the section 3.4.

3.3 Capture phase

With every hand input, capture phase has to be started. The goal is to obtain two images that are needed for the surface reconstruction, specifically:

- `bgHand` - image of the scene with a human hand;
- `stripesHand0` - image of the scene with a human hand illuminated by the structured light pattern made by the 1st laser;
- `stripesHand1` - image of the scene with a human hand illuminated by the structured light pattern made by the 2nd laser;
- `stripesHand2` - image of the scene with a human hand illuminated by the structured light pattern made by the 3rd laser;
- `stripesHand3` - image of the scene with a human hand illuminated by the structured light pattern made by the 4th laser.

The `bgHand` image is captured for the purpose of hand detection described in the next section. The rest of the images, `stripesHandX`, are the main images used for the reconstruction. Their processing is described later on.

3.4 Image segmentation

After the images are captured, segmentation needs to be done to find the object of interest and desired image features. The object of interest is the human hand placed in the scene. By features, I mean the structured light pattern visible in the image.

Hand detection

Hand detection is done in order to detect the object of interest and narrow down the region of further processing. Thanks to the device that has been designed, the task is really easy and as it was already mentioned, simple thresholding algorithms can be used. Before the thresholding is applied, the background image is subtracted from the image with the hand present which separates the hand from the rest of the scene very well. After that, the thresholding itself is done which creates the hand mask. To find a suitable threshold, iterative algorithm mentioned earlier in section 2.2 is used. After the image is thresholded, morphological operation called `open` is applied to unify the region occupied by the hand. The whole process is illustrated in Figure 3.4.

Structured light pattern detection

Structured light pattern has to be detected in two cases. First is during the calibration, when it is projected on the plain background. There is not any masking that would precede the detection. Then the second case is during each hand input, when the pattern is projected on the present hand. When there is a hand in the scene, it is required to first mask the region occupied by the hand and use only this region in the further processing. This fact speeds the reconstruction up a lot since the pure background parts of the image are not processed anymore.

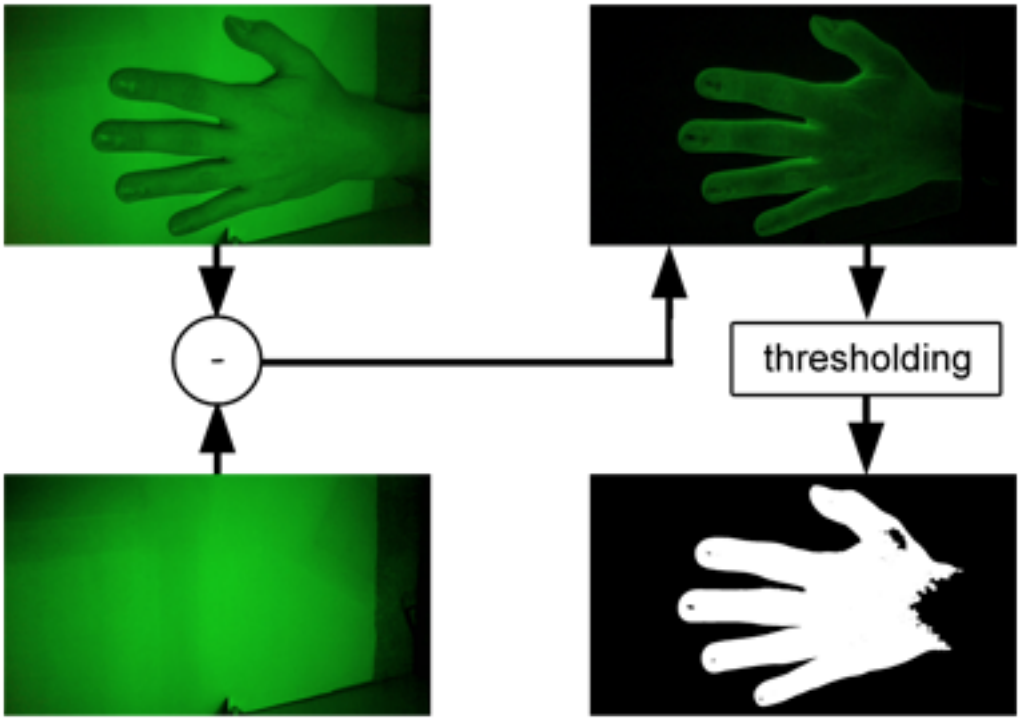


Figure 3.4: Hand masking processing pipeline.

The hand detection algorithm is a bit different in this case. As the first step, vertical edges are detected by convolving the image with the kernel

$$\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \tag{3.5}$$

.

After the vertical edges are detected, image is split into color channels R, G and B, where the G channel is used for the further processing. Considering the G(green) channel, adaptive thresholding is applied in order to filter the stripes. In the end, median blur is applied to remove some noisy pixels or small regions from the resulting image. See the illustration of this process in Figure 3.5.

3.5 Stripe indexing

Problem of stripe indexing comes with use of static projection of a multiple parallel lines. It is the bottleneck of the chosen approach since it has to be done as fast as posible but also precisely. As a result of bad stripe indexing, errors in the final 3D model would be produced. Algorithm proposed for the stripe indexing is described in the following subsection.

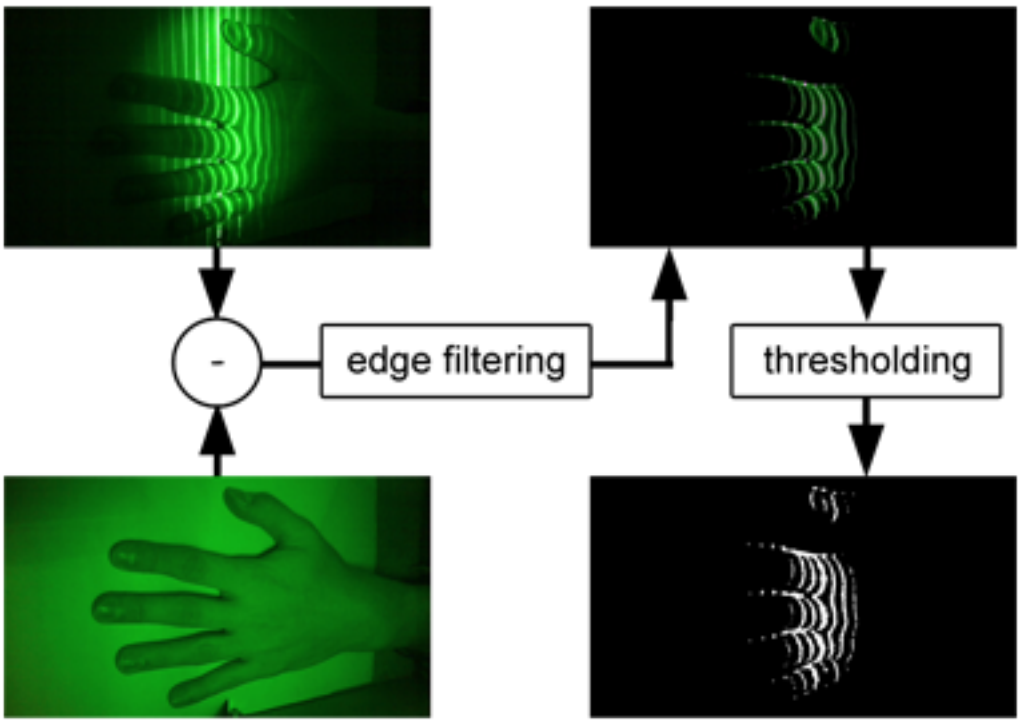


Figure 3.5: Stripe masking processing pipeline.

