Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

Bc. Zuzana Bobotová

PROCESSING OF THREE-DIMENSIONAL MEDICAL DATA USING COMPUTER VISION METHODS

Diploma Thesis

Degree Course: Information Systems

Field of Study: 9.2.6 Information Systems

Place of the development: Institute of Applied Informatics, FIIT STU Bratislava

Supervisor: doc. Ing. Vanda Benešová, PhD.

May 2018

Declaration of Honor

I honestly declared that this thesis was written independently by me under professional supervision of doc. Ing. Vanda Benešová, Phd. All references have been clearly cited.

……..
Zuzana Bobotová

Acknowledgement

The biggest thanks belongs to my supervisor doc. Ing. Vanda Benešová, PhD. Thanks her I had an opportunity to work on the interesting project and during that I learned many new important things. I am thankful for her professional support, human touch, patience and helpful advices during the work on this thesis and also for her time.

I would like to thank whole Vision and Graphic Group. The members inspired me a lot and they gave me many questions to think about.

I would also like to expect many thanks to my parents, boyfriend, family and friends. Their support helped me a lot during my studies and also during my work on this thesis.

Zuzana Bobotová

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Informačné systémy

Autor: Bc. Zuzana Bobotová
Diplomová práca: Spracovanie trojrozmerných medicínskych dát metódami počítačového videnia
Vedúci práce: doc. Ing. Vanda Benešová, PhD.
máj 2018

Práca je zameraná na metódy vhodné pre sledovanie vývoja nádorového ochorenia mozgu. V práci využívame dáta z magnetickej rezonancie. Ide o lekársky test, ktorý sa najčastejšie používa pri diagnostike rakoviny mozgu a tiež počas liečby - hlavne pre plánovanie ožarovania a sledovanie zmien nádoru. Aby bolo možné sledovať zmeny tumoru v čase, je potrebné segmentovať nádor z trojrozmerných dát a tiež zaregistrovať dáta z rôznych vyšetrení vykonávaných v pravidelných časových intervaloch. Analyzovali sme niektoré existujúce metódy pre segmentáciu mozgových nádorov a jednu z nich sme v našom výskume aj testovali. Najlepšie metódy pre segmentáciu nádorov sú založené na konvolučných neurónových sieťach. Následne sme analyzovali stav problematiky registrácie a navrhli metódu pre rigidnú a taktiež nerigidnú registráciu. Rigidná registrácia umožňuje zarovnať dáta z rôznych vyšetrení. Proces sa skladá z nájdenia kľúčových bodov, nájdenie zhôd medzi kľúčovými bodmi, filtrovania zhôd a rigidnej transformácie. Na druhej strane, nerigidná registrácia umožňuje sledovanie zmien nádoru a mozgu medzi jednotlivými vyšetreniami vykonanými počas procesu liečenia. Navrhli sme jednoduchú metódu založenú na optickom toku. Metóda nájde korešpondujúce body medzi dvomi rôznymi vyšetreniami. Následne sú tieto body použité na vizualizáciu zmien pomocou algoritmu morfovania obrazu. Posledná časť práce pozostáva z jednoduchého vizualizačného nástroja, ktorý sprostredkúva rôzne pohľady na dáta.

Annotation

Slovak University of Technology in Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Information Systems

Author: Bc. Zuzana Bobotová
Diploma thesis: Processing of three-dimensional medical data using computer vision methods
Supervisor: doc. Ing. Vanda Benešová, PhD.
2018, May

The work is focused on the methods suitable for monitoring development of the brain tumor disease. At the approach, we work with magnetic resonance data. It is a medical test that is the most commonly used for diagnosing the brain cancer and also during the treatment - mainly for radiation planning and tumor changes observation. In order to track tumor changes over time, it is necessary to segment tumors from the volumes and also register data from different examinations performed in regular time periods. We have analyzed some existing methods for tumor brain segmentation and tested one of them in our research. The best methods for tumor segmentation are based on the convolutional neural networks. Subsequently we have analyzed state of the registration problematic and proposed a method for rigid and also non-rigid registration. Rigid registration can align data from different examinations. The process consists of: finding key points, finding matches between key points, filtering matches and rigid transformation. On the other side, non-rigid registration allows track tumor and brain changes between the examinations captured during the treatment process. We proposed a simple method based on the optical flow. The approach finds corresponding points in two different examinations. In order, the points are used to visualize the changes using image morphing algorithm. Last part of the work is a simple visualization tool that offers different views on the data.

Content

List of terms

Axial, coronal sagittal: different directions of the brain slices

Brain: organ, center of nervous system

Cranium: skull, bones protecting brain

CSF: protects the brain from shocks and supports the venous sinuses

Fusions: volumes captured in different time during the treatment

High grade glioma: tumors of the brain with well visible boundaries of the tumor

Low grade glioma: tumors of the brain with hardly visible boundaries of the tumor

Magnetic Resonance Imaging: medical technique in radiology to create anatomy images

Tumor: abnormal mass of tissue

List of shortcuts

BRATS: Multimodal Brain Tumor Image Segmentation Benchmark

CRF: conditional random fields

CSF: cerebrospinal fluid

DFT: Discrete Fourier Transformation

DCT: Discrete Cosine Transformation

FAST: Features from Accelerated Segment Test

FCNN: fully convolutional neural networks

GUI: graphical user nterface

HGG: High Grade Glioma

LGG: Low Grade Glioma

ITK: Insight Segmentation and Registration Toolkit

MR: Magnetic Resonance

MRI: Magnetic Resonance Imaging

OpenCV: Open Source Computer Vision Library

ORB: Oriented FAST and Rotated BRIEF

SIFT: Scale-Invariant Feature Transform

SLIC: Simple Linear Iterative Clustering

SURF: Speeded-Up Robust Features

SVM: Support Vector Machine

VTK: Vizualization ToolKit

List of Figures

List of Tables

1 Introduction

Medicine is an area which is still going ahead. It is important field for people and their health. Many different techniques are evolved to diagnose diseases. In our research, we are working on the methods appropriate for monitoring the development of the brain tumor cancer. The most commonly used test for tumor diagnostic is magnetic resonance imaging. The result of one magnetic resonance test of the brain is a volume - series of images called slices. Nowadays, the computer is being used more and more during the diagnosing the patients. It gives more opportunities to the doctors and can help them work more effectively.

However, the processing of the three-dimensional medical data (in our case MRI) can be time-consuming and annoying task. Computer vision is a powerful tool to solve many interesting problems and can help with automation of the monotone and routine tasks in many fields. Medicine is one of the fields, where computer vision is very helpful and may be very powerful.

Actually, there is a lot of tasks in medicine and diagnostics, where some algorithm can ease the work of the specialist. Special computer vision methods are developed to solve one specific task. Thus, we decided to focus on the brain and cancer in the brain – tumors. In our work we present methods which are capable to help with diagnostic of the cancer in the brain, mainly to help with the tracking the tumor changes.

To process MRI data without computer vision methods, for example segment the tumor from the volume, the specialist have to segment the tumor on each slice of the volume. It is time-consuming and monotone task, but is really necessary and very helpful – mainly during the treatment process by the irradiation when health parts of the brain should be radiated as less as possible.

The main task the work was find automatic methods for tracking changes of the patient's disease: track changes of the tumor and track changes of the health brain parts. To make it possible, it is necessary to segment tumor from the volume, align data from different examinations and find corresponding parts of the brain and tumor in different examinations captured during the treatment process in regular time periods. Today many approaches appropriate for tumor segmentation exist. Thus we decided to analyze them and test some. To align data from different examinations we proposed a method based on rigid registration. The method can be used to align corresponding 2D slices from different examinations. Finally, to track changes we proposed method based on non-rigid registration. Method can find the corresponding points between the examinations using optical flow. It works with the 2D corresponding slices.

The work consists of several chapters. The 2nd chapter contains general information about the computer vision and about computer vision libraries. Additional three chapters give a look at the image preprocessing (chapter number 3), segmentation (chapter number 4) and registration (chapter number 5) methods. 6th chapter is devoted to the neural networks which is a powerful tool for many computer vision problems. Medical point of view on the brain cancer is provided in the chapter number 7. Overview on the state of the art methods of the brain tumor segmentation and brain registration are summarized in the chapter number 8. Finally, proposed method, implementation details and results of the testing are summarized in chapter number 9. The method consists of several steps. For every step is denoted one subchapter: preprocessing (chapter 9.3), segmentation (chapter 9.4), rigid registration (chapter 9.5), non-rigid registration (chapter 9.6), visualization (chapter 9.7).

2 Computer vision

Computer vision is an area that can deal with understanding digital images, videos and other digital data. It is important to realize that the computer see data differently than we do. The image is only a field of values. To get some information, it is inevitable to develop methods and algorithms.

Digital data may be reached by different units – cameras, sensors, scanners and other specialized units. Usually, it is necessary to process these data in order to get some information. It can be monotone and boring task for the human if it is done manually. Computer vision methods and algorithms are very powerful and may facilitate human work or help the human with the processing and data analyzing.

Computer vision is highly used in various science disciplines, but also in everyday life. For example, when you are leaving from the paid car park with the barriers in front of the shopping center, you do not have to scan the ticket which you took on your arrival. It is because the camera captured your car when you arrived, computer vision methods detected the sign on the car and paired your ticket with the sign and also remembered the time of your arrival. When you left, the camera again captured the car, computer vision detected the sign and the system had known if you have already paid.

Nowadays, we can also use many features of the cars which use computer vision, such as parking assistant or autopilot. It is still necessary to pay attention when you are using these features, but in future there will be absolutely autonomous cars.

Computer vision methods are also used in the industry to control the quality, in many monitoring systems to control the situation, prevent some accidences or detect some events. Physics, biology, chemistry, universe research or robotics are fields where computer vision has a lot of applications.

2.1 Computer vision in medicine

Medicine is another field where computer vision can ease the human work and help in different ways. In medicine is a lot of tests where different scanners and sensors are used. Well known tests are Magnetic Resonance Imaging (MRI), Computed Tomography (CT), sonography, x-ray and so on. Computer vision methods can help doctors and diagnosticians to process data and ease their work. During the keynote presentation by Dr. Yoav Medan [77] on the MICCAI 2016 conference in Athens were presented all fields of the medicine where some research in computer vision is registered. There are listed all medicine fields mentioned in Dr. Medan's presentation:

- "Oncological: Bone Metastases, Prostate Cancer, Breast Cancer, Kidney Cancer, Liver Cancer, Pancreatic Cancer, Soft Tissue Tumors, Brain Tumors, Pediatric Neuroblastoma, Head and Neck Cancer, Lung Cancer, Ovarian Cancer, Bladder Cancer, Colon Cancer, Esophageal Cancer

- Musculoskeletal: Back Pain, Osteoid Osteoma, Osteoarthritis, Disc Degeneration, Muscle Atrophy, Sacroiliitis, Spinal Cord Injury, Spinal Tumors

- Neurological: Essential Tremor, Neuropathic Pain, Parkinson's Disease, Brain Tumors, Depression, OCD, Alzheimer's Disease, Epilepsy, Hydrocephalus, Multiple Sclerosis, Stroke, Traumatic Brain Injury, Trigeminal Neuralgia, AVM's Cancer Pain

- Women's health: Uterine Fibroids, Breast Fibroadenomas, Uterine Adenomyosis, Tubal Pregnancy, Fetal Surgery, Ovarian Cancer, Polycystic Ovarian Syndrome

- Cardiovascular: Hypertension, Atherosclerosis, Atrial Fibrillation, Deep Vein Thrombosis, Heart Block, HLHS, Peripheral Artery Disease, Septal Perforation

- Urological: Prostate Cancer, Kidney Cancer, Benign Prostatic Hyperplasia, Acute Kidney Injury, Acute Tubular Necrosis, Ureterocele, Bladder Cancer

- Endocrine Disorders: Thyroid Nodules, Diabetes, Obesity

- Miscellaneous: Hypersplenium"

2.2 Computer vision libraries

A lot of methods and algorithms have been invented until today. Some of them is not necessary to implement again, because there are some libraries which provide the implementation of these methods and algorithms.

2.2.1 OpenCV

OpenCV is an open source library which contains many implementations of computer vision methods. It is written in C++ and C programming language and it is cross-platform. OpenCV provides interfaces for other languages – Python, Java, Matlab and Ruby. OpenCV has very strong community, online documentation is available [84] and some books are devoted to the library, such as Learning OpenCV: Computer Vision in C++ with the OpenCV Library written by Gary Bradski and Adrian Kaehler [7]. Methods from OpenCV library are able to work only with two-dimensional data.

2.2.2 ITK

ITK (Insight Segmentation and Registration Toolkit) is another open source, cross-platform computer vision library. It is written in C++ and wrapped for Python and Java. Cmake must be used to build the environment. It is primary specialized on the segmentation and registration methods. Big advantage of the library is that algorithms are able to process two, three or more-dimensional data. ITK is often used for processing the medical data. It was developed in collaboration with National Library of Medicine. ITK have also strong support and own community, documentation is available, too [28]. ITK has also simpler version called Simple ITK.

2.2.3 VTK

VTK (Visualization Toolkit) is an open-source, cross-platform system which is able to process images, 3D computer graphic and provides methods for visualization of two, three and more-dimensional data. VTK is written in C++ programming language, but contains interfaces for Python and Java. VTK was created by Kitware company which also provides support for VTK. There is also online documentation available [73], but people complain that documentation is not very clear. One big advantage is that VTK has a suite for user interaction and another one advantage is that VTK can be integrated with GUI toolkit, for example Qt.

3 Image preprocessing methods

Image preprocessing is first step after the acquisition of image data. In medicine, we meet with data which are not perfect, so it is important to apply some enhancement techniques. Enhanced images can contain clearer and more perceptible edges of the objects as wrote Toennies in his book [59]. The main goal of preprocessing is to improve the quality of the data.

Images can be processed in spatial or frequency domain as is explained in the book [58] from Šikudová et al. The most usual enhancement techniques of image quality are:

- contrast enhancement,
- resolution enhancement,
- edge enhancement and
- noise reduction.

3.1 Processing in frequency domain

We are basically accustomed to the data in spatial domain- To process images in frequency domain, it is necessary to convert them. Discrete Fourier Transformation (DFT), Discrete Cosine Transformation (DCT) and wavelet transformation are used to convert data from spatial to frequency domain. Wavelet transformation is the most usually used for image filtration techniques. Result of transformation is a sequence of spectral coefficients.

Basic prejudice of DFT is that every signal can be represented as sum of sinus and cosine functions. There also exists an optimized algorithm of DFT called Fast Fourier Transformation (FTT) which is faster especially when number of coefficient is power of 2. Resulting coefficients are imaginary numbers. On the other side DCT convert signal into a sum only of the cosine functions and resulting coefficients are only real numbers.

3.1.1 Basic filters

Filtration is process when image spectrum (image transformed to frequency domain) is multiplied by spectrum filter. Then image is again transformed back to spatial domain. Basic filters used for filtration in frequency domain are low-pass filter and high-pass filter. As their names say, low-pass filter passes low frequencies and on the other hand high-pass filter passes high frequencies. Thus, if we want to remove details from the image, such as texture, we need to remove higher frequencies, so we pass only low frequencies – we use low-pass filter. If we want to get image only with edges, texture and details, we need to remove low frequencies, so we will use high-pass filter.

Filtration in frequency domain is usually used to remove periodic noise. We can use special high-pass or low-pass filters to remove local maxims in frequency domain – filters are centered on the centers of the maxims and multiplied with them.

3.2 Processing in spatial domain

3.2.1 Linear methods

Linear filter may be divided into smooth filters and edge detection filters [58]. They are based on convolution operation. Convolution is a method which systematically go through the whole image and compute new value of the processed pixel using small surrounding of that pixel. Processed pixel is called origin and surrounding of the pixel is defined by the convolutional kernel.

The best known smooth filters are Gaussian filter and mean filter also called blur filter. Kernel of the Mean filter is used for computing the arithmetical average value of the pixel's surrounding. Kernel of the Gaussian filter is derived from Gaussian function.

The best known edge detection filters are Prewitt, Sobel and Laplacian. Prewitt filter has got two different convolution kernels – vertical and horizontal. Thus, two information are computed using Prewitt filter:

- horizontal changes G_x of the image I are computed using horizontal Prewitt kernel:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} * I$$

- vertical changes G_y of the image I are computed using vertical Prewitt kernel:

$$G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * I$$

Sobel filter is very similar to Prewitt filter:

- horizontal changes G_x of the image I are computed using horizontal Sobel kernel:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * I$$

- vertical changes G_y of the image I are computed using vertical Sobel kernel:

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I$$

Prewitt filter and Sobel filer are usually used to compute image gradient. Image gradient G is a two-dimensional vector, which coordinates are partial derivations of the function $f(x, y)$:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

We can also compute the angle of the vector according to the formula:

$$\theta = \arctan(G_x, G_y)$$

Laplacian filter is defined as sum of second difference in origin pixel according to surrounding of the pixels defined by the kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

3.2.2 Nonlinear methods

Typical nonlinear filters are ordering filters such as minimum, maximum and median filter [58]. Non-linear filtration does not use convolution function. During the nonlinear filtration is used a mask which specifies the pixels which are processed. Image pixels under the value 1 of the mask pixel are included to the processing and on the other hand image pixel under the 0 mask pixel value are not included to the processing. The mask cross through the image and all values of the image which corresponding with the mask are ordered. Then first, last or middle value is selected depending on the type of the filter (minimum, maximum or median).

3.3 Morphological operations

Morphology studies objects, their shape, topology and geometry [58]. Morphological operations use structuring element which go through the image. Structuring element is defined by the size, shape and origin – position of the processed pixel. Morphology can process binary or greyscale images.

Dilation and erosion are basic morphological operations. Basically, binary dilation enlarges white regions and on the other hand binary erosion shrinks white regions of the binary image. In general binary dilation can be defined as union of the image and structuring element. Then binary erosion is defined as intersection of the input image and structuring element. In the Figure 1 are shown results of binary dilation and erosion. Greyscale dilation is defined as a maximal value of the origin's surrounding - intended by structuring element and greyscale erosion is minimal value of the origin's surrounding.

Figure 1: LEFT – input image, IN THE MIDDLE – dilated image, RIGHT – eroded image [58]

Opening and closing are operation derived from erosion and dilation. Opening is operation where erosion is followed by a dilation. If I is an input image and S is a structuring element then opening can be defined as:

$$Opening = (I \ominus S) \oplus S$$

On the other hand, closing is operation where dilation is followed by the erosion:

$$Closing = (I \oplus S) \ominus S$$

Another derives of the erosion and dilation are top-hat and bottom-hat operations. Top hat operation is a subtraction of input image and the morphological opening:

$$Top\ hat = I - ((I \ominus S) \oplus S)$$

Bottom hat is a subtraction of morphological closing and input image:

$$Bottom\ hat = \big((I \oplus S) \ominus S\big) - I$$

Morphological gradient is next operation which uses erosion and dilation. Erosion and dilation are not inverse operation and this fact is used in morphological gradient. It is a subtraction of the dilation and erosion computed with the same structuring elements. It is defined with formula:

$$Morphological\ gradient = (I \oplus S) - (I \ominus S)$$

The method can be used to detect the boundaries of the objects on the image.

4 Segmentation methods

Segmentation methods divides image on several parts depending on some characteristic, such as shape, color, intensity or texture. Segmentation is usually used to detect some object on the image. Today a lot of segmentation methods are invented.

Many approaches such as book about medical image analysis from Toennies[59] divide segmentation methods on:

- manual,
- semi-automatic or
- automatic.

Manual segmentation does not require any computer vision methods. Objects are segmented by the expert manually using some special software and tools. It is time consuming and monotone task, so computer vision experts desire automatic or semi-automatic methods appropriate for different problems. Semi-automatic methods require some interaction with the user, but they are faster than manual segmentation. Automatic methods do not require any inputs from the user, everything is computed by to segmentation algorithm.

In medicine, segmentation algorithms are used to detect some specific organs and parts of the body or to detect some anomalies on the body, such as tumors. There are several processes how to reach data about the body structure. Every type of data acquisition is useful for different medical problem and requires specific processing.

Segmentation methods can be divided into several groups based on approach to segmentation. Surveys, such as survey from Masood et al. [40], divide them on

- edge-based methods,
- region-based methods,
- machine learning and
- atlas guided.

4.1 Edge-based methods

One of the most popular features are edges. They can be used to guide the segmentation and we can extract different types of edges: parametric, intensity, texture and watershed edges as is mentioned in approach from Elankib et al. [16]

4.1.1 Graph-cut algorithm

Graph-cut algorithm is based on special representation of the image. Pixels represents nodes and between neighboring pixels are edges as is shown in the Figure 2. Every edge has got some value

called weight. In the graph based computer vision algorithms weights depends on the similarities and dissimilarities between the neighboring pixels. Weight may be defined by differences in the intensity, color, texture or another characteristic. Type of the characteristic depends on the type of the algorithm.

Graph is cut to split image on several segments. Max-flow and min-cut algorithm are two basic graph-cut algorithms [55]. Min-cut algorithm finds the smallest possible cut of the graph edges and divide the image on foreground and background. Max-flow algorithm finds the highest possible flow of the path in the graph. The path is created in iterations in effort to maximize the flow. Path consists of the edges. The flow is defined by the maximal value of the edges included in path.

There are several approaches which use graph cut algorithm for the segmentation such as segmentation of the lung from chest radiographs by Candemir [9], segmentation in N-dimensional medical images by Boykov and Jolly [6] and so on.