Indexing algorithm

As the input, algorithm gets an image that has already been masked, so that only the region of interest is further processed and only the structured light pattern is visible in this region. This region is searched vertically column by column. When a white point is found, which means that it belongs to a stripe, flood fill algorithm is used to mark all the pixels belonging to the same area. If the area is too small, it is thrown away. Otherwise, a boundary of the area is extracted and marked as a stripe, then it is connected as a new node to the tree that defines the hierarchy of all stripes on the object. Each level of the tree stands for one stripe index, so the nodes at the same level has the same stripe index - they belong to the same stripe. This tree is in the end returned as an output. Example of the output hierarchy is shown in figure 3.6. See Algorithm 1 for the pseudocode.

Indexing error corrections

In order to reduce the amount of badly indexed stripes after the stripe indexing process is done, correction procedure has been proposed. Assuming that the distance between two neighbour stripes should be in some range(hand surface is quite smooth), it goes through the tree of stripe nodes and checks the distance between each stripe and its predecessor. If it is greater than some specific value V, the stripe is probably indexed badly and the indexing should be corrected. From the distance to its predecessor, number of missing stripes between the two neighbours is computed and from that value, new index is deduced. This change

Algorithm 1 Indexing of the stripes in the image

```
1: for all columns in the image do
2:   for all points in the current column do
3:     if point belongs to a stripe then
4:       FloodFill() {Match whole stripe area}
5:       if surface too small then
6:         Skip it {Probably not a stripe}
7:       end if{Otherwise it is probably a stripe}
8:       for all rows in the stripe area do
9:         Find stripe area edge point
10:      end for
11:      Make stripe from the found edge points
12:      Add stripe to the tree {Indices are assigned according to the hierarchy}
13:    end if
14:  end for
15: end for
```

of the index has to be propagated to all children of the particular stripe node. Algorithm 2 describes the process using pseudocode.

Algorithm 2 Stripe indexing errors corrections

```
1: for all stripe nodes in the model do
2:   if currStripeXCoor - prevStripeXCoor > V then
3:     Compute new index of the current stripe
4:     Propagate the change to all the children of the current stripe node
5:   end if
6: end for
```

Stripe indices alignment

Because the number of projected stripes from a particular laser visible on the background and on the hand does not have to be the same(e.g. both have different reflection properties, etc.), all the stripe indices has to be related to some feature that is the same in both images all the time. Because the central projected stripe, that corresponds to the 0th order of diffraction, has always the highest intensity of all the stripes, it can be considered the feature and all the other stripes can be indexed relatively to the one with the highest intensity. This ensures that the same stripe on the hand and on the background can be easily found using the relative indices.

On the background, the central stripes were already found during the laser calibration process so there's nothing more to be done in order to find them.

On the hand, the approach described regarding laser calibration cannot be used. In this case, average intensity of the image points belonging to a particular stripe is computed for each stripe. Then the stripe with the highest average intensity is marked as the central one.

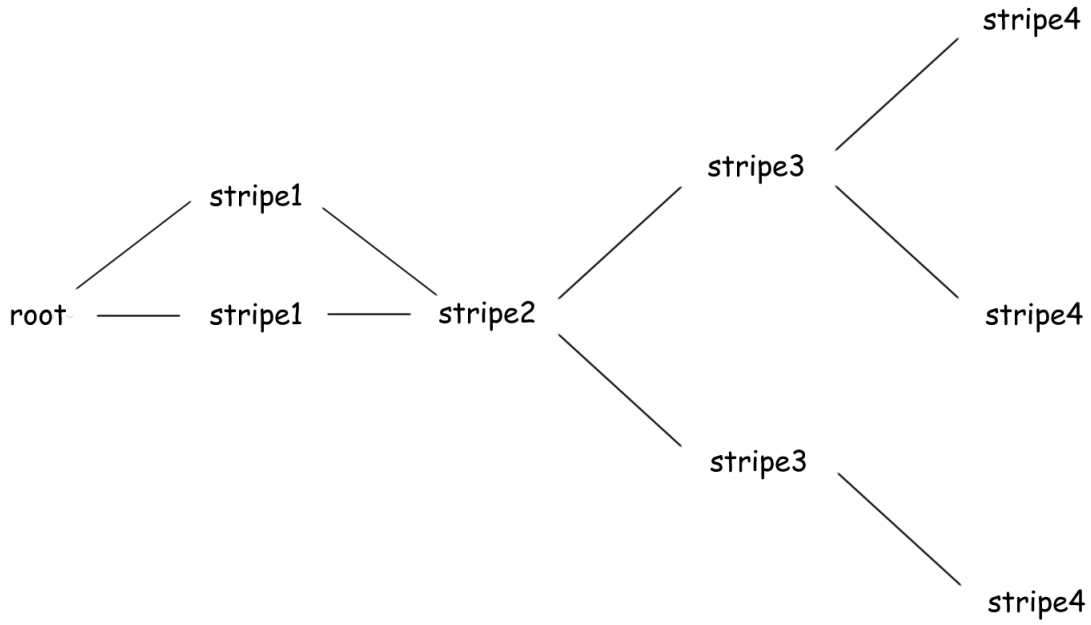


Figure 3.6: Example of the tree hierarchy that can be created by the proposed indexing algorithm.

3.6 Model reconstruction

At this point, all the necessary preprocessing is done and everything is prepared for the surface reconstruction itself. System is already calibrated, reference frames are processed so the background model is available and also, the hand images are already preprocessed and stripe indexing has already been carried out.

For the purpose of reconstruction, triangulation principle described in section 2.4 is considered and used. As the camera position and the grating position are already known, the 3D point can be computed by means of ray - plane intersection. For each point in the image corresponding to a stripe, a ray from the camera is casted that goes through this point into the scene. At the same time, a plane is created knowing a line(stripe in the image) and a point(the diffraction grating position). By computing the intersection of the casted camera ray and the laser stripe plane, a point in 3D space is obtained that is actually the final surface point. An algorithm has been proposed that encapsulates the reconstruction task. It is described in the next subsection.

Reconstruction algorithm

Designed algorithm goes through the indexed stripes from the hand image. For each stripe, its index relatively to the central one is got. Using this relative index, a stripe with the corresponding relative index in the background model is found. Having the corresponding background and hand stripes, the algorithm goes through all the points on the particular hand stripe and computes their 3D positions knowing the plane that is specified by the laser position and background stripe and ray that is specified by the camera position and the current point on the hand stripe. Simplified pseudocode of this algorithm is below.

In the above algorithm, the `bgStripe` is the stripe in the background model correspond-

Algorithm 3 Surface reconstruction using indexed stripes

```
1: for all stripe nodes in the model do
2:   Get current stripe node index related to the central hand stripe
3:   for all points on the current stripe do
4:     if there is a stripe with the corresponding relative index in the background model
       then
5:       Compute3DPos(handPt, bgStripe) {Compute the 3D coordinates using ray-
         plane triangulation}
6:     end if
7:   end for
8: end for
```

ing to the current stripe on the hand. The `handPt` is the currently processed point on a hand stripe.

Surface error correction

Reconstructed surface model is influenced by error due to the inaccuracies in the geometry of the assembled device and to the fact that camera is viewing the scene from a certain angle. This error can be approximated by 2D function considering xz-plane in the 3D space. Since the y-coordinate is the least important one during the reconstruction process, it can be excluded.

To find the error function, sufficient amount of testing data has to be acquired first. These testing data are usually rectangular objects of known proportions. Knowing the real height of these testing objects h, reconstruction error in the z-coordinate $zErr$ for each surface point P can be computed easily as

$$zErr = (P.z - h)/P.z \tag{3.6}$$

, where $P.z$ stands for the z-coordinate of the particular surface point. Having $P.x$(x-coordinate), $P.z$ and $zErr$ values for each surface point, the $zErr$ values can be approximated by 2D function of second order $e(x, z)$, that has the $P.x$ and $P.z$ values as variables, in the following form

$$e(x, z) = c1 + c2 \cdot x + c3 \cdot z + c4 \cdot x^2 + c5 \cdot x \cdot z + c6 \cdot z^2 \tag{3.7}$$

, where $c1$, $c2$, $c3$, $c4$, $c5$, $c6$ are constant values obtained during approximation.

It is known that each reconstructed point P should have the z-coordinate $P.z$ equal to height of the real object h, thus we can separate the h from the Equation 3.6 that is

$$h = P.z \cdot (1 - zErr) \tag{3.8}$$

.

As the zErr is approximated by the function $e(x, z)$, considering that the h is the correct height of the object, corrected z-coordinate $corrP.z$ of the surface point P can be computed as

$$corrP.z = P.z \cdot (1 - e(P.x, P.z)) \tag{3.9}$$

.

Chapter 4

Implementation Details

Approaches specific to the approach proposed in this publication has been described in the previous chapter. This chapter lists the used libraries, documents the code structure and emphasizes important implementation specific details.

4.1 Libraries

Several libraries were used in this project. In order to keep the program platform independent and to take advantage of managed pointers, `Poco` C++ library was used. For the purpose of image processing I have used well-known library called `OpenCV`. `OpenGL` library has taken care of displaying the reconstructed model. To make it easier for the user to operate the program, user interface created with `Qt` library is provided.

Poco

Poco is a set of powerful open source C++ class libraries and frameworks that can be used for building cross-compatible network and internet applications. It is also a good choice when designing an application for an embedded device. It provides a lot of useful stuff, e.g. classes for dynamic typing, events and notifications, smart pointers and memory management, etc. Thanks to the good documentation, it is quite easy to learn and use. For more information, visit [4].

OpenCV

OpenCV stands for Open Source Computer Vision and is a powerful library for programming image processing and computer vision applications. It provides functions for image transformation, image segmentation, feature detection, camera calibration and much more. You can find more about OpenCV at [13].

OpenGL

OpenGL(Open Graphics Library) is a cross-platform application programming interface that can be used to create interactive 2D and 3D graphics applications. OpenGL supports programmers with wide set of rendering, texture mapping, special effects and many other visualization functions. For further reading, I suggest [6].

Qt

As stated at [1], Qt is an application development framework based on C++ that provides a huge set of classes that can ease the programmers work. Apart from that, it also allows to create complex GUI application with an attractive design. Good thing about Qt is, that it also supports integration of OpenCV or OpenGL libraries, which makes it a good choice when creating a GUI for a computer vision application. In order to learn more, visit Qt web pages [1].

4.2 Code structure

In this section, the overall code structure is described. Application is separated into three main parts. The most important one is the `Application Core`. It takes care of the core functionality e.g. the calibration, image processing, reconstruction, etc.. Another part is `Application Logic and GUI`. Functions provided by the core are put together to acomplish the tasks they were designed for and all the functionality is presented to the user via simple graphical interface. The last one is `Device Controller` and it provides an interface for controlling the created device. Whole code is well structured using namespaces that divides it logically. Important parts of the core are described in the following sections that presents the most important namespaces and their classes. Each section corresponds to one namespace. Not Logic and GUI nor the Device Controller are described in detail since it's not of that big importance and can be designed in many ways.

4.3 Model

Classes for the surface model data storage are provided by this namespace. `SurfaceModel` class uses the other classes to describe the surface structure. The surface model is structured using n-ary tree. Each node is represented by `StripeNode` class. This class, apart from the others, defines an instance of `StripeSegment` class, which represents the stripe in the image that the node corresponds to. StripeSegment class maintans a list of stripe points, where each point is represented by `StripePoint` class.