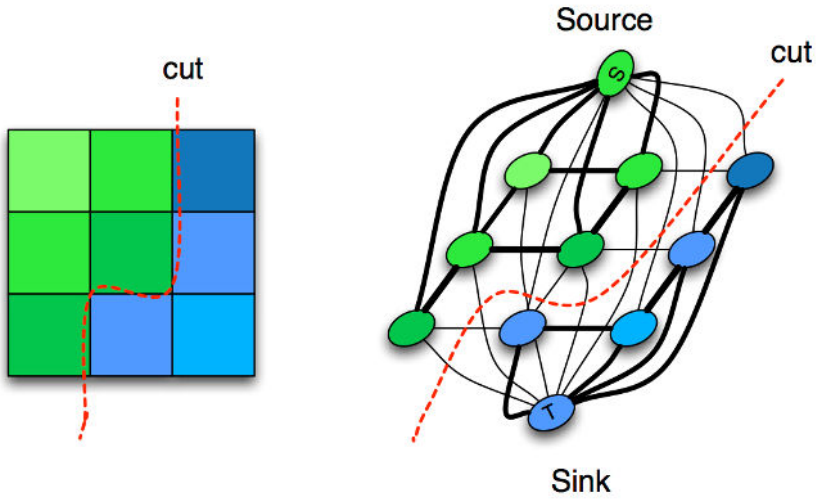


Figure 2: Visualization of graph cut algorithm [75]

4.1.2 Active contour algorithm

Active contour algorithm is also known as snake algorithm. It is based on movement of the curve – snake along the image structures as is shown in the Figure 3. Ivins[27] writes that algorithm try to minimize the energy function of the snake in several iterations. Energy of the snake consists of internal and external energy. Function of the external energy is proposed to find the most accurate edges of the object – the biggest local changes. On the other hand, function of the internal energy should keep the snake smooth.

In basic implementation of the algorithm, programmer must specify the weights of the energies which have an impact in minimization function and whole result of the segmentation. Next input for the method is initial snake which should approximately plot the boundaries of the object.

Algorithm was used in some approaches such as segmentation of the ventricle by Cohen [10] or for the segmentation of brain structures by Yushkevich et al. [65]

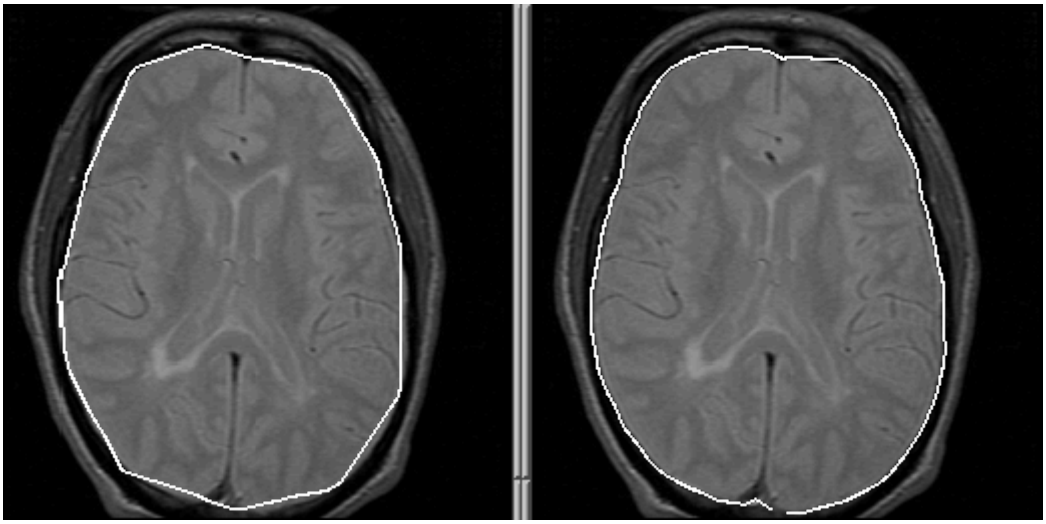


Figure 3: Active contour algorithm in practice [70]

4.2 Region-based methods

4.2.1 Thresholding

Thresholding is basic and very often used computer vision method. Input for the method must be grayscale image and output of the method is binary image (pixels can have values 0 or 1) as is shown in the Figure 4. Values of the final binary image are computed from the input image depending on the threshold value. Threshold is a constant entered by the programmer and defines the boundary between the higher and lower intensity values of the pixels from the input image. Resulting value of the pixel in output binary image is defined by the formula ($I(P_{i,j})$ is an intensity of the input pixel and T is a threshold constant):

$$P'_{i,j} = \begin{cases} 1 & if\ I(P_{i,j}) \geq T \\ 0 & otherwice \end{cases}$$

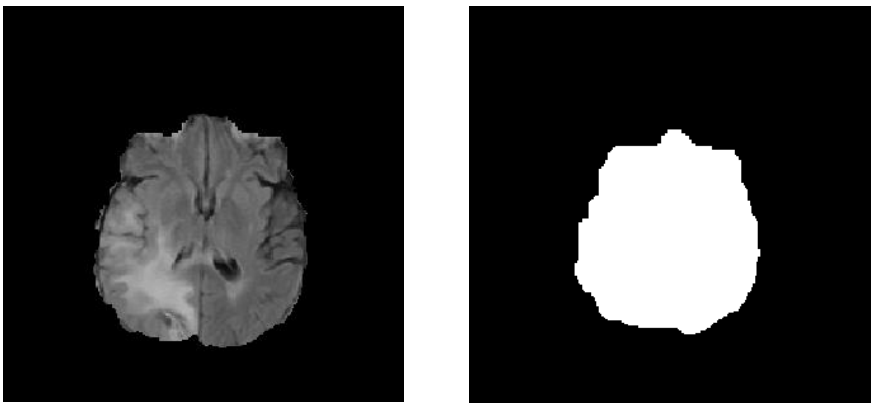


Figure 4: LEFT – input image, RIGHT – thresholded image

4.2.2 Region growing method

Region growing method is iterative algorithm and the region – segmentation grows in every iteration if the structure of the image allows it. It means that region-growing algorithm is a merging-type region-based algorithm as is mentioned in the book from Glasbey [19]. In the Figure 5 is shown several iterations of region growing algorithm.

The algorithm needs an initial position of the region called seed. Seed may be specified by the user using simple interaction or by computed using some statistical methods. In every iteration the algorithm controls all surrounding pixels of the region and decides if the pixel will be added or not. It depends on some characteristic of the image. If difference between region pixel and surrounding pixel is too big then pixel is not added to region, otherwise it is.

Region growing segmentation was used for example for the heart segmentation from ultrasound images by Hao et al. [23]

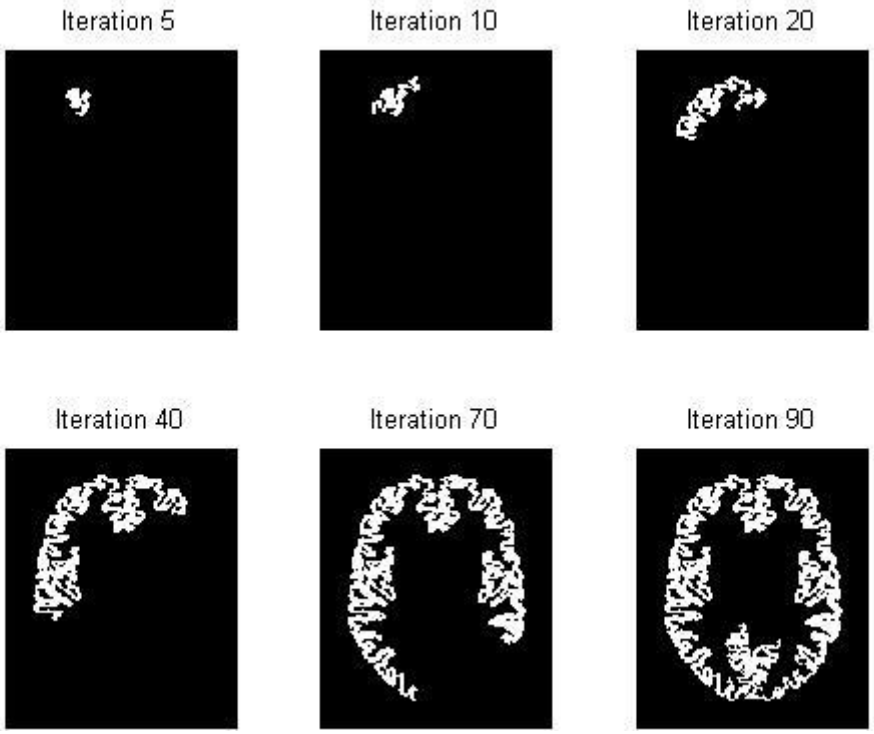


Figure 5: Iterations of region growing algorithm [80]

4.2.3 Watershed algorithm

Watershed algorithm is based on the principle of height map flooding. The relief is represented by the image gradient. Thus, as the relief of the height map is flooded, the segmentation of the watershed algorithm increases.

Basic algorithm starts in local minims of the image [5] (it can also start on local maxims and decrease the values in iteration). The pixel values that can be added to the regions are gradually increasing, so the regions are enlarged in iterations. When two different regions meet, the algorithm creates a border

between them and that's are the final boundaries of the image segments. The process of the algorithm is shown in the Figure 6.

Watershed segmentation algorithm with some improvements was used for example to segment the breast tumors by Huang [26] and Chen or Grau et al. tested watershed method for knee cartilage segmentation and gray/white matter segmentation [22].

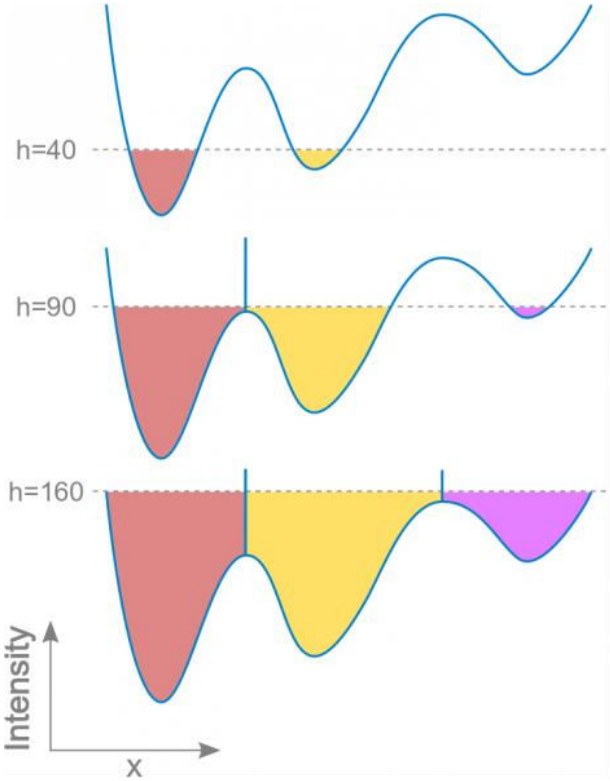


Figure 6: Iterations of the watershed algorithm [56]

4.3 Machine learning

Simply, machine learning makes computers capable to learn without explicitly programming. Computer can learn some models and make segmentations depending on the models and similar conditions on the input image or can divide image on the parts – segments depending on some characteristic. Basic division of machine learning techniques is on:

- unsupervised learning (clustering) and
- supervised learning (classification) as is mentioned in the book from Toennies [59].

Clustering is unsupervised learning method which divides pixels of the image on several clusters. No previous training of the model is needed. Pixels are divided depending on some specific characteristic – some feature. Pixels grouped in one cluster are more similar to each other than to the pixels from another groups.

Classification is supervised learning which means that we need to train some model, so we need some training dataset (data with real true results - segmentations). Training data are used to train the model

depending on some features and models are then used to classify pixels of the new input image into defined classes.

4.3.1 K-means

K-means is clustering efficient algorithm which can divide the image on K clusters. It is based on the criterion that the pixel is in the minimum distance from the center (mean) of all clusters according to the value of the pixel. Center is iteratively calculated as the mean of the all pixels in cluster [40].

K-means algorithm was used for example for brain tumor detection by Wu et al. [62] or for segmentation of brain parts by Foong et al. [44]

4.3.2 Fuzzy c-means

Algorithm is based on the theory about the fuzzy logic [40]. Algorithm is similar to K-means, but fuzzy c-means allows pixels to be a part of more than one classes.

Fuzzy c-means in combination with SVM algorithm was used for prognosis of diabetes in approach by Sanakal and Jayakumari [53].

4.3.3 Mean-shift

Mean-shift algorithm is clustering algorithm and consists of several steps. At first an initial search window is created. Then the algorithm computes centroid of the data (mean location) inside the search window and search window is centered at the mean location. These steps are repeated while the values converge.

4.3.4 Superpixel algorithms

At the beginning, superpixel is a set of pixels with some similar characteristic such as color, texture, intensity or some combination. Superpixels are used for the next processing, because their size is still small and they do not limit any whole object usually. However, the boundaries of the superpixels are corresponding with the boundaries of the object on the image.

SLIC (Simple Linear Iterative Clustering) is one of the best methods proposed to create superpixels from the image. It was invented by Achanta et. al [1] and the algorithm is used in many researches. At first, n initial cluster centers are created on a regular grid. The centers are moved in iterations to seed locations which corresponding to the lowest gradient. Thanks this superpixel boundaries corresponding with edges on the image. Example of superpixel clustering is shown in the Figure 7.

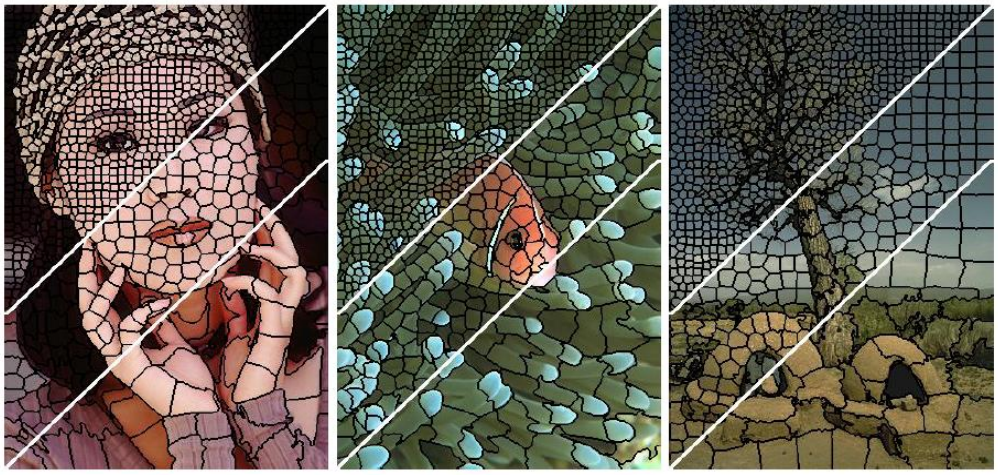


Figure 7: SLIC superpixel clustering

4.3.5 Support vector machine

Support vector machine (SVM) is supervised learning algorithm [31]. Thus it is necessary to have some training data to train the model which can divide another inputs to the classes after the training. During the training all training data are plot into n-dimensional space [59]. Number of spaces depends on the number of features which we can extract from the data. Then the algorithm finds the hyper-plane – the best bound between two classes in the several iterations as is shown in the Figure 8.

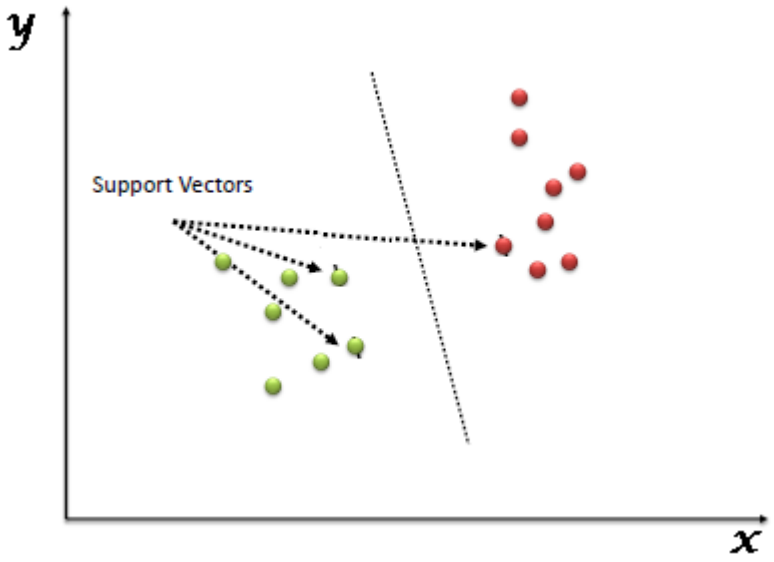


Figure 8: Graphical representation of support vector machine [82]

SVM classification algorithm was used in many approaches, for example to segment brain tumor from MRI images by Bauer et al. [3] or to recognize reliable heart beating from electrocardiogram waveform (ECG) [46].

4.3.6 Bayesian models

The simplest Bayesian model is naive Bayesian classifier – it is probabilistic classifier [30] good for large data. It is based on Bayesian theorem which calculate posterior probability [59]:

$$P\ (c|x) = \frac{P\ (x|c)\ P(c)}{P(x)}, \text{ where}$$

- $P\ (c|x)$ is the posterior probability of the class,
- $P\ (x|c)$ is the probability of the given predictor class,
- $P(c)$ is prior probability of the class and
- $P(x)$ is prior probability of the predictor.

However, a lot of improvements of the basic model exist and Bayesian models can be also used for clustering problem.

4.3.7 Decision trees

Decision tree is a classification algorithm which creates tree with decision nodes and leafs [30]. Decision tree is built using some features from the input data. Features and their possible values represent decision nodes and possible resulting output is represented by the list. Order of the features depend on the entropy. Entropy calculates the homogeneity of the data in the classes and if the class is homogenous then entropy is zero. It means that feature which can divide the data better is higher in the tree. After the training, the class of new input data is gained from the trained tree using the decision nodes and features.

Algorithm can be improved to decision forest which consists of many decision trees and result depend on more usual result of every decision tree. It is usually used for more complicated segmentation problems such as tumor segmentation in MRI images as in approach by Zikic et al. [68]

4.3.8 Neural networks

In the past years, neural networks were used to solve many different computer vision problems. Researches show that neural networks are very suitable for analyzing and processing the digital data. Neural networks are based on the prejudice that they will learn some rules on the big training dataset. Bigger training dataset can raise the accuracy of the network. Nowadays exist many different types of algorithm based on neural networks. Chapter number 6 Neural networks is devoted to the explanation of these methods.

4.4 Atlas guided segmentation

Atlas guided segmentation is special type of segmentation, because first step is to register data on the atlas [40]. Atlas is a general structure of some part of the body, or organ in the body. Thus input data have to be registered, so they are deformed. Input data should have similar structure as the atlas after the registration. When data are registered then atlas and information from the atlas can be used for the segmentation. Atlases are also usually used to detect some anomalies in the data – images or volumes.

5 Registration methods

A process which can align two or more images (the reference and sensed images) of the same scene is called registration. Crum, Hartkens and Hill in their article [14] write that for every one pixel from first input data is being found position of the corresponding pixel from another input data. Images may by captured by different sensors, from different points of view or they can be scanned in different times or with different conditions. Registration is a necessary step to analyze and gain some information from combination of various data sources.

From the point of view that what kind of images are registered, registration can be divided into four groups as written Zitová in her article [69]:

- registration of different viewports – the goal is to get bigger 2D view or 3D representation of the scene
- registration of data acquired in different times always also with different conditions – the goal is to detect changes between the acquisitions
- registration of data acquired by different sensors also called multimodal analysis – the goal is to work with more information obtained from different sensors
- registration of the model and the scene where model is some computer representation of the scenes such as atlases– the goal is find the localization of the scene in the model and compare them

Usually process of the data registration consists of four basic steps [69]: feature detection, feature matching, estimation of the transformation model and transformation

5.1 Feature detection

In feature detection step some salient objects are detected on the image depending on the method as shown in the Figure 9. They are usually represented as the array of the points called control points (or key points). There is a lot of possibilities how to detect the features, but they should be detected easily. Thus, we must decide what type of the features will be appropriate for our task. In the perfect world, the algorithm would be able to detect the same features on the all input images regardless of the deformation.

Feature detection methods can be divided from different points of view. Zitová divided feature detection methods from type of processing into the two groups in her article [69]: area-based methods and feature-based methods. On the other side, Ferrante and Paragios [17] and also Crum, Hartkens and Hill in their article [14] divided methods from matching criterion view and they are connecting feature detection and feature matching into one step called finding matching criterion (Ferrante and Paragios) or finding similarity measures (Crum, Hartkens and Hill). They divided methods into 4 groups: iconic (intensity based), geometric, sensor based and hybrid.

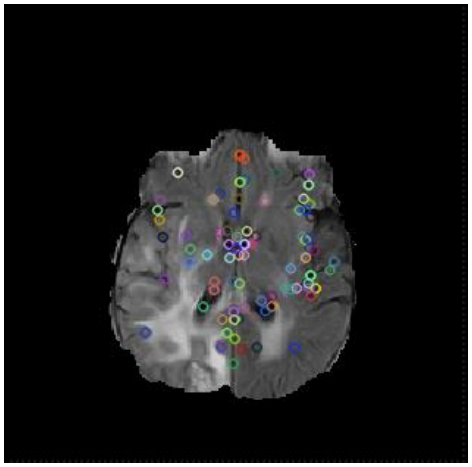


Figure 9: Set of key points detected on the image

5.1.1 Feature detection methods from processing view

5.1.1.1 Area-based methods

Area-based methods skip the feature detection step. No features are detected in these approaches, because it is merged with feature matching step. The algorithms are explained in the next chapter – feature matching.