StripePoint class

Describes 2D image point and 3D scene point correspondence by encapsulating data of both. Defines four neighbour stripe points to build a surface model structure. It also stores information about the index of a stripe it belongs to and index of a ray casted from the camera that it lies on.

```
class StripePoint
{
        ...
        //! StripePoint position
        cv::Point2d point;
        //! StripePoint 3D position
        cv::Point3d point3D;
        ...
        //! StripePoint ray index
        int rayIndex;
```

```
        //! StripePoint index
        unsigned int index;
        ...
        //! Neighbour points
        StripePoint::Ptr ptLeft;
        StripePoint::Ptr ptRight;
        StripePoint::Ptr ptTop;
        StripePoint::Ptr ptBottom;
        ...
};
```

StripeSegment class

Maintains list of stripe points that lies on the stripe and provides functionality for stripe smoothing. It keeps the index of the stripe. Also has an information about the stripe extreme points and stripe bounding box.

```
class StripeSegment
{
        ...
        //! Points belonging to this stripe segment
        std::map<int, StripePoint::Ptr> points;
        //! StripeSegment extremes(min and max values)
        Extremes extremes;
        //! StripeSegment index
        int index;
        //! StripeSegment bounding box
        cv::Rect boundingBox;
};
```

StripeNode class

Encapsulates one stripe segment and adds components that allows to build n-ary tree of this nodes. Each node in the tree has both information about its children and parent nodes.

```
class StripeNode
{
        ...
        //! StripeSegment belonging to this node
        StripeSegment::Ptr stripe;
        ...
        //! Child nodes
        std::map<int, StripeNode::Ptr> children;
        //! Parent nodes
        std::map<int, StripeNode::Ptr> parents;
        ...
};
```

4.4 Calibrator

Calibrator namespace contains all the classes that are needed for calibration. Namely it is `CamCalibrator` for the camera calibration, `LaserCalibrator` for the laser calibration and also `SystemParameters` class that encapsulates all the system calibration data.

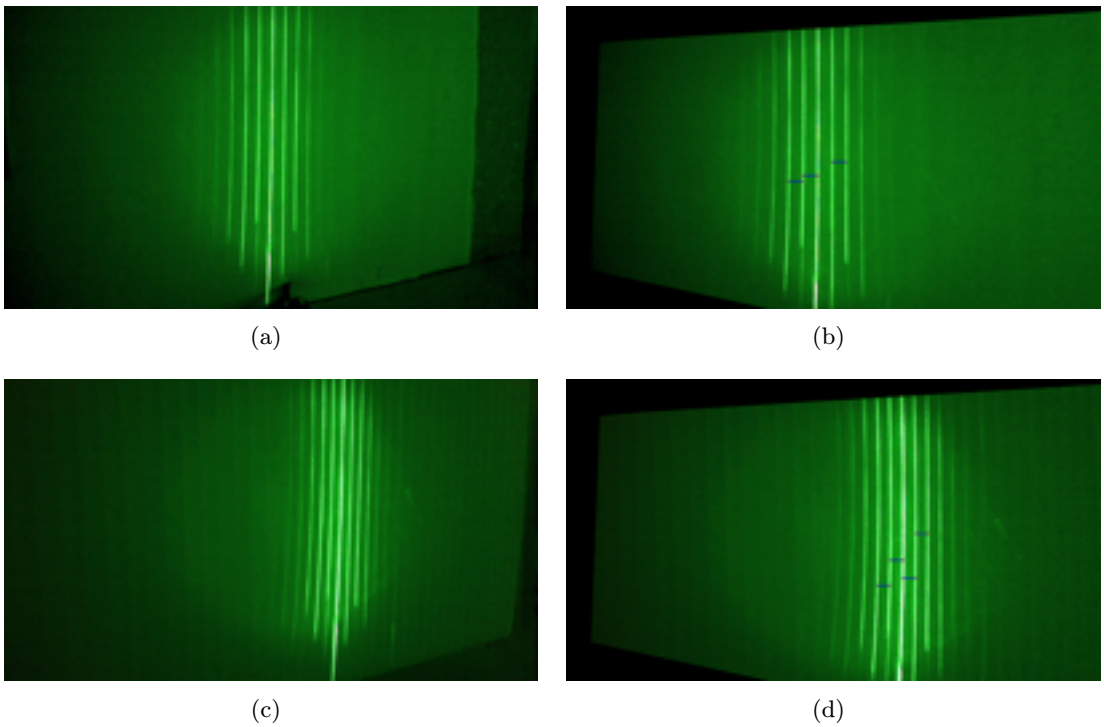
CamCalibrator class

As it was already mentioned, OpenCV camera calibration functions have been used for the camera calibration. Because of that, whole `CamCalibrator` was designed according to the Chapter 9 of [12] with some modifications to fit this project the best.

LaserCalibrator class

For the purpose of the laser calibration, OpenCV library was used again. But as there are no special functions to calibrate a laser-camera system, the laser calibration had to be carried out separately. As mentioned in Section 3.2, calibration is done by analyzing the projected laser stripes. To detect the stripes, class `LineFinder` described later in Section 4.5 was used. Detected lines are stored in a `std::vector` structure from the C++ STL library for further processing.

Example of the input laser calibration images and the result of their processing is shown in Figure 4.1, where the blue lines represents the computed distances from which the average distance is obtained.



(a) (b)

(c) (d)

Figure 4.1: Example of the laser calibration images. Input images can be seen on image (a) and (c). Result of processing is on images (b) and (d), where you can also see the computed distances represented by the blue lines.

SystemParameters class

SystemParams class does not provide any special functionality, it is more of a data storage class. It groups all the three calibration data structures(`CameraParameters`, `LaserParameters`, `GratingParameters`) together for easier manipulation later on.

4.5 Image Processors

Second namespace described here, called `ImageProcessors`, encapsulates all the image preprocessing functionality. It encapsulates four classes, namely `FloodFill`, `LineFinder`, `MaskExtractor` and `StripesExtractor`. Some of the classes are described below.

LineFinder class

Line finder class has its place in the laser calibration phase as mentioned in the previous section. It uses OpenCV function `cv::HoughLinesP` to detect straight lines in the image. Detected lines are then stored into std::vector C++ STL structure. It also provides function to draw the detected lines.

MaskExtractor class

As for every task in image processing phase, OpenCV was used again. Function `cv::subtract` was used to subtract the scene background. For the edge detection task, `cv::filter2D` function was used to implement the convolution operation. Because thresholding technique is used, the iterative threshold finding algorithm was implemented, which was mentioned earlier in Section 2.2. The thresholding itself is done using `cv::threshold` function or `cv::adaptiveThreshold` function in case of adaptive thresholding. Morphological functions that were used in the end are also implemented in OpenCV, concretely `cv::morphologyEx`.

StripesExtractor class

This class implements stripe indexing functionality, that is crucial to the whole image processing phase. Apart from the previous classes, no special OpenCV functions were needed. `FloodFill` class is used to fill a stripe region in the image. Example of the stripe indexing is shown on Figure 4.2.