5.1.1.2 Feature-based methods

Feature-based methods find significant parts of the images based on regions, lines or points. The algorithms can find features efficiently on the input images. They do not work with image values of the intensities directly, but they use some methods to get high level information of the images, so they can facilitate with differences of the images. Region features of the image are detected thanks high contrast boundaries of these regions. Region-based features are usually represented by their centers of gravity. Line features can be some line segments, region boundaries or object contours. They are usually represented by pairs of end or middle points. Point features usually define some local extremes.

5.2 Feature matching

Feature matching algorithms try to find correspondences between the features detected on the sensed image and features detected on the reference image as is shown in the Figure 10. The problem is, that if we had some incorrect features detected in previous step it spreads into this step. Another problem is that corresponding features can be dissimilar, because images might be scanned with different conditions. This problem should be solved with good choice of the feature descriptor. On the one hand, the descriptor must be able to distinguish among the different features, but on the other hand it cannot be influenced by noise and unexpected variations of the features. Result of description is a feature vector which is used for the feature matching.

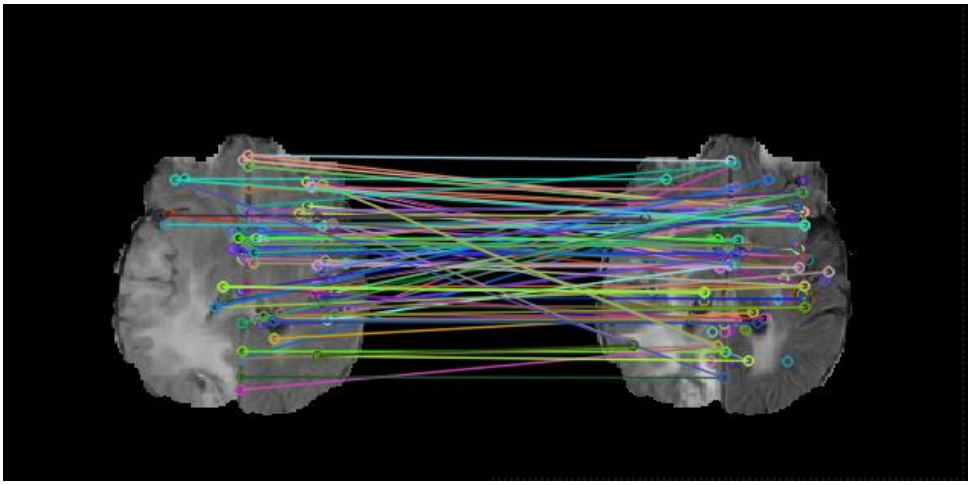


Figure 10: Matching of key points

5.2.1 Feature matching methods from processing view

5.2.1.1 Area-based methods

Area-based methods use correlation function or template matching algorithms. Thanks this, the area-based methods can skip the feature detection step and merge it with matching step. These methods do not detect any significant objects on the images. They are using windows or whole images which go through reference image to find correspondence. Some statistical values are calculated to evaluate the strength of the correspondences, such as correlation. These methods are very limited, because they are not appropriate for images that are deformed.

5.2.1.2 Feature-based methods

Input for the feature-based methods are two sets of key points – one from reference image and second one (or more) from sensed image. Future-based methods are divided on several types:

- Spatial relation algorithms are used if the surrounding of the key points is deformed. They are based on spatial relations, so they require information about the distance between the key points and spatial distribution.

- Invariant descriptors are an alternative of spatial region algorithms. Good descriptors must fulfill some conditions. The descriptions of the corresponding features must be the same – invariance condition. Descriptors of two different features must be different – uniqueness condition. And they must deal with local deformation – stability condition.

- Relaxation methods are based on labeling features from sensed image with labels of the features from reference image. The result of labeling are pairs of features – one from reference image and second from sensed image. Then the pairs are iteratively changed, depending on the strength of the match, until the situation is not stabilized.

5.2.2 Feature detection and matching methods from matching criterion view

5.2.2.1 Iconic (intensity based)

Iconic methods use information about intensity of processed pixel and its neighborhood. In volumetric data are used intensities of voxels to find similarities between two input data. The biggest challenge is to deal with different types of input data (they can be reached with different sensors – multimodal registration). In this case data have to be transformed to the same space where they can be easily compared. Matching of the similarities in multimodal data usually use complex statistical metrics such as cross correlation. On the other side, simpler methods, such as sum of absolute differences, can be used to find similarities between the monomodal data.

For example, Xiao et al. used iconic based registration for calibration of ultrasound during the patient examination in their survey [63]. Intensity based registration was also used for deformable registration of MRI and ultrasound images for neurology surgery by Hassan, Chen and Collins [49].

5.2.2.2 Geometric

Geometric methods use for registration a sparse set of salient image locations. It means that methods create explicit models of the elements on the image which can be easily identified. The best identifiable parts of the image are usually some curves, special surfaces or special points which can match with another input data. The pros of this type of methods is that mapping is biologically valid and allows anatomically and physiology correct transformation.

Geometric registration was used for example by Yavariabdi et al. [64] to register 2D ultrasound images to 3D MRI of woman pan. The main goal was to detect endometrial implants on both modalities. Endometriosis is a disease that affects women in fertile age.

5.2.2.3 Sensor based

Sensor based methods are not based on information from images, but on another kind of information – they work with information from some specialized sensors. The most usually types of sensors are optical and electromagnetic tracking systems. Optical systems, such as infrared camera or laser camera, track some special markers and it is helpful to identify position of tracked object. On the other side, electromagnetic tracking systems use transmitters to sensors and processor units to register location and orientation of the tracked object.

For example Polak in his surgery [47] used sensor based method for motion tracking in magnetic resonance image for retrospective volume reconstruction. MR images are very sensitive for motion. It is hard manly during the examination of the children.

5.2.2.4 Hybrid

Hybrid methods are some kind of combinations of previous three methods. They are usually used to reach better and more exact results.

One example of interesting hybrid registration which combined sensor based and iconic registration was developed by Bernhard et al. [18] to automatically register MRI and ultrasound images for

neurosurgery. It is required for practice to register preoperative MRI images with ultrasound images reached during the operation.

5.3 Transformation model estimation

In this step a model for transformation is estimated. Mapping functions are created to align the sensed images with reference image. Mapping function depends on the feature correspondences and must decide which correspondences are correct and will be matched and which will not be matched. Information about the image degradation is used in this step to create appropriate mapping functions.

Usually authors of articles [14][17] divide methods only from deformation point of view on rigid and non-rigid transformation also called elastic. However, Zitová et al. [69] and also Kybic [35] divided methods in their articles also by kind of mapping on global mapping models, local mapping models and mapping by radial basis function.

5.3.1 Division of transformation models from deformation point of view

5.3.1.1 Rigid

Rigid registration does not deform structure of the data. Basic functions of rigid registration - transformation are:

- Scaling – when size of the image is enlarged or reduced.
- Rotation – when the image is rotated several degrees around some point.
- Translation – when the image is shifted in some direction.
- Reflection – when the image is flipped over some line.

In domain of brain surgery is rigid registration usually used to register different types of modalities to reach more relevant information about the patient. One example of surveys in this field is survey by Colligon et al. who registered CT, MRI and PET volumes in their approach [11].

5.3.1.2 Non-rigid elastic registration

Elastic registration does not depend on the searching for the best values of the parameter as previous rigid registration which is based on parametric mapping functions. This new type of the registration was invented by Bajcsy and Kovacic [2].

The elastic registration is based on minimization of the strength energies in iterative process. We can imagine an image as a rubber sheet. Then internal and external forces act on that sheet to stretch or shrink it. While external forces stretching the image, the internal forces minimize the stretching depending on their stiffness constraints. In elastic registration methods, feature matching step is performed concurrently with mapping function calculation. This is a big advantage because no feature descriptors invariant to complicated deformations have been invented yet.

Well known non-rigid registration methods are fluid registration, optical flow, diffusion-based registration and level sets.

Non-rigid registration can be also used for segmentation of some anomalies if we register data on atlases as is mentioned in previous chapter 4.4 Atlas guided segmentation. Kyriacou et al. used elastic registration to register brain MRI data on Brain atlas and to detect anomalies – tumors in the volumes in their approach [36].

5.3.2 Division of transformation models from mapping point of view

These methods are used for different types of rigid registrations and need to count parameters to perform some transformation.

5.3.2.1 Global mapping models

One of the most popular and the simplest global mapping model is similarity transform. It is a linear model defined by two key points. This model preserves the shapes and it uses only translation scaling and rotation.

Another linear model is affine transform. It is defined by three key points. The advantage of the model is that it preserves straight lines and their parallelism. The most usual usage is registration of different viewports, but the scene must be larger than scanned area, scene must be flat and geometry cannot be disturbed by any local factors.

If the scene is smaller than scanned area, then the perspective projection model should be used. It is defined by four independent key points and describes the deformation of the flat scene.

5.3.2.2 Local mapping models

In the previous chapter, the methods created mapping function according to global deformation of the images. However, images can be deformed also locally. It is typical for medical imaging. Mapping functions register information about the local geometric changes.

5.3.2.3 Mapping by radial basis function

Radial bases functions are also able to work with local geometric distortions, but they are using some combinations of global mapping methods to create mapping functions. They are linear combinations of translation function polynomials. They were originally developed for interpolation of the irregular surfaces. The most important property of radial basis function is that they depend only on the distance of the key point, not on the concrete position of the key point. The most popular radial bases functions are Gaussian, Wendland, multiquadrics and thin-plate splines (TPS).

5.4 Transformation

Mapping functions which were created in the previous step are used for the transformation of the sensed images. There are two different approaches:

- Forward manner uses mapping functions for direct transformation of every pixel from sensed image. It can produce overlaps or holes, so it is hard for implementation.

- Backward manner uses inverse of the mapping functions and coordinates of the target pixel based on reference image to compute data for sensed image transformation. It is easier for the implementation, because algorithm do not produce any overlaps or holes.

5.5 Validation

Validation means to show that registration was correct. It is sometimes really hard task to evaluate correctness of the registration as writes Crum, Hartkens and Hill in their article [14]. Actually, average volume error of the registration can be estimated, but only in case when registration is rigid. It is count as distance between the corresponding landmarks. This kind of analysis is not possible to use for non-rigid registration techniques, because deformation is not consistent with real world. Thus, validation is always performed to show on the dataset that transformation worked as we excepted for this kind of registration.

6 Neural networks

Neural networks are simplicity inspired by biological neural system as is mentioned in the book by Kriesel [32]. It means that they can progressively learn to do some task from some examples. Neural network can learn how to do some task without being explicitly programed. For example, neural network can learn to distinguish between different type of flowers on the images. Using many examples of those flowers with labels, neural network can detect right type of flower on the new input image.

Neural network consists from many neurons which are connected such as brain. Those neurons are called artificial neurons. Mathematically, we can represent the neural network as a directed graph where neurons are nodes and connections are weighted edges.

In the Figure 11 we can see a schema of one neuron, where:

- x_1 ... x_n are inputs,
- w_1 ... w_n are weights of connections and
- b is bias.

The inputs are multiplied by weights and summed up inside computing unit. Weights are crucial to solve the problem. They represent the strength of connection between neurons. Result of the sum may be number between 0 and ∞.

Thus, activation function (also called transfer function) is used to control the results. Activation function can be defined by different function. The simplest is binary activation function (threshold), which can reach only two values – 0 or 1. Another functions usually fall into the linear or sigmoid category of functions.

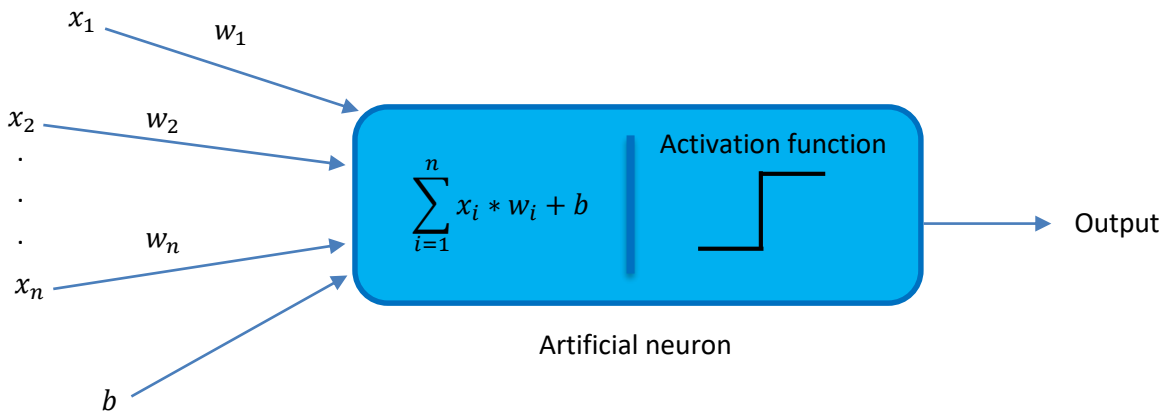


Figure 11: schema of the artificial neuron

We need sophisticated neural networks to solve hard problems. More difficult neural networks consist of several layers and every layer consist from several artificial neurons. Typical neural network consists of:

- input layer,
- several hidden layers and
- output layer.

Input layer contains artificial neurons which receive input from outside world. Input is used to learn network – find the best values of the weights to make network capable to process and make decision about new inputs. Between input and output layer can be several hidden layers. Their main job is to transform input into something that output layer can use. Finally, output layer contains neurons which react on the information how network learned the task. Layers are usually fully connected – it means that every neuron in one layer is connected with all neurons in its previous layer and next layer. Typical architecture of neural network is displayed in the Figure 12.

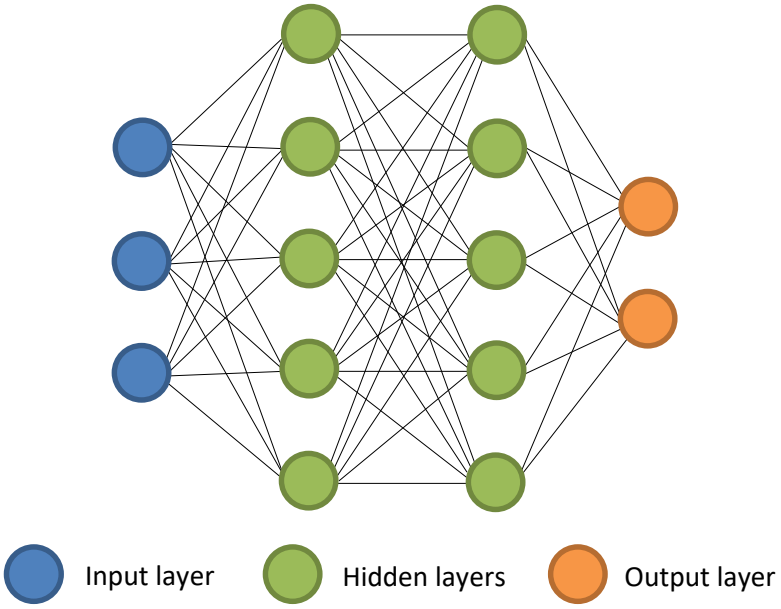


Figure 12: Architecture of neural network

On the webpage of Asimov institute [83] is a list of different kinds of neural network architectures. In the Figure 13 are shown schemas of architectures from the web page. However, the schemas do not explain how neural networks work, they only show how different neural networks are composed. Since many different architectures exist in the next subchapters are explained only two of them: perceptron (because it is basic architecture) and convolutional neural network (because we are interested in this kind of neural network architecture in our approach).

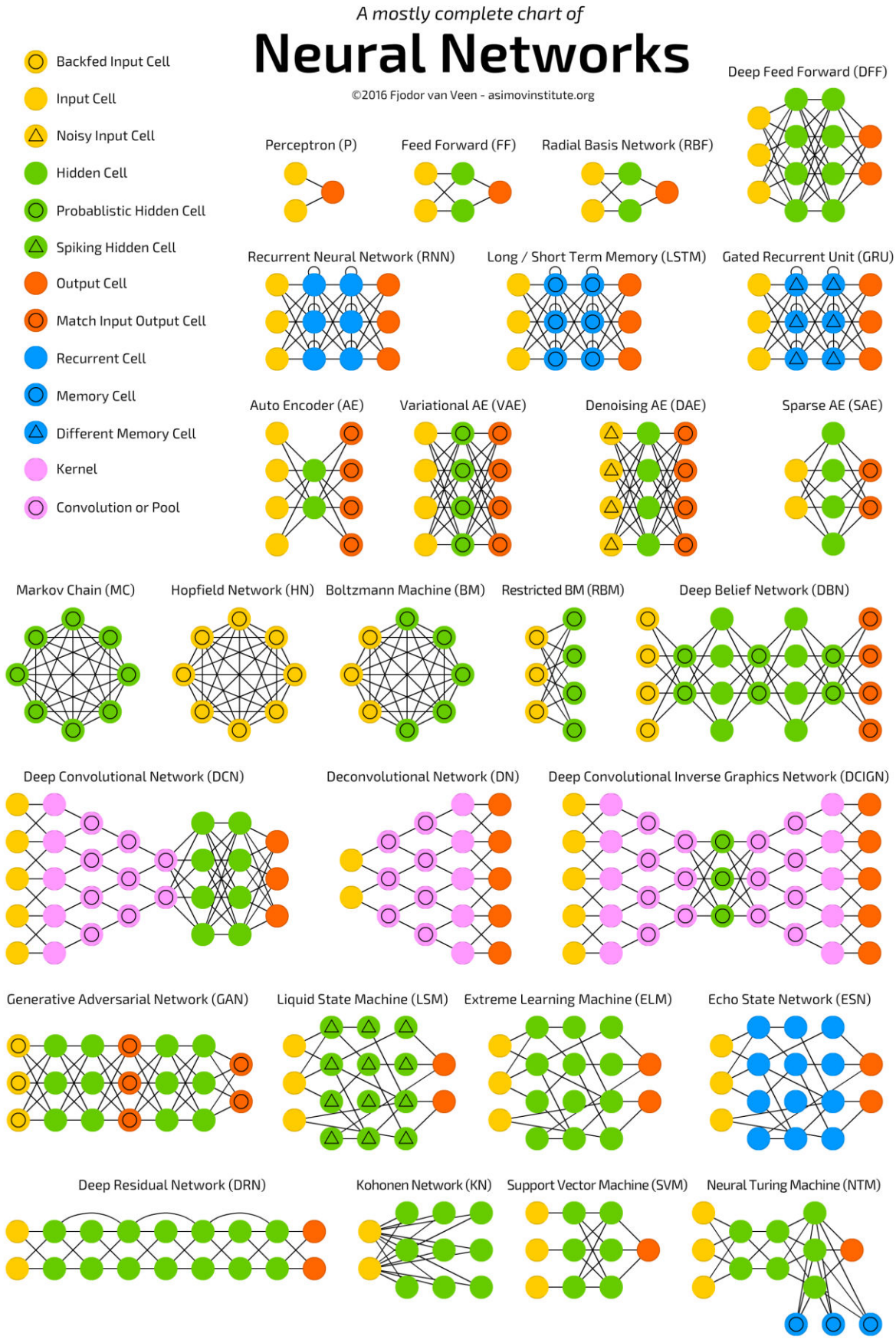
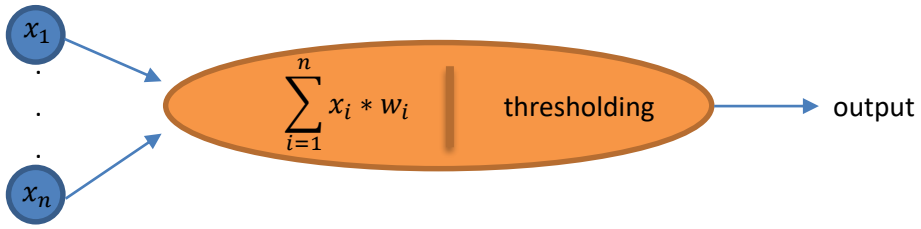


Figure 13: Different architectures of neural networks by Asimov institute [83]

6.1 Perceptron

Perceptron is one of the simplest neural network developed by Rosenblatt in 1958 [50]. It is very well explained in the online book by Goodfellow et al. [20]. It consists only from input and output layer, so it does not have any hidden layers as is shown in the Figure 14. Input layer can have two or more input



units and Rosenblatt designed rules to compute the output.

Figure 14: Schema of perceptron

In the research, he introduced weights. That are real numbers which determine the importance of the inputs to count correct output. Output can have values 0 or 1 depending on the formula:

$$output = \begin{cases} 0 \; if \; \sum_i x_i w_i \; \leq threshold \\ 1 \; if \; \sum_i x_i w_i \; > threshold \end{cases}$$

Then the perceptron learns correct outputs depending on the input data. Perceptron get some input and it returns some output. That output is compared with estimated output and the error is counted. Then the error is used to edit weights.

6.2 Convolutional networks

First modern convolutional neural network was presented in paper by LeCun, Bottou, Bengio and Haffner in 1998 [37]. This algorithm is very well explained in the online tutorial by Michael Nielsen last edited in 2017 [45]. Convolutional networks are a little different from another networks because they are primary used for image processing. This is the reason why we are interested at it. Typical task for convolutional network is to classify images into different classes and she need to learn on many input examples.