4.6 Surface Constructor

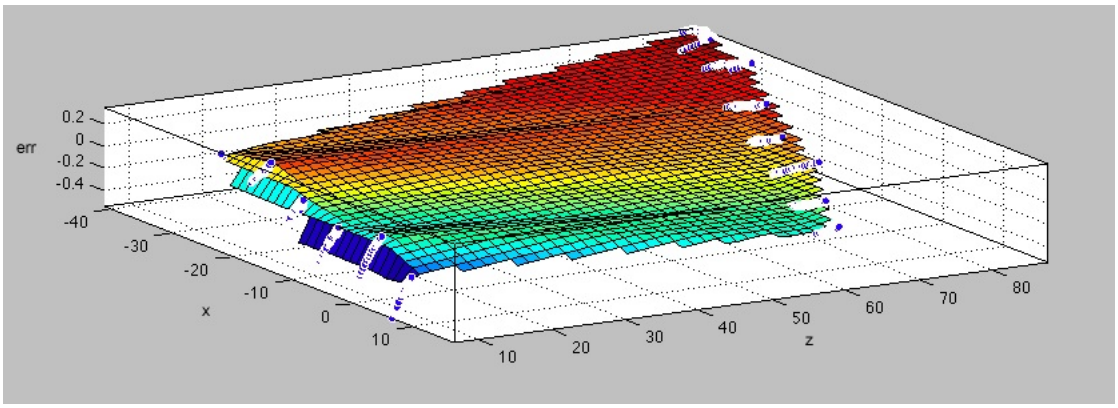
Last class described in this chapter does the main task in the end. As the stripes are already indexed and both background and hand models are prepared, it uses triangulation principle described in 2.4. It goes through the StripeNode tree and for each node, all the StripePoints that belongs to it are reconstructed. Final 3D surface model is saved to file using PLY file format.

It also implements the error correction functionality. Error correction function is obtained by reconstructing surface of a multiple rectangles with different heights from about $10mm$ to about $50mm$. Error at each point for each surface is computed and the results from all surfaces are put together. These merged error data are then approximated using

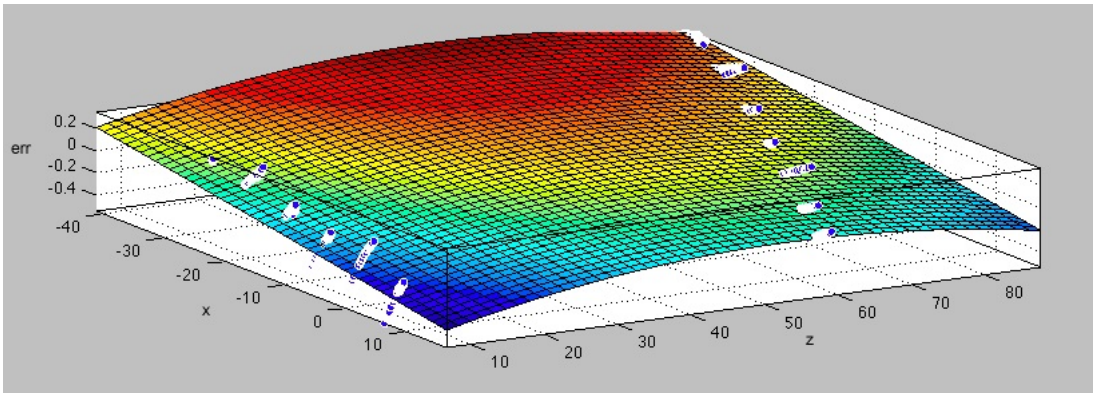
(a) (b)

Figure 4.2: Example of the stripe indexing process. Input image can be seen on image (a). Created node tree then on image (b).

Matlab Surface Fitting Tool. Obtained 2D function is in the end implemented in the reconstruction process and every point is corrected right after it is reconstructed. Example of the error function can be seen on the Figure 4.3(b).



(a)



(b)

Figure 4.3: Example of the reconstruction error function. In both images, x and z are x- and z-coordinates of a 3D point respectively and *err* is the error value. Reconstruction error distribution is shown on image (a). Approximation of this distribution is then on image (b).

Chapter 5

Experiments and Results

In this chapter, experiments that were done are presented and the results are evaluated. In order to verify the reconstruction process in general, first tests were done using simple geometrical objects of known height, etc. boxes, rectangles. After these basic tests, some experiments using actual human hands were performed. In the end, database of 20 users was obtained in order to have some testing data for future work.

Unfortunately I have to precede that the 4th laser(the one furthest away from the camera) is not used in the end due to the fact that the 532nm filter attached to the camera captures intensively only part of the image. The intensity decreases from the center to the edges of the image which causes the 4th laser to not being captured that intensively and it almost cannot be seen on the image.

5.1 Simple objects

As it was already mentioned, simple 3D objects were used first. At first, problem with the object masking was faced. Due to the fact that the device is really focused on human hand surface reconstruction, also the masking algorithm is made for that task and does not count with really bright objects on the bright background, hence for example object of a white color cannot be used just as it is, but the side considered as the upper one has to be covered with some let's say brown material first. For this purpose, simple color paper or a piece of cardboard is sufficient.

Rectangular boxes

As the first sample, a few simple rectangular boxes were chosen. Each of the boxes had different height(from 13mm to 40mm) so that the results can be evaluated for the objects of different proportions. For all the objects, at least 3 captures were done in order to compare the results of multiple reconstructions of the same object. The smaller the height of the surface was the bigger the reconstructed surface error was. Two examples were chosen to be published here, specifically the ones with heights of 13.5mm and 25.5mm. For both of them, root mean square error is computed having the real model and the reconstructed one. This value($RMSE$) is then obtained from the set of reconstruction data as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(zVal_i - h)^2}{n}} \tag{5.1}$$

, in order to specify the precision of the reconstruction method for the particular testing object with n surface points, knowing that the real rectangular surface had height of h and having the z-coordinate $zVal$ for each point on the model.

To verify the stability of the reconstruction, more reconstructions of the same object were done and the resulting models were compared. Difference between the two models $diff$ was computed for each pair of models of the same object considering their $RMSE$ values

$$diff = |RMSE2 - RMSE1| \tag{5.2}$$

, where $RMSE2$ stands for the $RMSE$ of the first model and $RMSE1$ stands for the $RMSE$ of the second model.

For both objects, $RMSE$ is measured first for each laser separately and then for the whole model created by merging the results from each particular laser. Also $NRMSE$ value is measured, which is a normalized $RMSE$ and is often expressed as a percentage, where lesser values means less residual variance. The following Table 5.1 shows all the results. Table 5.1 shows that the highest precison reconstruction does the laser with index 2. The

Table 5.1: Surface reconstruction $RMSE$(Root Mean Squared Error) values

	Box 13.5mm		Box 25.5mm	
Model	NRMSE	RMSE	NRMSE	RMSE
Laser 0	0.1671	1.2816	0.2190	1.4558
Laser 1	0.1144	0.6912	0.1993	0.7977
Laser 2	0.1525	0.3743	0.1764	0.4317
Merged	0.1155	0.9114	0.1576	1.0474

differences between the $RMSE$ of reconstructions from different lasers are due to the fact that the geometry of the system is not ideally precise. In addition, the image preprocessing for the images related to the laser with index 0 always produces worse results than for the other lasers, which is also reflected in the results from the table above this article.