Convolutional networks are not fully connected as other neural networks. Nodes are connected only with the nearest neighbors – it depends on the implementation. Convolutional neural networks use 3 basic ideas for image recognition:

- **Local receptive fields:** In typical networks are neurons ordered into the vertical line, but in convolutional networks are neurons ordered to the matrix. Then the input matrix is passed to the hidden layers, but not each pixel is connected with all hidden neurons. There are connections only between small regions from the input image. It means that neuron from hidden layer will be connected only to some region from input layer and that region is called local receptive region. Of course, each connection has a weight. Thus each region from the

input image is connected with neuron on the hidden layer. The algorithm starts at left top corner and slide over one pixel in every step through whole image. This process is called convolution. Finally, size of the hidden layer is smaller because of margins of the image. Size of the reduction depends on the size of the regions.

- **Shared weights:** Each connection to hidden neuron have a weight matrix (size of the weight matrix is the same as size of the region from input image) and a bias. Those weights and value of bias is shared for all hidden neurons. It means that output value of the hidden j, k^{th} neuron is defined as:

$$\sigma\left(b + \sum_{l=1}^{n}\sum_{m=1}^{o} w_{l,m} x_{j+l,k+m}\right)$$

Where σ is activation function, b is shared value of the bias, $w_{l,m}$ is a shared matrix of weights and $x_{j+l,k+m}$ define positions of the hidden neurons. In other words, every neuron in the hidden layer detect the same one feature. This process is usually denoted as creation the feature map. If we want to extract more features from the input image, we need to create more feature maps. Different feature map can be created using different filter (also called kernel) which is defined by the weights and bias. More about the features, feature maps and filters is written in the article by Zeiler and Fergus [67].

- **Pooling:** After the convolution layer is usually used pooling layer which simplify the information from the convolutional layer. Thus, feature map is processed again and max pooling creates condensed map as summarization of some small regions. It again creates smaller layer. One of the best known algorithm is max-pooling algorithm which compute the maximum from given region. There are also some other pooling algorithms. Of course, pooling must be done separately for each feature map.

7 Medical background of the brain cancer

The brain consists of many and many neurons [57]. That's are special cells which are located only in the brain. Every cell has got some life cycle. It means that old cells die, so new cells must be created. During the creation and cell growing process can occur some mutation of the cell. Mutation of the cell causes that the original functionality of the cell is lost. If there are more and more mutated cells and they take some regions in the brain, they create the tumor.

The growth of the tumor may cause some problems which are considered as symptoms of the brain cancer. Type of the symptoms depends on the location of the tumor in the brain - the tumor pushes on different brain centers and cause different problems. First signs of the tumor cancer are strong and particular headaches. Tumor may cause speech disorders, memory problems, movement disorders and so on.

It is very important to recognize the first symptoms of the brain cancer and start to solve them. Early diagnostics may be crucial for the patient. In the past, it was not possible to see tumor without direct interference with the brain, which was not also possible because medicine was not so far. It was possible only to determine the position of the tumor depending on the symptoms.

The invention of the MRI and CT scanners allowed to see brain structures and get more information. The typical process of MRI or CT image evaluation consists two steps. At first, one specialist has to inspect all images and make diagnosis. In addition, conclusions of the first specialist must be consulted with another one. That is long and time-consuming process. Nowadays, special programs are used to highlight tumor boundaries on the slices. However, without computer vision methods it is still time-consuming and monotone task. Thus, computer vision finds here its application.

7.1 Magnetic resonance imaging – MRI

MRI scanners are based on the magnetic fields and create grey scale images of the organs in the body [34]. The patient lies in the special scanner where strong homogenous magnetic field acts. A short radiofrequency pulse is transmitted to the patient's body. It bounces back from the body structures as a weak signal which is finally used for image reconstruction. The signal is scanned with different settings, so different modalities are created. The most usual modalities are Flair, T1, T2 and T1c with contrast substance.

MRI is the most often used technique during the brain cancer diagnostics. Patients absolve more MRI tests at several monthly time intervals. MRI data are not used only for diagnostic, but also for the healing process and tumor changes monitoring.

7.2 Data

The result of the MRI are volumetric data. That are three-dimensional data – a set of images called slices. MRI data are not saved in classical image formats such as jpg or png. They are stored in special medical formats such as DICON, NIFTI, MHA and many others. It consists of the header with important information and body where scanned values are stored.

Brain MRI data may be visualized using special 3D visualization methods, but doctors often uses printed 2D images - slices for diagnostic. They can be printed and also visualized in computer in three different directions:

- axial,
- coronal and
- sagittal.

Examples of different views are shown in the Figure 15, where left image is axial, image in the middle is coronal and right image is sagittal view.

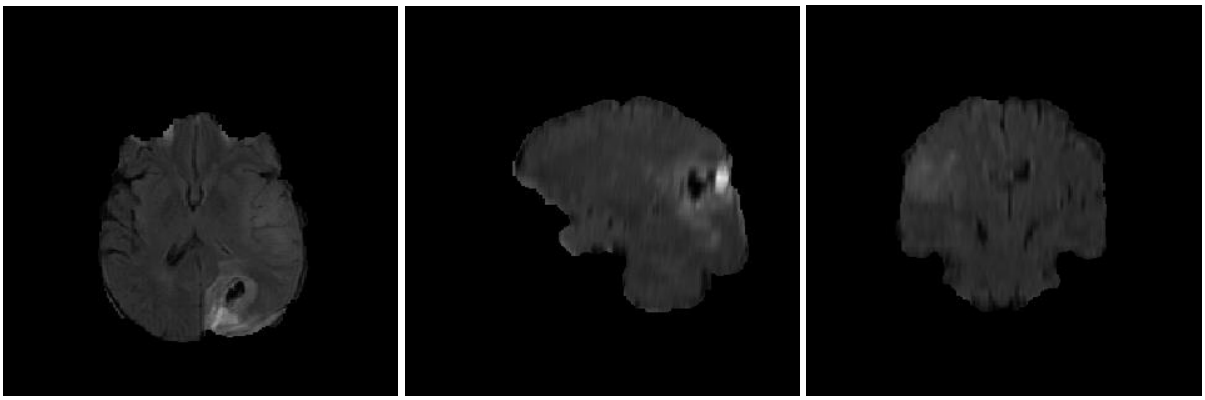


Figure 15: Axial, coronal and sagittal view of the brain MRI

8 State of the art

This part is devoted to the most interesting state of the art methods in the field of medical imaging, mainly tumor segmentation methods and brain registration methods.

8.1 Segmentation

8.1.1 Brain Tumor Segmentation Benchmark (BRATS)

BRATS is a challenge aimed on brain tumor segmentation. It is a part of BRAINLES workshop which take place every year from 2012 as part of International Conference on Medical Image Computing and Computer Assisted Intervention - MICCAI. This year (2017) BRATS has got two challenges:

- segmentation of gliomas in pre-operative scans and
- prediction of patient overall survival from pre-operative scans.

In the last year, one of the challenges was monitoring of the tumor changes.

BRATS provides own dataset of MRI brain data with tumor disease [72] and also labels of tumor segmentations made by experts. Dataset is divided on training and testing data and is increased every year. In the last year challenge (2016) it contained 191 subjects in testing dataset and 200 subjects in training dataset. Dataset for BRATS challenge 2017 is not available yet. In the BRATS dataset every subject contains flair, T1, T1c and T2 MR modalities.

Prof. Dr. Bjoern Menze is one of BRAINLES organizers and he is in charge of BRATS. In 2015 Menze et al published an article [42] about results from BRATS 2012 [60] and BRATS 2013 [41] challenges. Twenty algorithm for tumor brain segmentation were presented on BRATS challenges – 17 fully-automatic and 3 semi-automatic algorithms.

In the article is also mentioned how data were labeled by expert. Authors wrote that edema was segmented from T2 images and Flair was used to cross-check the segmentation. The tumor core (which consists also from enhancing and non-enhancing tumor structures) was segmented from T1-c and T1 modalities. Then threshold of T1c was used to segment enhancing core. Necrotic core parts were defined as the tortuous and low intensity necrotic structures and finally the non-enhancing structures was segmented as subtraction of the enhancing core and necrotic core structures. Manual segmentation was necessary for the evaluation of the methods for automatic segmentation.

For the evaluation of the segmentation organizers of BRATS challenge created the benchmark. It computes accuracy, sensitivity, specificity, Dice score and also Hausdorff score:

- Sensitivity: $sens(P,T) = \frac{|P_1 \wedge T_1|}{|T_1|}$
- Specificity: $spec(P,T) = \frac{|P_0 \wedge 0|}{|T_0|}$

- Dice score: $Dice(P,T) = \frac{|P_1 \wedge T_1|}{(|P_1|+|T_1|)/2}$, where T_1 is true lesion area, T_0 is normal area, P_1 is predicted to be a lesion and P_0 is predicted to be normal as is shown in the Figure 16.

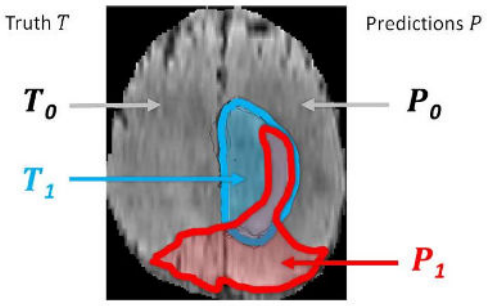


Figure 16: Truth segmentation by the expert (T_1 and T_0) and prediction of the algorithm (P_1 and P_0)

Authors evaluated all methods using accuracy, specificity, Dice score and Hausdorff score. They evaluated segmentations of whole tumor, core tumor and active tumor part. They compared results of the methods and in the Figure 17 are shown results of sensitivity and specificity comparison. In the Figure 18 are presented the results of Dice score evaluation.

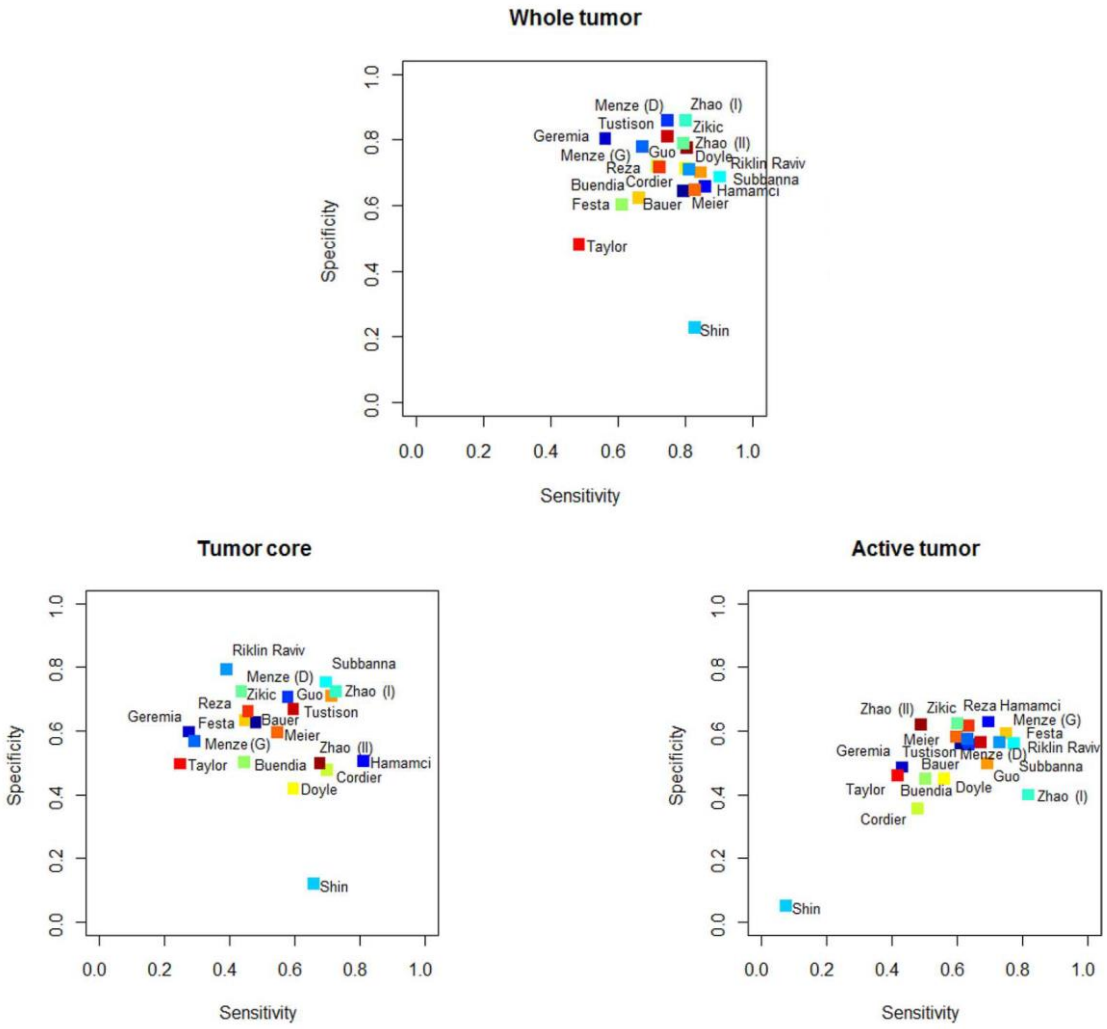


Figure 17: Distribution of Sensitivity and Specificity evaluation results of tested methods [42]

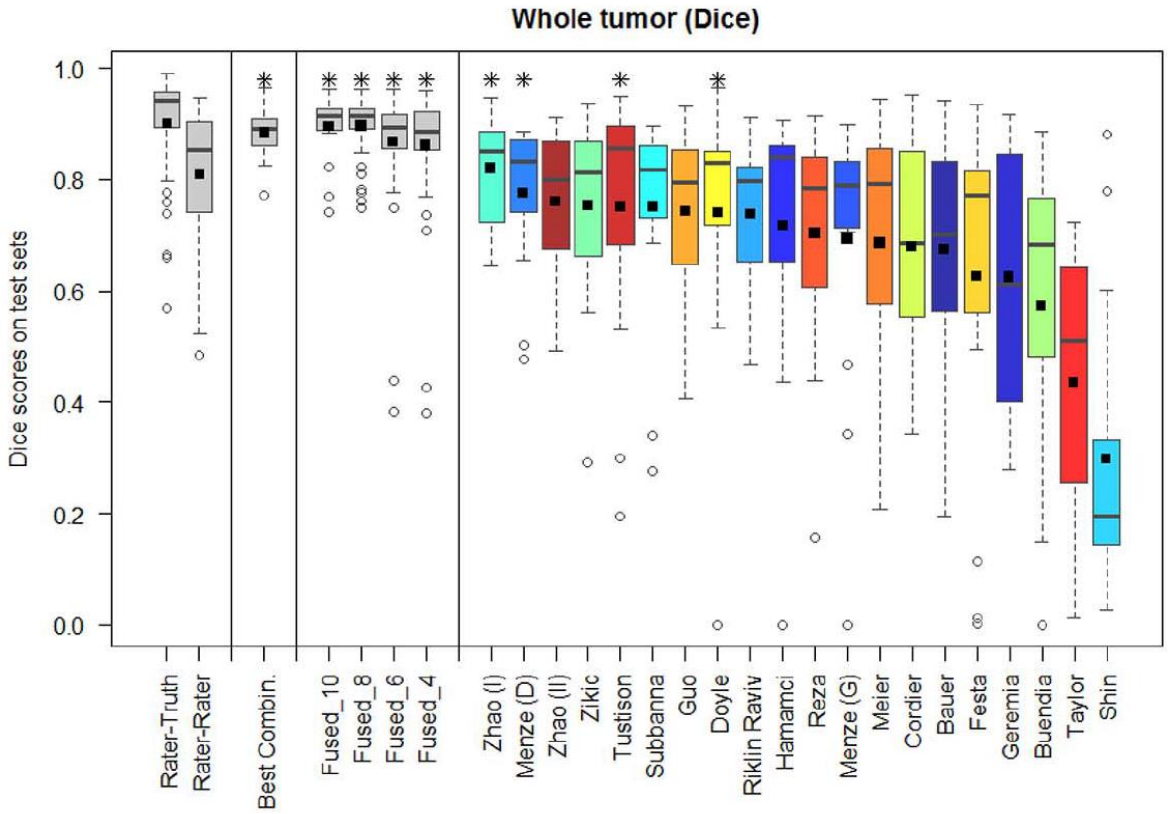


Figure 18: Dice score evaluation results of the methods (only whole tumor) [42]

In the figures we can see that Zhao(I), Menze(D) and Zhao(II) methods reached the best results in case of whole tumor segmentation. The methods are based on:

- Zhao(I) – Learned Markov Random Field on Supervoxel Clusters
- Menze(D) – Generative-Discriminative Lesion Segmentation Model
- Zhao(II) – the same as Zhao(I), but with updated unary potential

There is not any new summarization of BRATS 2014 – BRATS 2016 available, but proceedings of the workshops are: BRATS 2014 [43], BRAINLESION 2015 [48] have also official journal [13] and BRAINLESION 2016 [8] have also official journal [12]. BRAINLESION workshop consist of:

- brain lesion image analysis fully
- brain tumor image segmentation improving (originally BRATS)
- ischemic stroke lesion image segmentation predicting

I attended BRAINLES 2016 workshop and as the best method from BRATS 2016 was evaluated Chang's method - Fully Convolutional Neural Network with Hyperlocal Features. The method exceeds the state of the art with Dice score of whole tumor segmentation 0.87. This method and also some other interesting methods are explained in the next chapters.

8.1.1.1 Brain Tumor Segmentation Using a Fully Convolutional Neural Network with Conditional Random Fields [12]

Zhao at al. proposed new method for tumor segmentation based on the fully convolutional neural networks (FCNN) and conditional random fields (CRF) used for post-processing step after FCNN. Their method consists of three steps: pre-processing, segmentation using the FCNN and post-processing.

Authors describe that preprocessing is necessary, because devices of MRI are not the same, so intensity ranges and bias field can be different. They decided to use N4ITK method to correct bias fields and normalize the intensity.

Their model for tumor segmentation consists of FCNN and CRF and in train phase can segment tumor from 2D slices. FCNN was trained on two dimensional slices and the architecture is shown in the Figure 19.

The input for FCNN have got two different sizes. Bigger region is passed through several convolution and pooling layers and feature map is created. Then the smaller region is added and it again passes several convolution and pooling layers. Finally, the label of center pixel is predicted. Then the predictions are optimized using CRF algorithm. Finally, in post-processing phase, authors remove small 3D-connected objects using simple threshold method.

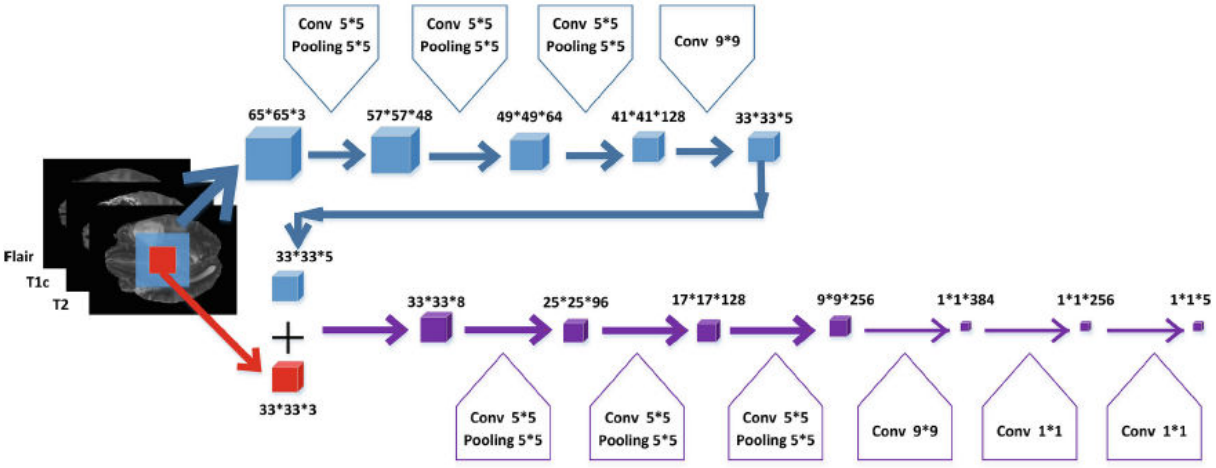


Figure 19: Architecture of FCNN

Authors evaluated the method on BRATS 2013 dataset and reached first position of all methods tested on this dataset. Their method also reached good results using only three modalities (Flair, T2, T1c) rather than all four which can reduce the cost of data acquisition.

8.1.1.2 Fully Convolutional Neural Networks with Hyperlocal Features for Brain Tumor Segmentation [8]

Chang proposed simple architecture of the convolutional neural network and used also local features for brain tumor segmentation. His method is based on fully convolutional architecture. Output is a classification matrix with the same size as input image. The activation functions from the deepest convolutional output layer are combined with local features. Architecture of the proposed method is

show in the Figure 20. Proposed method is very fast. It can segment the tumor from whole volume in less than one second, because network is composed only from 130400 parameters.

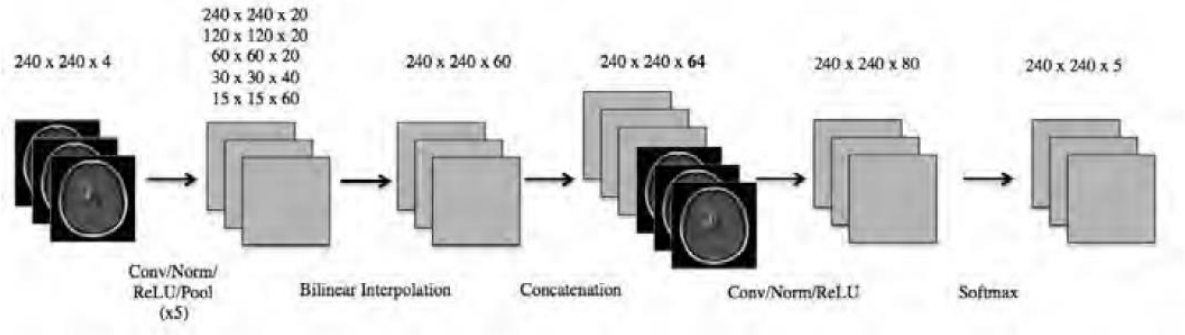


Figure 20: Architecture of convolutional neural network [8]

Authors tested proposed architecture on BRATS 2015 dataset and evaluated the results and compared them with BRATS 2016 leaderboard. Only dice score and Hausdorff were presented in their article. They reached 87 % dice score and 9,1 Hausdorff for whole tumor segmentation.

8.1.1.3 Image Features for Brain Lesion Segmentation Using Random Forests [13]

Maier, Wilms and Handels used random decision forests for tumor segmentation. They tested different features and found best combination to reach good results in segmentation. Their method consists of pre-processing, random forest classification and post-processing step.

In the pre-processing phase, authors normalized the intensities and corrected biases. In the next step authors trained the classifier on the subset of data. They have shown that carefully selected subset of data can reduce training time with minimal effort to the accuracy. Model was trained using random forest algorithm which output is a probability map. Finally, in post-processing phase they selected the most probable classes for each voxel from probability map.

Authors tested their method on BRATS 2015 and also on ischemic stroke lesion dataset and reached very good results in both cases. Thus, their method is robust for segmentation tasks in the brain MRI volumes. In the BRATS 2015 leaderboard they reached dice score 75 %, sensitivity 88% and specificity 71% for whole tumor segmentation.

8.1.1.4 Learned Markov Random Field on Supervoxel Clusters [42]

Zhao, Srakaya and Corso removed the noise from the data and normalized them in the preprocessing phase. During the normalization they put the data to the same scales using z-score (zero mean and unit covariance). In the next step they created supervoxels using SLIC algorithm. SLIC algorithm cluster pixels into the voxel by color similarity and proximity in the volume.

In the next step they created Markov random field and used it for the graph cut segmentation. Of course, graph cut needs some input information about probably foreground and background. Thus, they used training data to create histogram and then the histogram was used during the testing to create probably background and foreground.

This method was explained in the summarization article by Menze et al. [42] and evaluated on BRATS 2013 dataset. The results of the method are presented in the figure: sensitivity and specificity in the Figure 17 and Dice score in the Figure 18.

8.1.1.5 Generative-Discriminative Lesion Segmentation Model [42]

Menze et al. proposed fully automatic method for chanel-specific tumor segmentation. The method extends general EM segmentation algorithm to solve problems with specific spatial structures which cannot be described sufficiently. Their method models a tumor as a multi-dimensional sequences. It allows channel-specific segmentation of the tumor. That model also contains the information about the location of the lesion. Method also uses information about the specific signals of healthy brain parts thanks brain atlases.

This method was explained in the summarization article by Menze et al. [42] and evaluated on BRATS 2013 dataset. The results of the method are presented in the figure: sensitivity and specificity in the Figure 17 and Dice score in the Figure 18.

8.1.1.6 Brain Tumor Segmentation with Deep Neural Networks [43]

Davy et al. proposed novel architecture of neural network for tumor brain segmentation. In the first step, they preprocessed the data, they removed 1% of the lowest and highest intensities and applied N4ITK filer.

Then, they trained model using convolutional layer, maxout convolutional layer, max pooling layer, fully connected layer, fully connected Maxout layer, Softmax layer and Droput. Architecture of proposed method is shown in the Figure 21.

The main job of the used layers is:

- *Convolution layer* is used to model features by a set of different kernels.
- *Maxout convolutional layer* is special convolutional layer, where one feature map can be associated with more kernels – feature map is calculated using more kernels.
- *Max pooling layer* track the maximum feature value over sub-windows.
- *Fully connect layer* connects all units of the layer to all units in previous layer (this is not typical for neural networks as is mentioned in chapter 6.2 Convolutional networks).
- Then the *fully connected maxout* layer is only a fully connected version of convolutional maxout layer.
- *Softmax layer* is a special case of fully connected layer, but with softmax activation function.
- Finally, *dropout* is a regularization method that adds noise in the computation of the hidden layers.

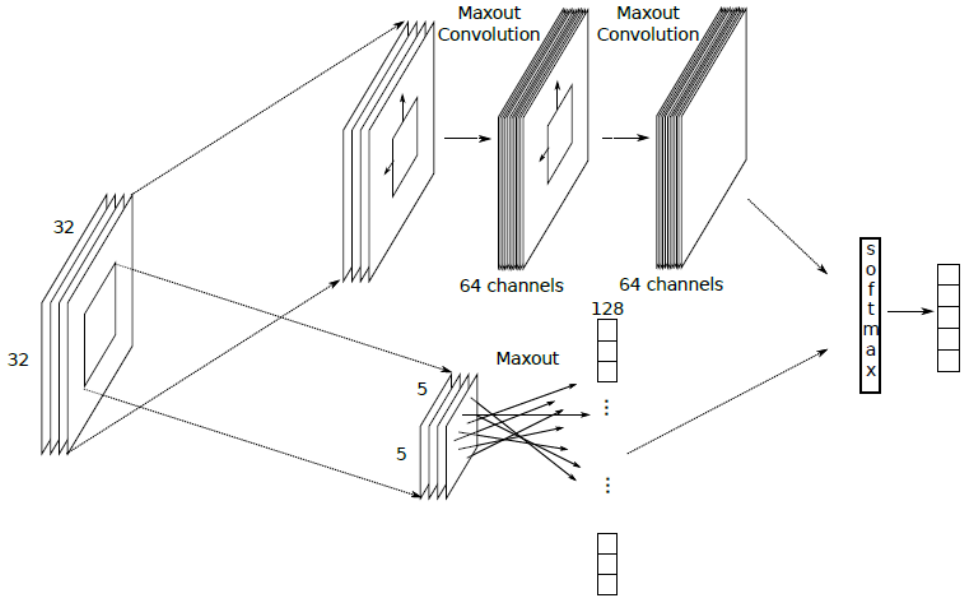


Figure 21: Proposed architecture of deep neural network

Authors tested proposed architecture on BRATS 2013 dataset and evaluated the results and compared them with BRATS 2013 leaderboard. They reached dice score 74 %, sensitivity 78% and specificity 74% for whole tumor segmentation.

8.1.2 Brain tumor segmentation with Deep Neural Networks

The method is based on deep neural network and is refinement of previous method mentioned in the previous chapter 8.1.1.6. Authors proposed 4 different architectures based on the convolutional neural network and compared results of each trained model with other state of the art method. Results of the comparison from the article are shown in the Figure 22. As we can see, they reached the best results with input cascade CNN method.

Method	Dice			Specificity			Sensitivity		
	Complete	Core	Enhancing	Complete	Core	Enhancing	Complete	Core	Enhancing
InputCascadeCNN*	0.88	0.79	0.73	0.89	0.79	0.68	0.87	0.79	0.80
Tustison	0.87	0.78	0.74	0.85	0.74	0.69	0.89	0.88	0.83
MFCascadeCNN*	0.86	0.77	0.73	0.92	0.80	0.71	0.81	0.76	0.76
TwoPathCNN*	0.85	0.78	0.73	0.93	0.80	0.72	0.80	0.76	0.75
LocalCascadeCNN*	0.88	0.76	0.72	0.91	0.76	0.70	0.84	0.80	0.75
LocalPathCNN*	0.85	0.74	0.71	0.91	0.75	0.71	0.80	0.77	0.73
Meier	0.82	0.73	0.69	0.76	0.78	0.71	0.92	0.72	0.73
Reza	0.83	0.72	0.72	0.82	0.81	0.70	0.86	0.69	0.76
Zhao	0.84	0.70	0.65	0.80	0.67	0.65	0.89	0.79	0.70
Cordier	0.84	0.68	0.65	0.88	0.63	0.68	0.81	0.82	0.66
TwoPathCNN	0.78	0.63	0.68	0.67	0.50	0.59	0.96	0.89	0.82
LocalPathCNN	0.77	0.64	0.68	0.65	0.52	0.60	0.96	0.87	0.80
Festa	0.72	0.66	0.67	0.77	0.77	0.70	0.72	0.60	0.70
Doyle	0.71	0.46	0.52	0.66	0.38	0.58	0.87	0.70	0.55

Figure 22: Results of proposed methods presented in the article [24]

Input cascaded CNN exploits the effectivity of CNN architecture and directly models dependencies between neighboring labels in segmentation thanks two cascades. Output probabilities from the first CNN cascade are used as direct additional inputs to the second CNN cascade as is shown in the Figure 23. Thus final predictions are influenced by the model beliefs that the value is nearby other same labels. In many other state of the art method this problem is solved using some post-processing methods.

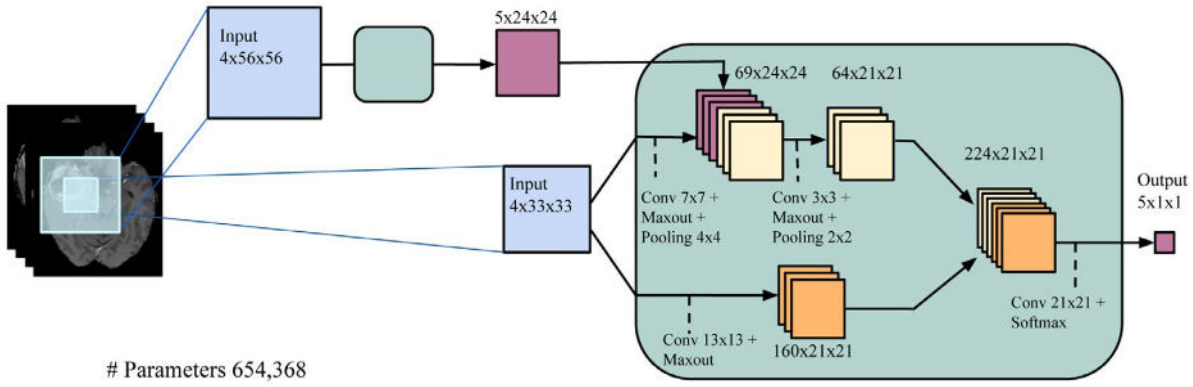


Figure 23: Architecture of input cascaded CNN presented in the article [24]

Authors trained proposed CNN on BRATS 2013 training dataset and they used three methods to improve the segmentation results:

- **Gradient descent optimization** was used to maximize probability of all labels in the training set and minimize negative probability.
- **Two phase training** was used to minimize the problem with unbalanced distribution of labels. While health brain parts make up 98%, only remaining 2% belong to pathology. Because of this the model can be overwhelmed and cause problems with training. Thus in the first training phase authors constructed dataset of equivalent-probable labels and then in the second phase they retrain the model only on the output layer to take into account unbalanced data.
- **Regularization** was used to prevent the overfitting. Authors applied L1 and L2 regularization and also dropout algorithm.

8.1.3 Automatic Brain Tumor Segmentation using Cascaded Anisotropic Convolutional Neural Networks [61]

Wang, Li, Ourselin and Vercauteren proposed cascaded architecture of the convolutional neural network for brain tumor segmentation. The cascade is used to decompose the multi-class segmentation into a sequence of 3 binary segmentations where resulting segmentation of first cascade is used as the mask for second cascade.

Architecture of each cascade consists of several convolutional layers where anistropic and dilated convolution filters were used. Whole tumor is segmented in first cascade, then tumor core is segmented in the next cascade and finally enhanced tumor core is segmented in the last cascade.

Authors tested their method on BRATS dataset and evaluated the results and compared them with BRATS 2017 leaderboard. They reached dice score 90,5 %, sensitivity 91,18% and specificity 99,42% for whole tumor segmentation.

8.1.4 Efficient Brain Tumor Segmentation in Magnetic Resonance Image Using Region-Growing Combined with Level Set [25]

Hien and Binh proposed method based on region growing algorithm for tumor segmentation from MRI volumes. Region growing methods are usually semi-automatic and requires simple user interaction, but they create initial seed automatically from the preprocessed image. In the first step, they preprocess the data. Thus, they compute histogram equalization and use it to enhance the image. It shows tumor regions if is present.

In the next step they use, enhanced image to find an initial seed for region growing method. Then they run region growing algorithm in several iterations – while value of the nearest pixel minus mean value of region (set of pixels from previous iterations) is smaller than predefined threshold. Result of this step is segmented image and in the next step, authors create exact boundaries of the tumor using level set method which is appropriate to analyze motion of the boundary.

Authors used own dataset for evaluation and computed Jaccard Index to evaluate the segmentation. Their best segmentation reached Jaccard index value equal to 87,69 – which is better than simple region growing algorithm (81,15).

8.2 Registration

8.2.1 Atlas to Patient Registration with Brain Tumor Based on a Mesh-free Method [15]

This is the article from 2015 and authors Diaz and Boulanger present a novel method for registration of the MRI brain images with tumor disease to the brain atlases. As is mentioned in the article, it is a difficult task because tumor causes deformation of the brain structure. Thus, traditional methods based on intensities and shape do not achieve good results. Authors proposed the method which simulates brain changes caused by the tumor growth and pressure.

The method consists of three steps. In the first step, the tumor is segmented from the MRI and registered to the atlas. The second step is a simulation of the tumor growth and consists of two main sub-steps:

- sampling and seeding the atlas to simulate tumor growth: Boundaries of the segmented tumor are shrinked in several iterations to a singular seed point – tumor center. This process depends on the level set method. In every iteration is created a list of new tumor boundary points and links to their previous positions – mapping function.

- mass-effect simulation based on mesh-free method: The method has got the mapping function of the tumor growth, so it is applied for the atlas in this sub-step.

Final step of the method is deformable registration of the atlas. Binary image which represents brain tissue was used for the registration. The reason is that method do not have to deal with different

intensities between source and target image, so it gives better results. Then the registration is applied to the atlas in intensity scale.

Authors compared their method with two other MRI registration algorithms. They used 12 images from BRATS dataset. They used metric based on distance fields for the evaluation. Their results show that their method outperformed two other tested methods and their method was also faster.

8.2.2 PORTR: Pre-Operative and Post-Recurrence Brain Tumor Registration [33]

Kwon et al. proposed new method for non-rigid registration of pre-operative and post-recurrence brain MR scans. Authors called their method PORTR. This is challenging task because tumor and edema cause big deformations and inconsistent intensity profiles between the scans.

PORTR aligns pre-operative scan with the post-recurrence scan of the same patient. In the first step, authors segment tumor and tumor parts from both scans to extract pathological information. They use atlases for tumor segmentation – they register atlas to both scans (pre-operative and post-recurrence) and divide the tissues on health and pathological. This is possible, because atlas provides spatial information and expecting intensity values about health brain parts and this can be used to detect pathology.

Then they exclude pathological regions from the scans and provide rough registration based on the tumor boundaries. Registration is in the next step improved using symmetry information of the brain. Finally, they measure overlapping between aligned tumors.

They apply their method on 24 glioma patients with T1, T1c, T2 and flair modalities. They collaborated with experts who placed the landmarks on some specific brain parts on both per-operative and post-recurrence scans. Then they used this information for the evaluation to measure distance between registration created by the method and by the expert. Authors compared quantitative and also qualitative results with other methods and they reached very good results in comparison with state of the art. In the 10 randomly selected subjects they reached smaller error of the registration than 5 other methods. Only one method had better results than PORTR.

8.2.3 Registration of Brain Images with Tumors: Towards the Construction of Statistical Atlases for Therapy Planning [66]

Zacharaki, Shen, Mohamd and Davatzikos proposed method for registration of brain scans with tumors for therapy planning. Authors decided to register brain scans to the atlases of the brain. The motivation is that during the therapy planning, doctors need to know position of healthy and important brain parts to protect them as best as possible. Atlases can provide this information, but tumor changed the structure of the brain, so registration is not simple. Deformable registration is necessary, because tumors cause irregular changes of the health brain parts.

The main goal of the proposed method is to determine anatomical correspondences if exist, rather than matching image intensities. The method is based on estimation of the tumor growth model using the deformations caused by the tumor. And more, the method is robust to the little inaccurate estimations of tumor growth simulation. Authors use the nonlinear biomechanical model to simulate tumor growth

in the atlas. The method tries to minimize differences between scan and atlas durng the tumor growth simulation.

Authors used own method for the registration based on deformation around the tumor because it can be indicative for the accuracy of the model estimation of tumor growth. They evaluated their method on real MR dataset, but they do not compare it with other algorithms.

8.2.4 A Model of Tumor Inducted Brain Deformation as Bio-Physical Prior For Non-Rigid Image Registration [39]

Authors of the article Mang et al presented new method for registration of the brain threatened by tumor to a standard atlas of the brain. Authors used studies about density of cancerous cells and information about soft tissues to create non-linear weighted function to control the deformation of the brain cause by tumor growth.

The method is very complicated and is based on real studies from medicine thanks cooperation with some medical institution. It uses many functions in respect to brain and tumor composition. Thus, proposed method of the non-rigid registration and computation of the mapping function are based on the changes from the real approaches. Thanks this, the method reached very good results. However, authors said that it is hard to evaluate results, because there is no general method for registration evaluation.

8.2.5 An EM Algorithm for Brain Tumor Image Registration: A Tumor Growth Modeling Based Approach [21]

Gooya, Biros and Davatzikos tested Expectation-Maximization algorithm for registration of brain MR images with tumors. The main goal of the work is model tumor growth and register scans to the brain atlas.

In the first step, authors used SVM classifier to divide the brain pixels on six classes: gray matter, white matter, cerebrospinal fluid (CSF), enhancing tumor, non-enhancing tumor and edema. Then they used five classes (they merged enhancing structures with non-enhancing structures to the one class) as posterior probability maps for the registration with the atlas. They used EM algorithm for registration which is based on finding he maximum likelihood parameters of statistical model. Simply, authors proposed the method which try to find model of the tumor in brain atlas which will be the most similar with the real MR scans of the patient.

Authors evaluated the method on simulated datasets and real multimodal glioma datasets. They proposed own method for the evaluation based on the posterior probability maps. The results show that method achieve good similarity between patient images and wrapped templates.

9 Proposed approach for tracking brain changes

The main goal of the work was to propose and implement automatic methods appropriate for tracking tumor changes. While the MRI is the most usual test for the brain cancer diagnostic, proposed methods work with input MRI data - examinations from one patient captured in regular time periods.

MRI brain scans are volumetric – three-dimensional data. It means that they consist of several slices. During the treatment of the brain cancer, patient has to undergo several examinations in regular time intervals. Doctors have to evaluate and control all results of the patient examinations to follow the progress of the cure. It can be really very time consuming and annoying task, so computer vision can be very helpful in this field and automate some monotonous jobs.

There are some problems which are necessary to take into account:

- When volumes are captured in different time they can be differently rotated. It is necessary to rotate volumes to the same position to track tumor and brain changes. This is task for the rigid registration.

- The tumor and also brain changes over time. Tumor can grow or diminish, which also affects the structure of the surrounding healthy parts of the brain. In order to monitor changes of the disease, it is important to know how the parts of the brain were pushed out or deformed by the tumor. It is task for the non-rigid registration. It is necessary to be careful and do not deform the tumor during the calculations. This is one of the reasons why segment the tumor. Another reason is that it allows better monitoring of the tumor changes.

- When volumes are captured with different devices or only with different settings of one device, they are not consistent. The values of the same brain parts can be different. We need to deal with it before the registration and also for the clear visualization. Thus, we have to preprocess the data.

9.1 Basic workflow

In the Figure 24 is denoted basic flow (using activity diagram) of the proposed approach for detection and tracking the tumor changes. Input for the method are data from magnetic resonance imaging test as was mentioned. That's are volumetric data. If we want to work only with slices, we need to create them from the input volumes. In the survey we use data from BRATS dataset and data from practice. Datasets are explained in chapter 9.2 Input MRI volumes.

First step of the method is preprocessing. We have to deal with different brightness and contrast of the volumes captured in different time. It is important mainly for the visualization, but it can be helpful for whole process. Preprocessing method is based on normalization, scaling and histogram matching algorithm. The process is explained in the chapter 9.3 Preprocessing.

Second step is segmentation of the tumor. We need to know boundaries of the tumor to be able to track its changes. It is also important for the next step - rigid registration. Changed parts of the tumor and surrounding brain can cause errors in the rigid registration if they had not been segmented. Tumor segmentation is very hard task for computer vision and many approaches is dedicated to this theme.

We decided to analyze state of the art methods and find opportunities to get some sources for testing. We have found three different successful and available methods based on convolutional neural networks and tried one of them. Methods for tumor segmentation and our testing are explained in the chapter 9.4 Tumor segmentation.

Next step is rigid registration. We need to rotate volumes to the same direction and scale them to the same size which is perfect task for the rigid registration. We use basic flow of the rigid registration – it is explained in the chapter 9.5 Rigid registration.

In the next step we need to do non-rigid registration (also called elastic) to be able track the changes of the health brain parts and also of the tumor. Changes caused by the tumor are irregular so it is perfect task for non-rigid registration. We decided to use optical flow algorithm to find corresponding points between the examinations as is explained in chapter 9.6 Non-rigid registration. To visualize the changes between several different examinations we decided to test image morphing.