As the $RMSE$ values were known, they were compared in order to verify the stability of the method using equation 5.2. Three models of the rectangular box with height of 25.5mm were reconstructed and then all the pairs were compared. The Table 5.2 presents the results of this comparison. It is obvious from table 5.2 that the reconstruction method

Table 5.2: Surface reconstruction $RMSE$(Root Mean Squared Error) values

Models	Laser 0	Laser 1	Laser 2	Merged
1 and 2	0.0173	0.003	0.0089	0.0123
1 and 3	0.0266	0.0261	0.0146	0.0141
2 and 3	0.0093	0.0291	0.0057	0.0018

produces quite stable results for multiple reconstructions of the same model, since the value of difference of $RMSE$ between 2 models of the same object is in orders of 10^{-2} - 10^{-3}. Examples of the reconstructed surfaces are shown on the Figure 5.1.

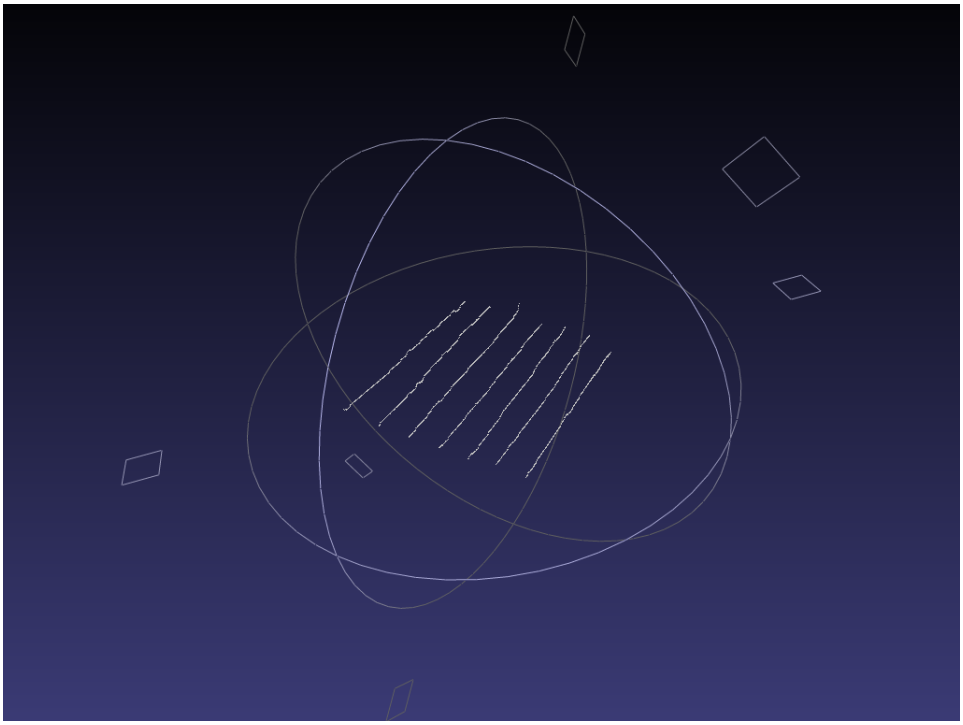
5.2 Human hands

Some testing reconstructing the actual human hands was done also. Especially for the purpose of stripe indexing. As it can be seen on the Figure 5.2, errors can occur in the case of the shorter fingers(e.g. thumbs, small fingers). Where the stripe with the index 1 lies is not important since the indices are always aligned to the central stripe index and each index during actual surface reconstruction is always considered respectively to that central one. Another example that I want to present is the result of the surface reconstruction. You can see the point clouds on Figure 5.3. Result is presented for each of the lasers separately, to get the final hand model, all the separate models has to be merged together into the final one. As it was already mentioned at the begining of this chapter, 4th laser is not considered for now, so not the whole hand surface can be reconstructed, because three lasers cannot cover the whole hand surface.

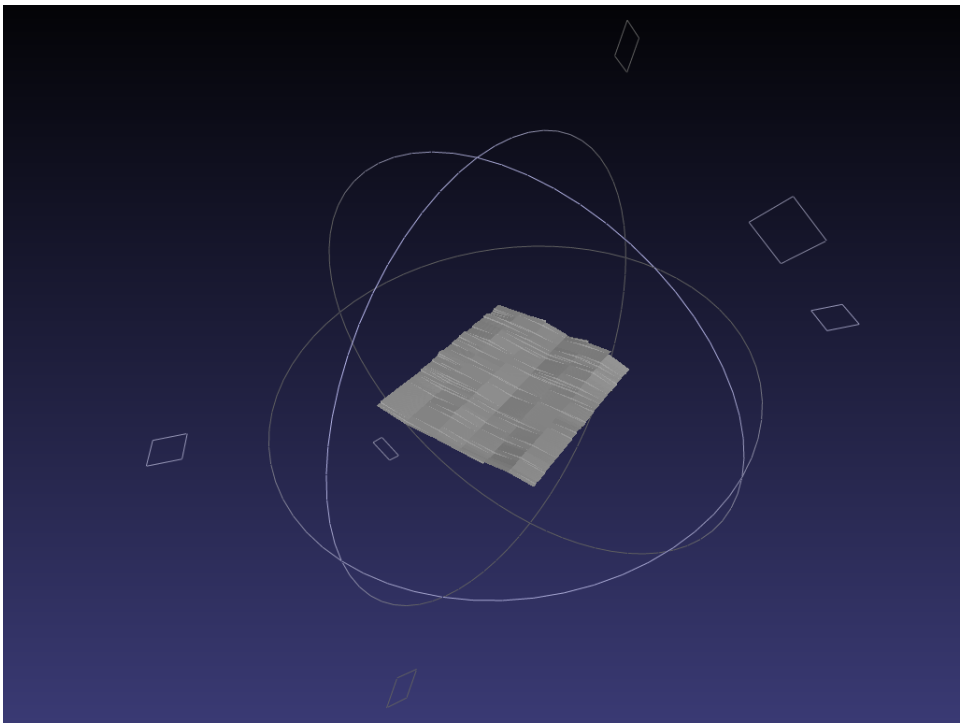
5.3 DB creation

Last task was to create a database of 20 human hand pairs(of both right and left hands) for the purpose of further testing. It was important to obtain both men and women samples and ideally also cover wider range of ages.