Finally, we would like to visualize the results with the simple desktop application. Application shows slices in all directions (axial, coronal and sagittal) and is also capable to render volume in 3D. Application also provides a simple interaction to the user and can show boundaries of the tumor. Visualization tool is explained in the chapter 9.7 Visualization tool.

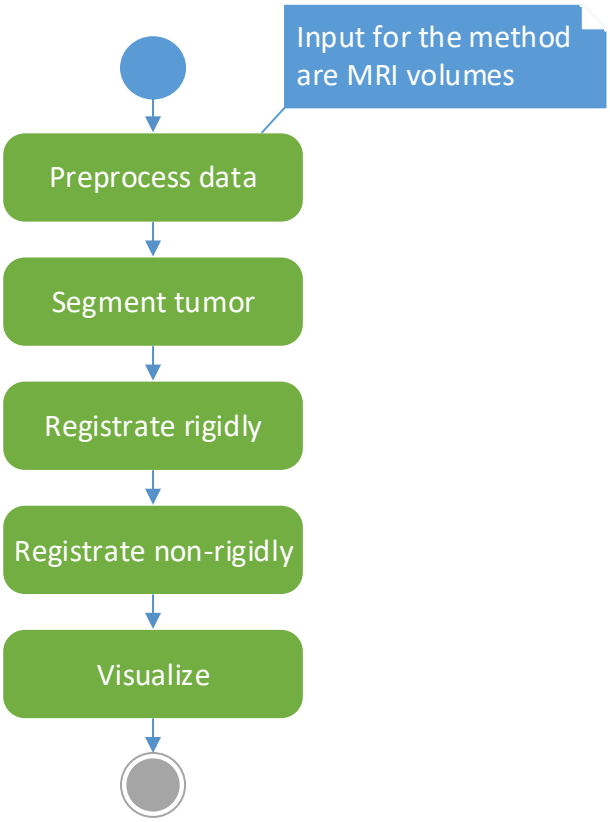


Figure 24: Basic flow of proposed method represented by activity diagram (UML notation)

9.2 Input MRI volumes

Input for the method are 3D data called volumes from magnetic resonance imaging. MRI is the most usual test for the brain cancer diagnostic and is repeated periodically during the treatment. Volumes can be converted to the slices to see them in 2D views. Doctors like to use 2D images for diagnostic, but also 3D view can be interesting for the visualization. In the Figure 25 is shown an example of MRI data transformed to 2D slices in axial perspective.

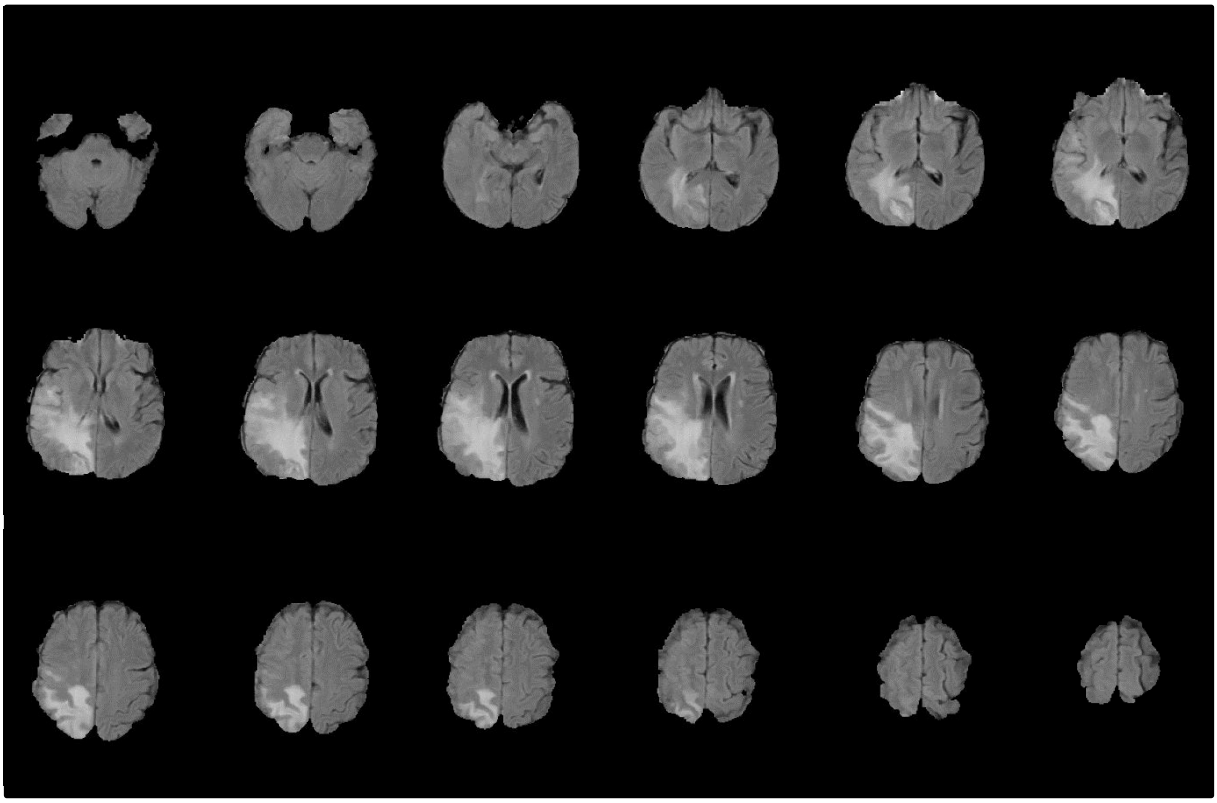


Figure 25: Slices of the MRI volume in axial view

9.2.1 BRATS dataset

In our approach we work with public MRI dataset of brains with tumors called BRATS. We are using data from BRATS 2015 [72], because later datasets do not contain testing data. Dataset BRATS 2015 consists of:

- training data
    - 220 high grade glioma studies (HGG) – tumors in the brain have well visible boundaries
    - 54 low grade glioma studies (LGG) – tumors in the brain do not have clearly visible boundaries
- testing data – 110 studies of high grade gliomas (HGG) and also low grade gliomas (LGG)

Training and testing data are preprocessed. They are rigidly registered and the craniums are removed from the volumes. The pro of the dataset is that it contains 4 different modalities: T1, T2, Flair and T1c. Different modality can highlight different parts of the tumor – necrosis, edema, non-enhancing tumor, enhancing tumor.

Training and also testing data contains masks as shown in the Figure 26. It labels background, health brain parts and tumor with 5 different values (it is explained in BRATS article from 2015 [42]):

- 1 – necrosis – visualized in the Figure 26 D with green color
- 2 – edema – visualized in the Figure 26 D with yellow color
- 3 – non-enhancing tumor – visualized in the Figure 26 D with red color
- 4 – enhancing tumor – visualized in the Figure 26 D with blue color
- 0 – everything else – health brain and black background

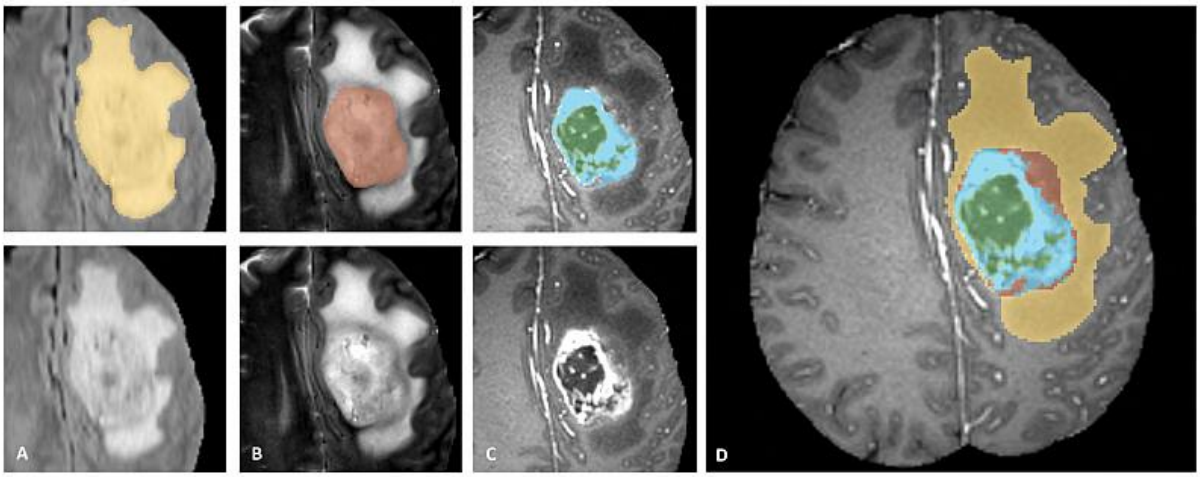


Figure 26: Labels of the brain parts: from the right A: the whole tumor (yellow) in the FLAIR modality, B: the tumor core (red) in the T2 modality, C: the enhancing tumor structures (blue) in the T1c modality and necrotic components (green) in the T1c modality, D: edema (yellow), non-enhancing solid core (red), enhancing core (blue), necrotic core (green) [42]

9.2.2 Siemens dataset

We will also test our method on dataset from practice which was provided by Siemens Healthineers company. The pro of the dataset is that it contains data which are not preprocessed – volumes contain craniums. However, examinations of the subjects contain only one modality (usually flair) for every and provided segmentations of the brain are labeled only with 0 for not tumor and 1 for tumor pixels. It consists of 108 studies – some of them have more fusions (= volumes captured in different time during the treatment). These studies are interesting for our survey.

9.3 Preprocessing

Volumes from different examinations have got different values of the same brain parts. It is because they were captured in different times and with different device settings. They can also have different contrast and brightness. Thus, we proposed the method to preprocess the data. Basic flow of the preprocessing is visualized in the Figure 27. It consists of three steps: removing the percentile, scaling the data and performing histogram matching.

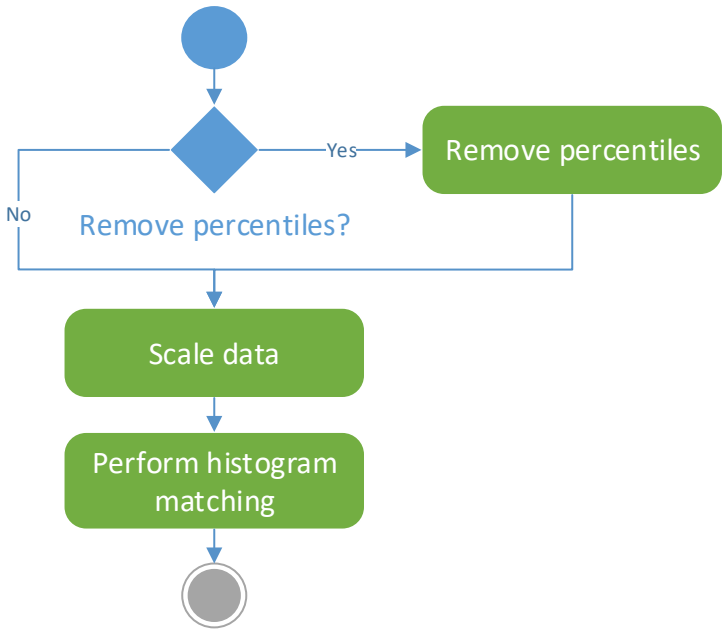


Figure 27: Flow of preprocessing represented by activity diagram (UML notation)

9.3.1 Removing the percentile

Data we were working with have a big range of the values. Some minimal and maximal outliers are not very relevant for the next processing. In addition, we have to scale the data to the same range. Ignoring the outlier values allow us reach better results. Thus we can remove data above and under some percentile. It is possible to do it by following simple conditions:

$$if\ x(i,j) < bottom\ percentile\ then\ x(i,j) = bottom\ percentile\ and$$

$$if\ x(i,j) > top\ percentile\ then\ x(i,j) = top\ percentile,$$

where $x(i,j)$ is a value of the i^{th}, j^{th} pixel and $bottom\ percentile$ is a margin for minimum outliers and $top\ percentile$ is a margin of maximum outliers.

Actually, volumes contain black background and foreground. We should also ignore black background and count percentile only from the foreground to remove relevant outliers.

In the Figure 28 we can see the histogram of the scaled volume data and in compare in the Figure 29 we can see histogram of the same volume where we removed percentile ignoring background and then scaled the data.

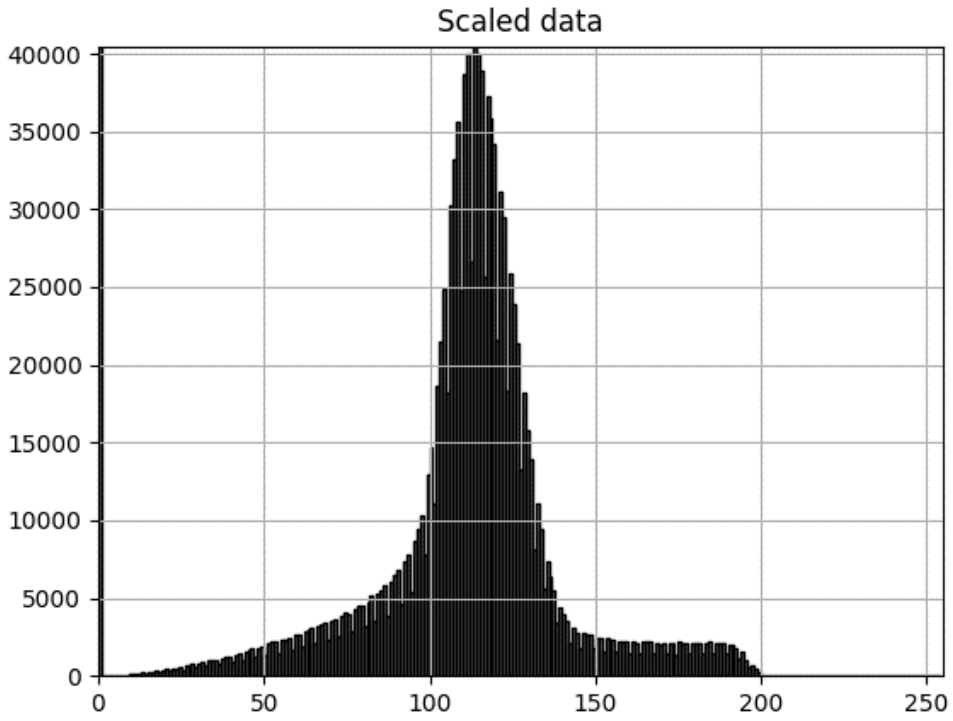


Figure 28: Histogram of scaled volumetric data

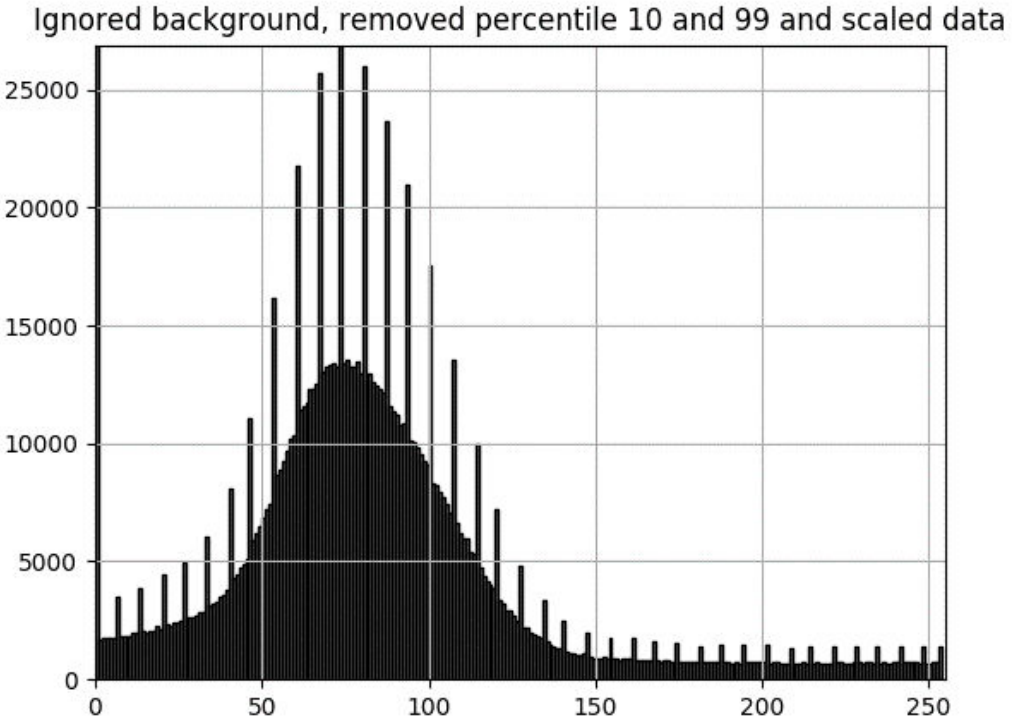


Figure 29: Histogram of volumetric data where percentile was removed ignoring background and data were scaled

9.3.2 Data scaling

We have to scale the data to the range 0 – 255, because in the next steps we use some methods which can work only with 8-bit images. We use min max normalization formula to down scale the data:

$$image = \frac{\text{image} - \min(image)}{\max(image) - \min(image)}$$, where $image$ is n-dimensional array (2D or 3D).

In the Figure 30 we can see difference between the histograms of the original and scaled volume data. In the Figure 30 a) we can see original input volumetric data histogram and in the Figure 30 b) Is histogram of the same data but scaled to the range 0 – 255.

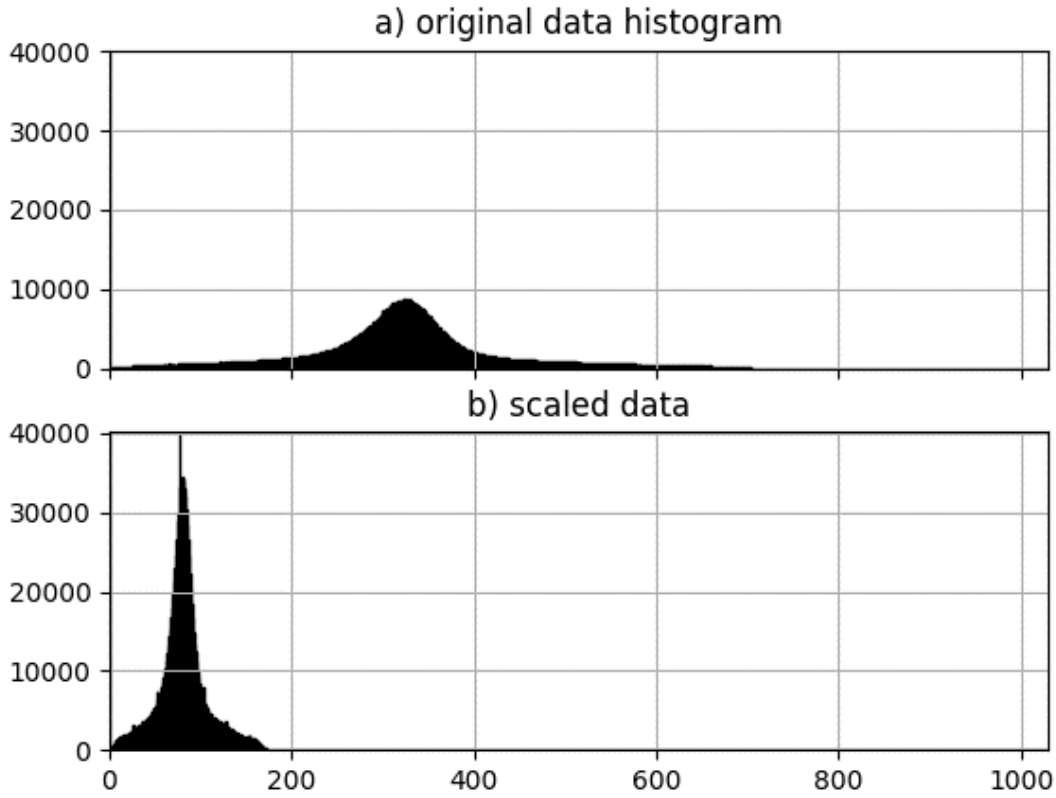


Figure 30: Comparison of original and scaled data visualized in histogram

9.3.3 Histogram matching

Histogram matching is a simple algorithm which can match the histogram of one input data called reference to the histogram of another input data called target. In the first step we compute histogram of target and also histogram of reference. In follow, we compute cumulative distribution functions for both, reference $CDF_1()$ and target data $CDF_2()$. It gives us a probability that variable will take a given value or less. Finally, we compute a function $M()$ which can match reference data to target data using the cumulative distribution functions – we find the gray level G_2 of reference data where cumulative distribution function of that gray level equals to cumulative distribution function of the gray level

G_1 from target image: $CDF_1(G_1) = CDF_2(G_2)$. Then we apply a function $M()$ on the each pixel of the reference image.

In the Figure 31 we can see the effect of the histogram matching algorithm. In the Figure 31 a) is histogram of reference volume, in the Figure 31 b) is histogram of target volume and finally in the Figure 31 c) is histogram of matched reference volume.