The database consists of 18 men and 2 women. Age of the men that provided samples ranges from 19 to 40 years, most of them is about 25. Regarding women, persons of age in range 20 and 30 provided their samples. All the samples can be found on the DVD attached to this thesis.

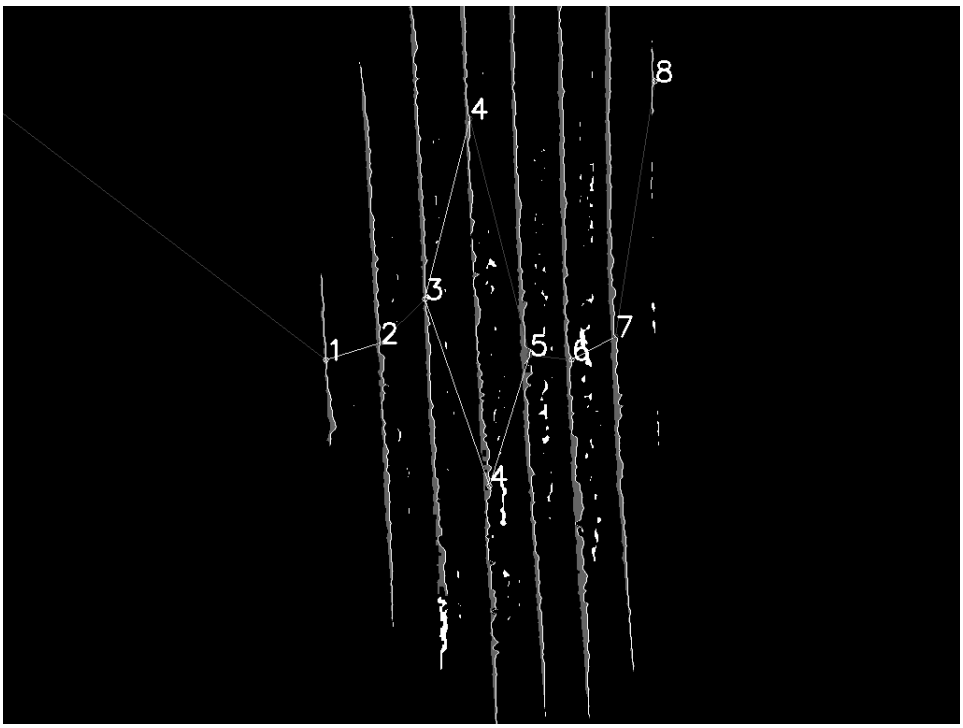


(a)

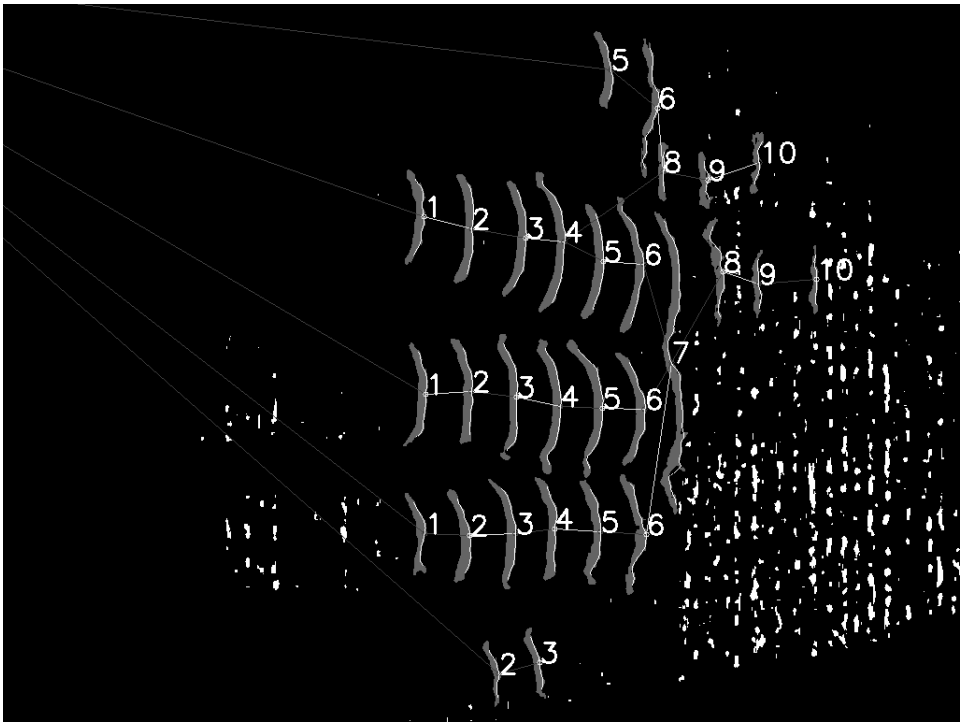


(b)

Figure 5.1: Point cloud of the rectangular surface reconstructed from the stripes of laser with index 1 can be seen on image (a). Image (b) contains the same surface but the point cloud is already meshed.

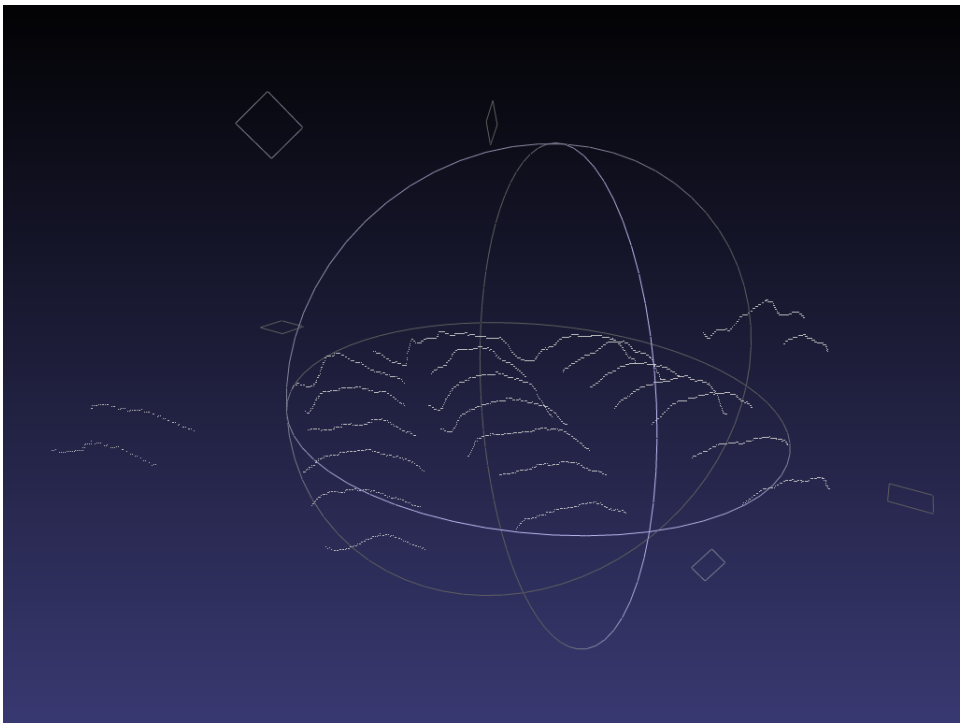


(a)