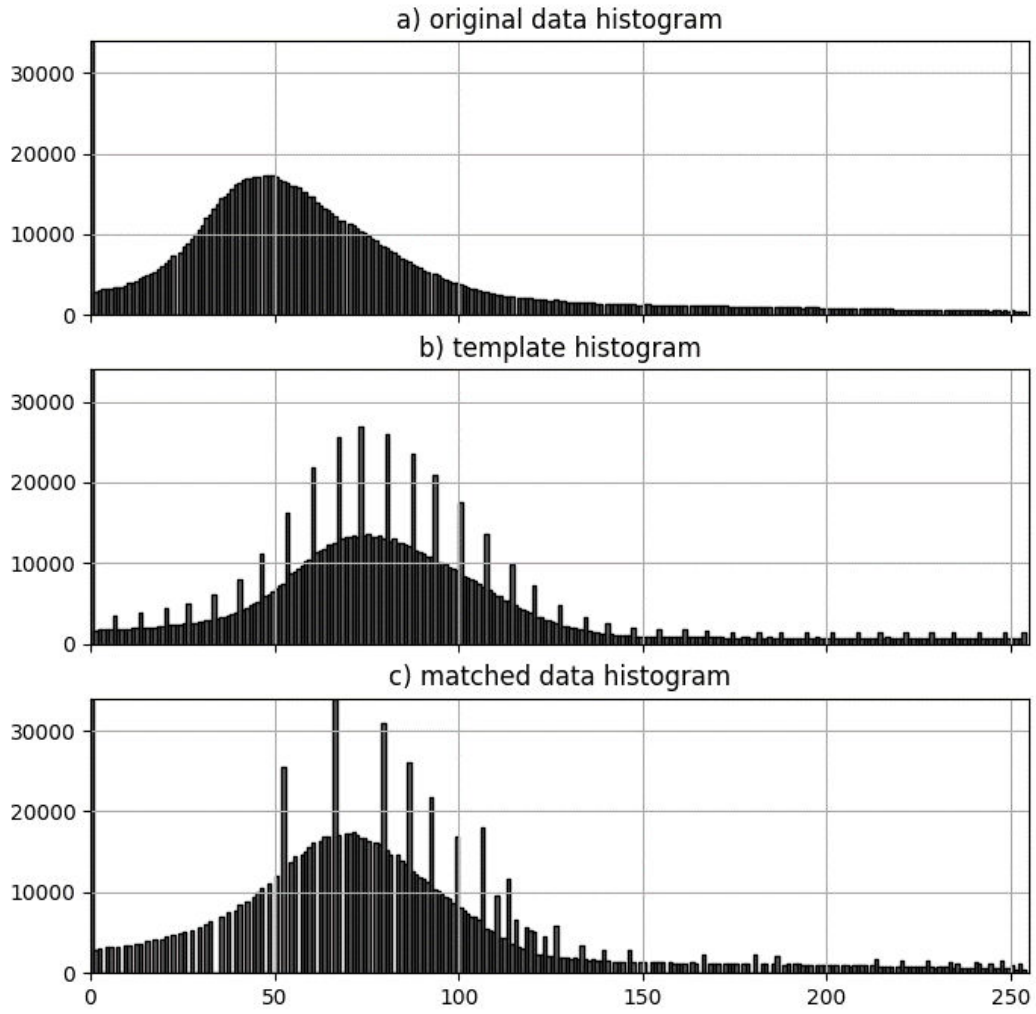


Figure 31: Effect of the histogram matching algorithm

9.3.4 Testing and results

We tested some configurations of the preprocessing method on the BRATS dataset (mentioned in the chapter 9.2.1). In this section, we present some results of different configurations and the impact on the data. When we work with 3D data, we choosed one slice from pat_153 – examination 0109, modality flair, axial slice number 78 – to present the results.

At the beginning we tried only to scale the data, because original data have bigger range and are stored as 16-bit volumes. However, in the next process we use some algorithms which can process only 8-bit images (gray level range between 0 and 255). In the Figure 32 a) we can see the histogram of original data and in the Figure 32 b) is histogram of scaled data. Actually scaling causes loss of some data, but

we can minimalize the damage. It is because gray level of the original data is in range 0 – 600 or more and we have to shrink gray level values to the range 0 – 255. Actually this is not possible to see very well in the volumes because they have got small resolution. Slice 78 of the scaled volume is shown in the Figure 37 a).

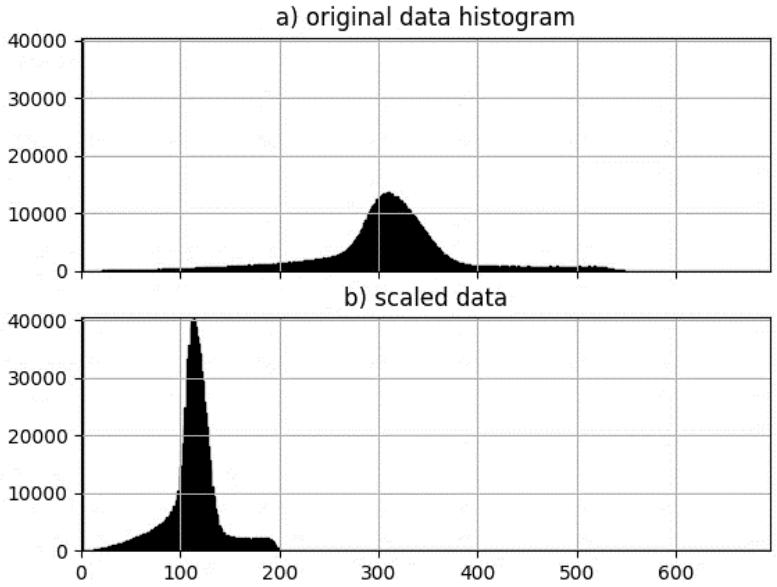


Figure 32: Histograms of original and scaled data

In the Figure 33 we can see a big column on the value 0 – it is caused by the black background. And then it looks that histogram starts later than on the value 1 and ends at the value approximately 200 not at 255. It is because the values between these ranges do not have such high representation in the volume. Thus, we can remove outliers – small values with small representation in the volume and high values with small representation in the volume thanks computing some percentile of the data.

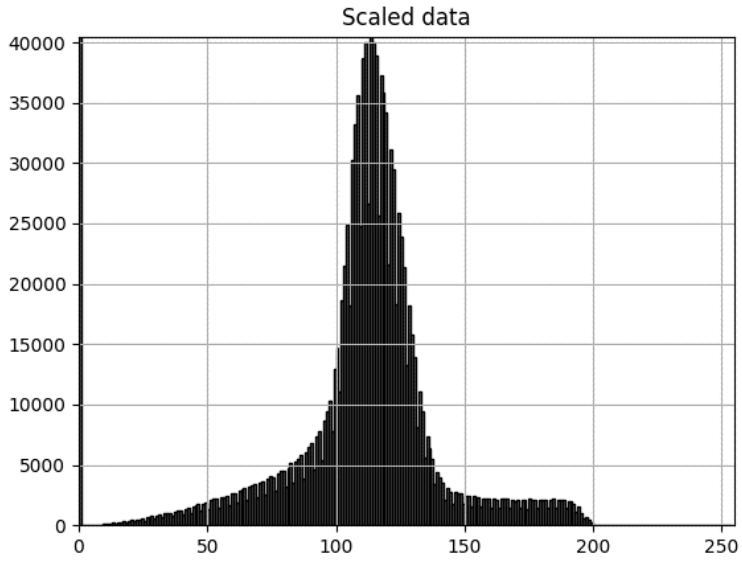


Figure 33: Histogram of scaled data

In the Figure 34 we can see the histogram of the volume where we removed values lower than bottom percentile 5 of the data and values higher than top percentile 99. This change is possible to see very well also in the volume and is shown in the Figure 37 b) using same slice 79 from the same volume.

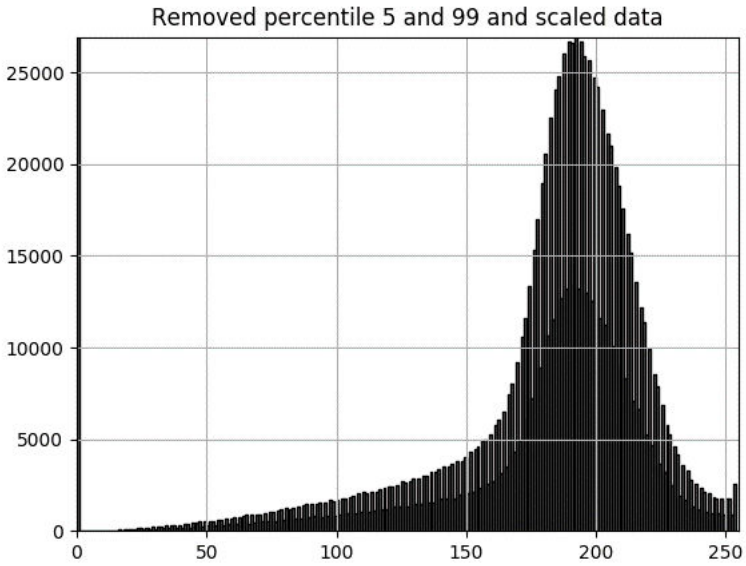


Figure 34: Histogram where data were scaled after removing top and bottom percentile

Actually as we can see on the histogram, it still looks that histogram starts later than on the value 1. Rather than enlarge the bottom percentile we decided to compute bottom percentile 5 ignoring 0 values = black background. We can see results of that operation presented by histogram in the Figure 35. Also volume looks different and is shown using the same slice 78 in the Figure 37 c).

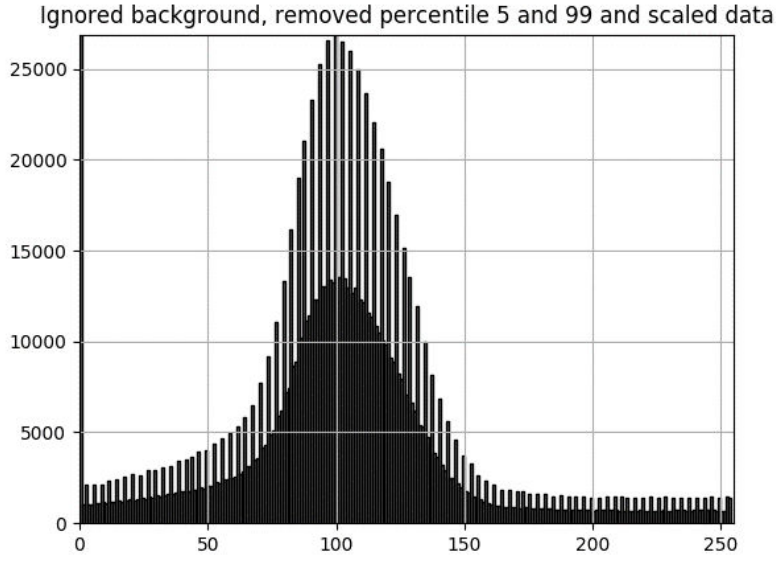


Figure 35: Histogram where data were scaled after removing top and bottom percentile ignoring background

Finally, we tried to compute higher bottom percentile – percentile 10 and remove values lower than the percentile. However, it looks on the histogram presented in the Figure 36 that it removes too much

data, so we decided to use percentile 5 for the preprocessing. Also volume presented using the slice 78 looks too dark as is visible in the Figure 37 d).

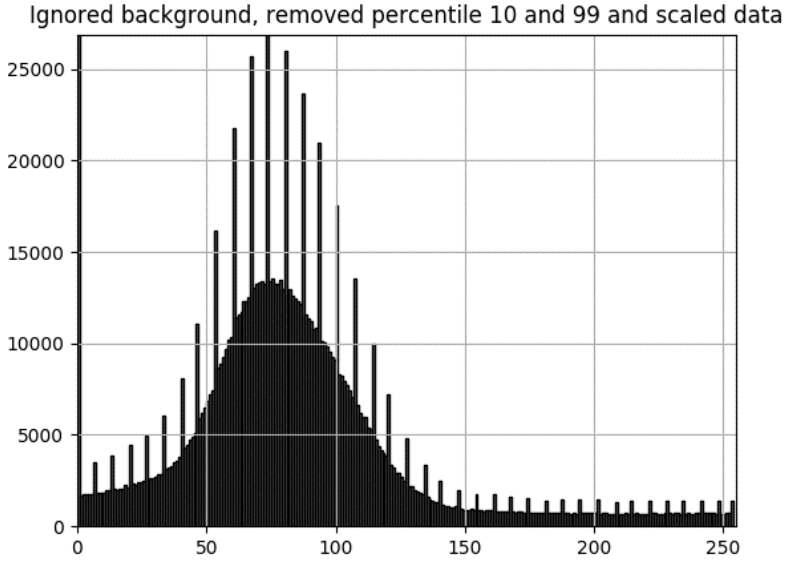


Figure 36: Histogram where data were scaled after removing top and bottom percentile ignoring background

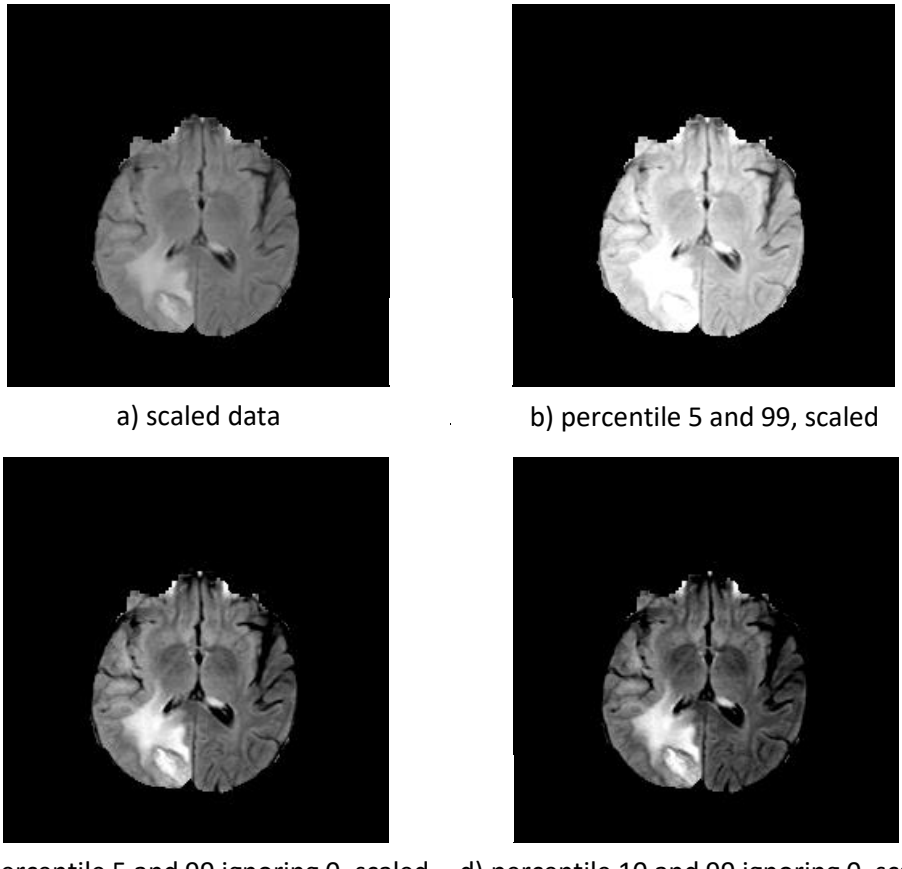


Figure 37: How removing percentile changes data

We scaled all examination of the patients using the same process, but we still did not solve a problem with different brightness of the examinations captured in different times. Thus, we decided to use histogram matching algorithm to align histograms of the images as is presented in previous chapter.

The results of our preprocessing method are presented in the Figure 38. In the top row we can see slice 78 of the all original volumes from pat_153, flair modality. In the bottom row are the same slices from the same examinations, but after all operations – remove top and bottom percentile of data, scaling using min max normalization and histogram matching. First image was used as target image and all next images were in histogram matching used as reference images – one by one. We can see that our preprocessing method helps to balance brightness of the volumes.

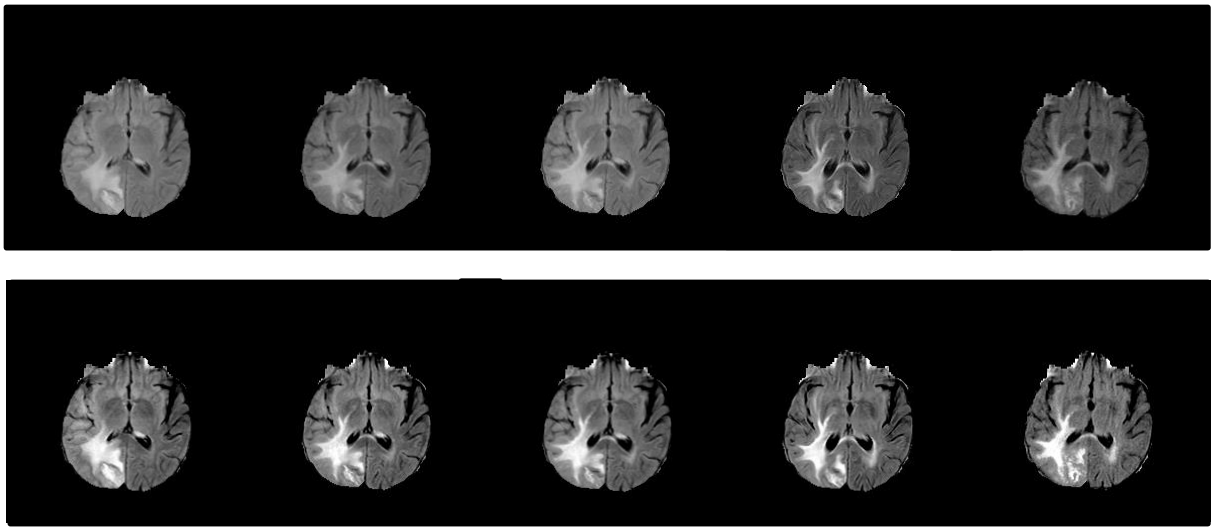


Figure 38: Original (top row) and preprocessed (bottom row) data comparison

9.3.5 Implementation

Preprocessing can work directly with 3D volumes during the processing and all operations are performed directly on three dimensional volumetric data not two dimensional slices – slice by slice. This is possible using NumPy library. For the visualization of the histograms we have also used Matplotlib library.

9.3.5.1 Structure of the source code presented by a class diagram

Preprocessing class is organized in *logic* package and used by the *main* function as is shown in the Figure 39. *Preprocessing class* has two important parameters:

- *top_percentile* – value of top percentile: values, higher than *top_percentile*, are replaced with its value
- *bottom_percentile* – value of bottom percentile: values, higher than *top_percentile*, are replaced with its value

And also implements several methods:

- *__init__()* – method initializes preprocessing and preprocess target data

- *process_data()* – method preprocess data as reference data
- *remove_precentile()* – replace values higher and lower than top or bottom percentile by their values
- *normalize()* – method scale the data using min max normalization
- *hist_match()* – performs histogram matching of target (in code template) and reference image
- *get_template()* – returns preprocessed template

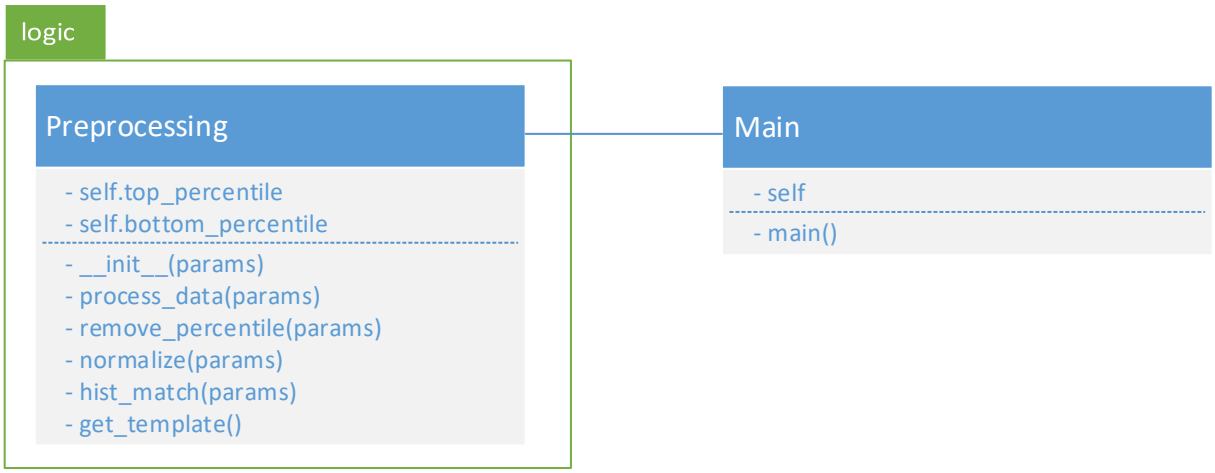


Figure 39: Class diagram of preprocessing (UML notation)

9.3.5.2 Interaction of the objects presented by a sequence diagram

In the Figure 40 is shown sequence diagram of the preprocessing. *Preprocessing* is used by the *main*. First we have to initialize preprocessing – we send target data to the preprocessing and preprocess them replacing values higher or lower than top or bottom percentile by its value and scale them using min max normalization. Then we get the preprocessed template to use it later in process.

In the next step we process all additional examination in loop – we send all of them one by one to the preprocessing and perform the same operations – remove the percentile (replace values lower or higher than top or bottom percentile by the values of these percentiles) and scaling (using min max normalization) and finally we compute histogram matching between target (template) data and reference data.