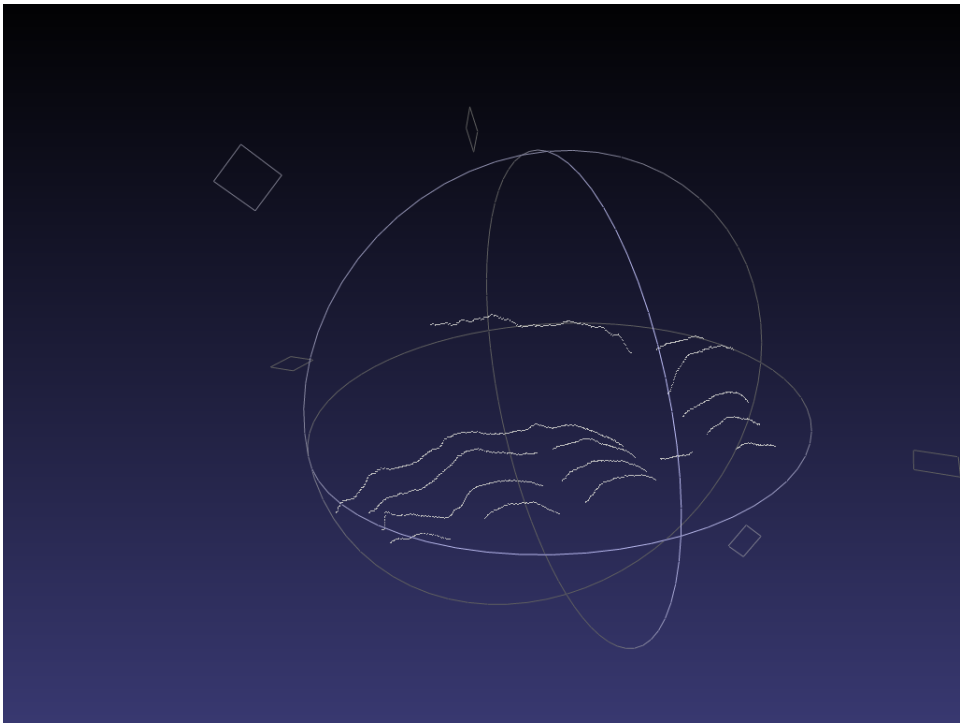


(b)

Figure 5.2: Indexed stripes of the background model are shown on image (a). Image (b) presents the indexed stripes of the hand model. You can see some bad indexing due to the noise in the filtered image on image (b) from index 7 on.



(a)



(b)

Figure 5.3: Point cloud of the hand surface reconstructed from the stripes of laser with index 1 can be seen on image (a). Image (b) shows point cloud of the hand surface reconstructed from the stripes of laser with index 2. Notice the wrongly positioned stripes on the image (b) due to the bad stripe indexing process.

Chapter 6

Peroration

Biometric recognition of people according to the human hand surface is quite a youngh task that has a big potential due to the fact that it can be applied in the cases where for example fingerprint devices cannot be used. Many methods for the 3D reconstruction of an object surface are already developed and can do the task of hand 3D model creation. But many of these methods require use of devices that are inappropriate in this case. Some of them do not have a good properties, others are too expensive. Thus the structured lighting principle has been used and a method that uses cheap, sufficient devices and provide enough precision has been proposed. Some 3D surfaces were successfully reconstructed, but the method itself still needs improvements, especially the image preprocessing and stripe indexing process, that influence the precision of the result a lot.

The future work, hand in hand with the algorithms improvements, also includes revision of the camera optical filter, which limits the use of 4th laser now. This has to be improved, so that all four lasers can be used in the end to create the whole model. As the final model was mentioned, another thing that has to be done comes in mind. The model merging process has to be improved and some robust method to be proposed. Also, an approach to mesh the resulting point cloud will be chosen.

In order to achieve higher precision, use of a second camera will be tried, which can, in combination with the structured lighting approach, provide much more information about the observed scene. I will also try to remove the backplane, since it is not needed after the calibration is done, but on the other side, it will require to control the hand input some other way.

Bibliography

[1] Nokia Corporation. Qt - cross-platform application and ui framework. http://qt.nokia.com/.

[2] Martin Drahansky and Filip Orsag. *Biometrie*. Computer Press a.s., 2011. ISBN 978-80-254-8979-6.

[3] Kjell J. Gasvik. *Optical Metrology Third Edition*. John Wiley & Sons, Ltd., 2002. ISBN 0-470-84300-4.

[4] Applied Informatics Software Engineering GmbH. Poco c++ libraries. http://pocoproject.org/.

[5] A. P. Godse. *Computer Graphics*. Technical Publications, 2008. ISBN 9788189411589.

[6] Khronos Group. Open graphics library. http://www.opengl.org/.

[7] h2physics.org. H2physics.org. http://h2physics.org/?cat=49.

[8] David Halliday, Robert Resnick, and Jearl Walker. *Fundamentals of Physics, Fifth Editon Extended*. John Wiley & Sons, Inc., 1997. ISBN 0471105597.

[9] Ph.D. Jean-Yves Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj/.

[10] Prof. Dr. Bernd Jähne. *Digital Image Processing 5th revised and extended edition*. Springer, 2002. ISBN 3-540-67754-2.

[11] Reinhard Klette, Karsten Schluns, and Andreas Koschan. *Computer Vision: Three-Dimensional Data from Images*. Springer, 1998. ISBN 9813083719.

[12] Robert Laganičre. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, Ltd., 2011. ISBN 978-1-849513-24-1.

[13] opencv dev team. Open source computer vision library. http://docs.opencv.org/.

[14] Riha Zdenek a kolektiv Rak Roman, Matyas Vaclav. *Biometrie a identita cloveka*. Grada Publishing a.s., 2008. ISBN 9788024723655.

[15] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall, Inc., 1998. ISBN 0-13-261108-2.

[16] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Appendix A

Contents of CD

- `src/` - program source files;
- `bin/` - program binary for Linux and required libraries;
- `doc/` - program documenation;
- `thesis/` - electronic version of the thesis;
- `db/` - database of 20 persons;
- `readme.txt` - important information about how to use the application.