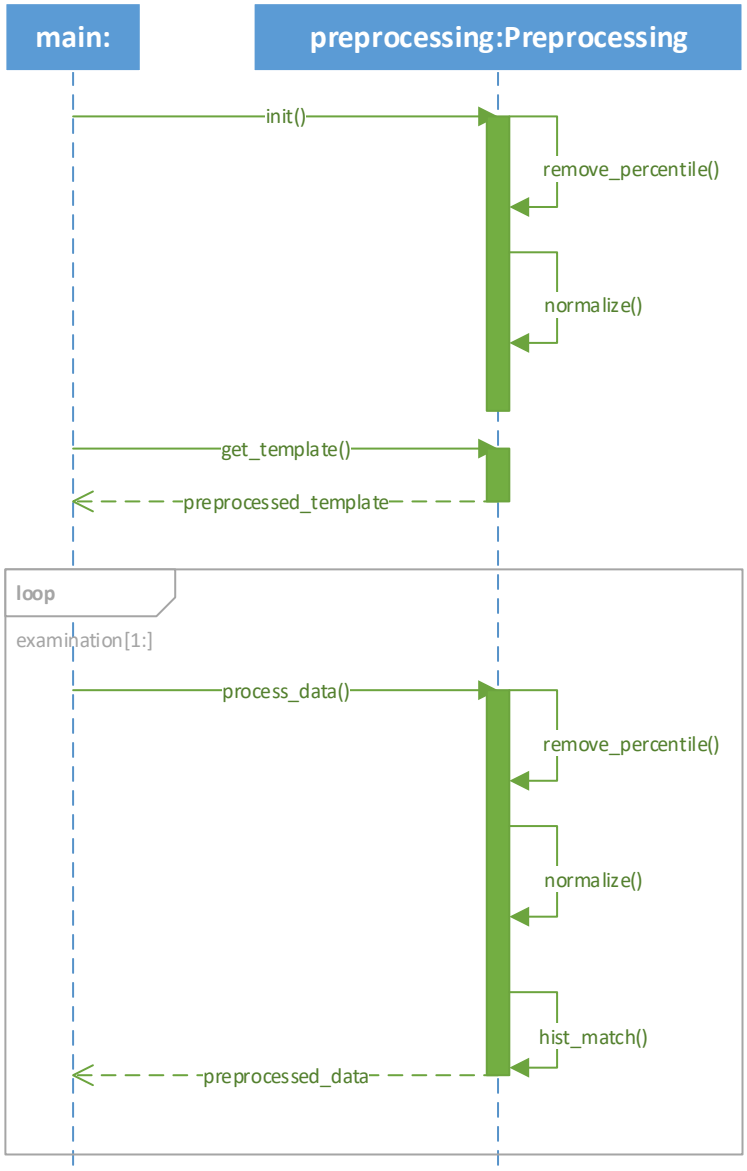


Figure 40: Sequence diagram of preprocessing (UML notation)

9.3.6 Discussion

Preprocessing was necessary to balance brightness of the volumes from different examinations – means that they were captured in different time. In our method we change values lower than percentile 5 and higher then percentile 99 to the values of these percentiles. We also decided to compute percentile only from foreground – it means that we are ignoring black background. We tried also different values of the bottom percentile, but percentile 5 looks the best for our data.

Then we scale the data to range 0 – 255 using min max normalization. This is necessary, because some methods from the libraries used in the next steps can process only 8-bit images. We minimize the damage of the important data thanks the previous percentile operation.

Finally, we performed histogram matching – volume from the first examination was used as target data and all next examination were reference data. This method helped to balance the brightness of the

volumes and they look almost the same. This is helpful mainly for the visualization, but can help also other methods to work better.

9.4 Tumor segmentation

Tumor segmentation using computer vision is a problem which is studied a lot by professional community. Solution of this problem is based on a big motivation – help doctor, radiologist with some manual and monotonous work which they have to do. It is important to segment tumors from the examinations. Boundaries of the tumor and other information – information about health and important brain centers, are used for radiotherapy planning. Thus, radiologist have to segment tumor. While MRI are volumetric data, radiologic have to segment tumor on all slices, if they have to do it manually. It is really very time consuming task. Thus, finding appropriate automatic method is a big motivation to help doctor and radiologist with their work.

Actually segment tumors from MRI brain scans is hard task, because the boundaries are not always clear and sometime tumors are not significant. Many people tried to solve this problem also we had try to implement own method in bachelor thesis. There is a lot of approaches in the field of tumor brain segmentation with very good results. We have list some of the best in the chapter 8.1 Segmentation.

While our approach has a wide range, we decided to re-implement one of the state of the art methods which reached the best results on the BRATS 2013 dataset and is explained in the BRAINLES 2014 proceedings [43]. We summary important parts of the article in our work in chapter. However, we came out of the newer article dedicated to this method from 2017 by Havaei et al. [24] – in our work we analyze this method in the chapter 8.1.2 Brain tumor segmentation with Deep Neural Networks.

One of the reasons why we decided to choose this approach was that author reached very good results presented in the articles and second one was that the source code is available online [81].

In segmentation step we proposed a simple pipeline which is visualized using activity diagram in the Figure 41. It consists only of two steps: preprocessing and tumor segmentation using model trained on the CNN proposed by Havaei et al.

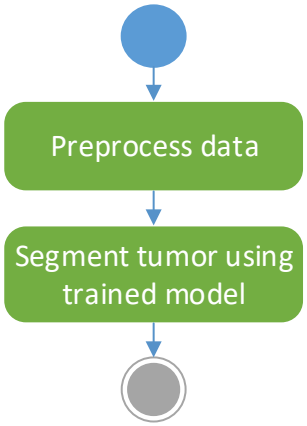


Figure 41: Flow of the segmentation step visualized by activity diagram (UML notation)

9.4.1 Preprocessing

First step of the method is preprocessing. It is necessary to preprocess the data by the same methods such as authors. Thus, we used the same preprocessing steps. At first, we removed 1% of the highest and lowest intensities (from original data, not from data preprocessed in the previous chapter – to preserve the consistency of the data). Then, we have applied N4ITK bias correction to the modality T1 and modality T1c using ANTs library [71]. Then we have normalized the data within each input channel. Finally, it is necessary to create patches before the segmentation. Patches represent data by the way they can be processed by the model.

9.4.2 Segmentation using trained model

Main step of the proposed workflow is segmentation. In our proposal, we have planned to use trained model for the tumor segmentation. However, authors did not provide these model, thus we had to train own model using their architecture explained in the chapter 8.1.2 Brain tumor segmentation with Deep Neural Networks.

9.4.3 Implementation and discussion

As was mentioned before, source code of the proposed CNN is available free online on github [81]. The problem was that it is implemented in the old technologies – using old libraries. Thus, we re-implement some parts of the sources and adapt them to the newest version of the libraries, because authors did not mention anywhere which versions of the libraries they were working with.

We tried to train own model using the same dataset such as authors of the method – dataset BRATS 2015. However, due to the available calculation power, we had to minimize all parameters which pulling out the RAM and processor.

We have to review, that we were not able to reach satisfactory results while we tried to trained own model. The main problem was, that we tried to adapt complicated CNN architecture for the new libraries and we had probably violated important parts of the architecture and were not able to find our misconduct.

Finally, we decided to use only available segmentation masks of the tumors to continue in our pipeline. Both, BRATS 2015 dataset and Siemens dataset contains mask of the tumor for patient examination which we work with in the next steps. Properties of datasets are mentioned in the chapter 9.2.1 BRATS dataset and chapter 9.2.2 Siemens dataset.

9.5 Rigid registration

The next step of our pipeline is rigid registration. When the volumes of the brain are captured, they can be rotated, translated and scaled a little differently because patients are examined in different time in some periods. If we want to track changes of the tumor and brain, we have to rotate, scale and translate volumes to the same position. This is perfect task for the rigid registration.

Rigid registration consists of six steps. Basic flow of the proposed method is shown in the Figure 42. Input for the method are two 2D slices from MRI volumes which we are going to register. In the first step we create a registration mask for both input images. The masks are used in the next step to detect

key points. Then for every key point descriptors (feature vectors) are computed. Now we have feature vectors for both input images and in the next step we find matches between them. In this step besides right matches, we find also wrong, too. Thus, we filter them using simple condition. From the correct matches we can compute matrix for the transformation and use this matrix to transform and register one of the input images to another.

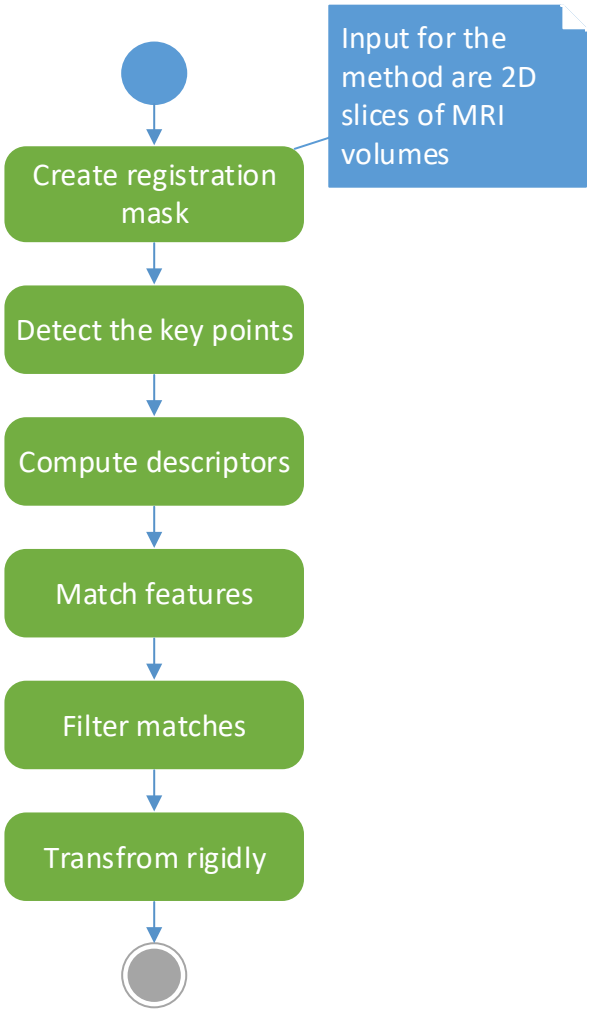


Figure 42: Flow of rigid registration represented by activity diagram

9.5.1 Input

Input for the proposed method are 2 two-dimensional images – one reference image and one input image which will be registered to reference image. These images are created from MRI volumes in axial direction. In the following chapters we present the steps of the method on the slices crated from volumes from the Siemens dataset - subj_1 examination 20090306 (as reference) and 20091029 (as input). Example of the input is shown in the Figure 43.

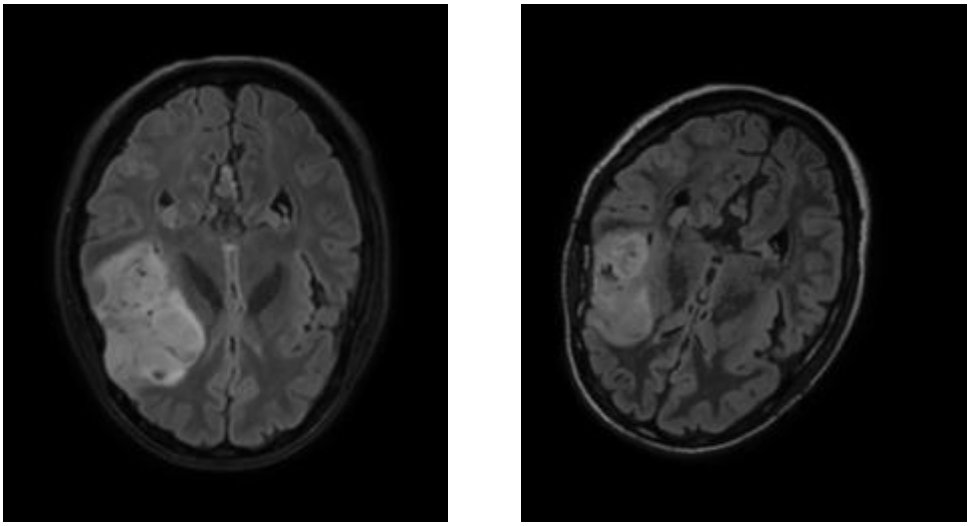


Figure 43: axial MR slice 71 from Siemens, subj_1,
LEFT (reference): examination 20090306, RIGHT (input for transformation): examination 20091029

9.5.2 Creating the registration mask

In the first step we need to create mask for the registration. There are two possible approaches:

- mask for the health brain or
- mask for the cranium.

9.5.2.1 Mask for the health brain

In the next steps we do not want to calculate with background, because it is not important. We also want to remove tumor parts from the mask, because tumor parts change a lot between two MRI tests.

Creation of the registration mask is demonstrated in the Figure 44 on the reference image from previous Figure 43. It is based on the subtraction. First we create mask of the brain. Then the mask is eroded with small kernel to get rid of very small black regions and dilated back to create mask of the brain. In our survey we create mask smaller than whole brain, because we want test results of the registration method only on the health brain parts, so we exclude the cranium. Finally, from the brain mask is subtracted mask of the tumor (as was mentioned in chapter 9.4.3 we work with the masks of the segmentations from the datasets) and the result is a mask for the registration.

Figure 44: Visual view on the process of the registration mask creation

9.5.2.2 Mask for the cranium

When the background is not important, in this step we also compute only with brain parts – concretely only with cranium of the brain. In our survey, we test difference between two mentioned approaches, thus we create mask of the cranium in this step.

Creation of the registration mask is demonstrated in the Figure 45 on the reference image from previous Figure 43. It is based on simple threshold algorithm and finding contours algorithm. At first we threshold the image to get mask of the whole brain. Then we compute outer boundary of the brain and the contour is used to create close mask of the cranium by enlarging this contour.

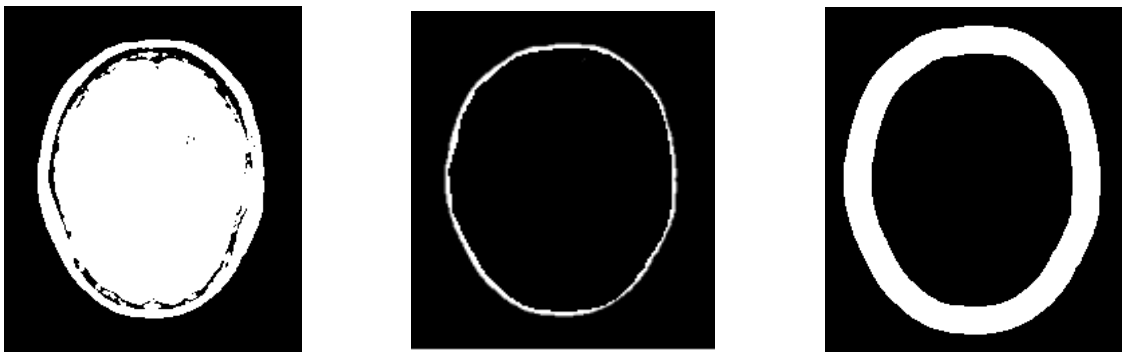


Figure 45: Visual view on the process of the registration mask creation

9.5.3 Key point detection

In the next step key points are detected using four different methods – FAST, SIFT, SURF and ORB. Results of key point detection are shown in the Figure 46 on the reference image from Figure 43. Key point detection is based on differences between the pixels. It depends on the concrete algorithm. We have used implementation from OpenCV library. Algorithms were explained in the online OpenCV documentation and explanations were referenced to official articles:

- SIFT (Scale-Invariant Feature Transform) [78] – In 2004 David Lowe came up with new feature detection algorithm SIFT [38]. The algorithm uses Difference of Gaussians. Difference of Gaussians is computed for the image as difference of Gaussian blurring of the image with two various σ. Then the local extremes are detected and marked as possible key points.

- SURF (Speeded-Up Robust Features) [76] – After the SIFT algorithm SURF was invented in 2006 as faster alternative for feature detection by three people H. Bay, T. Tuytelaars and L. Van Gool [4]. Algorithm is based on approximation of Hessian blob detector and also uses integral of the input image to make it faster. Hessian matrix (second derivation of the image) is used to find the local changes. The biggest changes are marked as the key points.

- ORB (Oriented FAST and Rotated BRIEF) [79] – In 2011 Ethan Rublee, Vincent Raubaud, Gary Bradski and Kurt Konolige invented ORB [52] as alternative for SIFT or SURF algorithm. Actually, SURF and SIFT are patented, and for commercial use you have to pay. ORB is not. It is a connection of FAST and BRIEF key point detector – first key points are found using FAST algorithm, but ORB return only top of them using Harris corner measure.

- FAST (Features from Accelerated Segment Test) [74] – FAST was invented by Edward Rosten and Tom Drummond. The algorithm is fast and allow real-time detection. The process is described in detail in article from the year 2006 [51]. Input for the algorithm is a contrast threshold constant T. Basically, for every pixel P of the image algorithm decides if the pixel is key point. A circle of 16 pixels is created around every pixel P and if n pixels are higher or lower then then intensity of P pixel + threshold constant T, then pixel P is the key point.

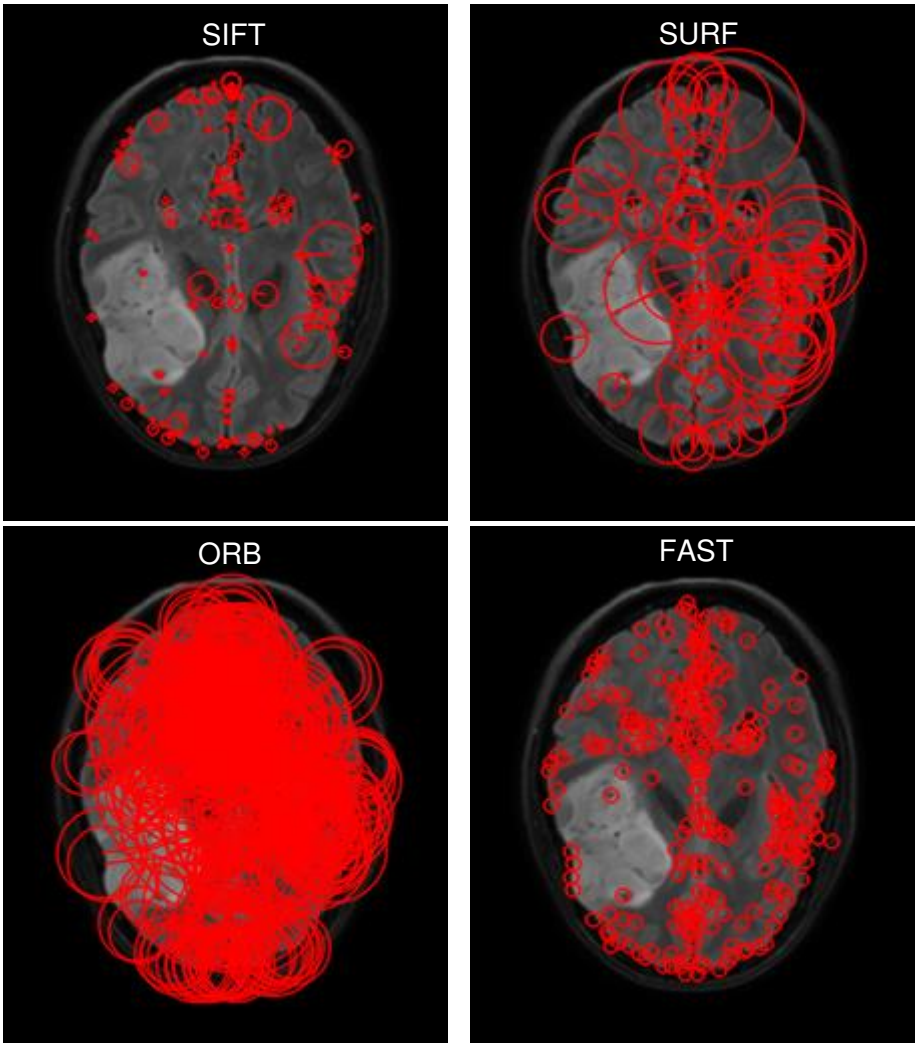


Figure 46: Visualization of detected key points by different detectors (FAST, SIFT, SURF, ORB)

9.5.4 Calculating descriptors – the feature vectors

Descriptors are computed in the next step. Result of the computation are vectors of the features which contains additional information. Type of the information depends on the type of the algorithm for descriptor computation. SIFT, SURF and ORB have got their own algorithms for feature detection. We have used implementation from OpenCV library, were algorithms were explained:

- SIFT [78][38] – Descriptors are computed for every key point. SIFT algorithm takes 16x16 neighborhood of every key point and divide it on 16 blocks with sizes 4x4. An 8-bin histogram is created for every block. Thus, the resulting feature vector consists of the 16*8 = 128 computed values.

- SURF [76][4] – SURF descriptor algorithm uses Haar wavelet in horizontal and vertical direction. Feature vectors are computed as the sum of Haar wavelet response around the key points.

- ORB [79] [52] – As was mentioned, ORB fuses FAST and BRIEF, and BRIEF algorithm improved by the capability to compute with orientation is used to compute the feature vectors of the key points.

9.5.5 Matching feature vectors

In our approach we have combined SIFT key point detector with SIFT descriptor algorithm, SURF key point detector with SURF descriptor algorithm and ORB key point detector with ORB descriptor algorithm. However, FAST does not have own algorithm for descriptor calculation, so we decided to combine FAST key point detector with SIFT and SURF descriptor calculation algorithm and compare results.

Depending on the similarities, which are evaluated from the feature vectors, in this step key points are matched using brute force matching. In the Figure 47 are shown results of the matching between the image from the Figure 43. Count of the matches is based on the count of the key point detected with different key point detectors. Matched key points depend on the type of descriptor calculation – how the feature vectors were computed.

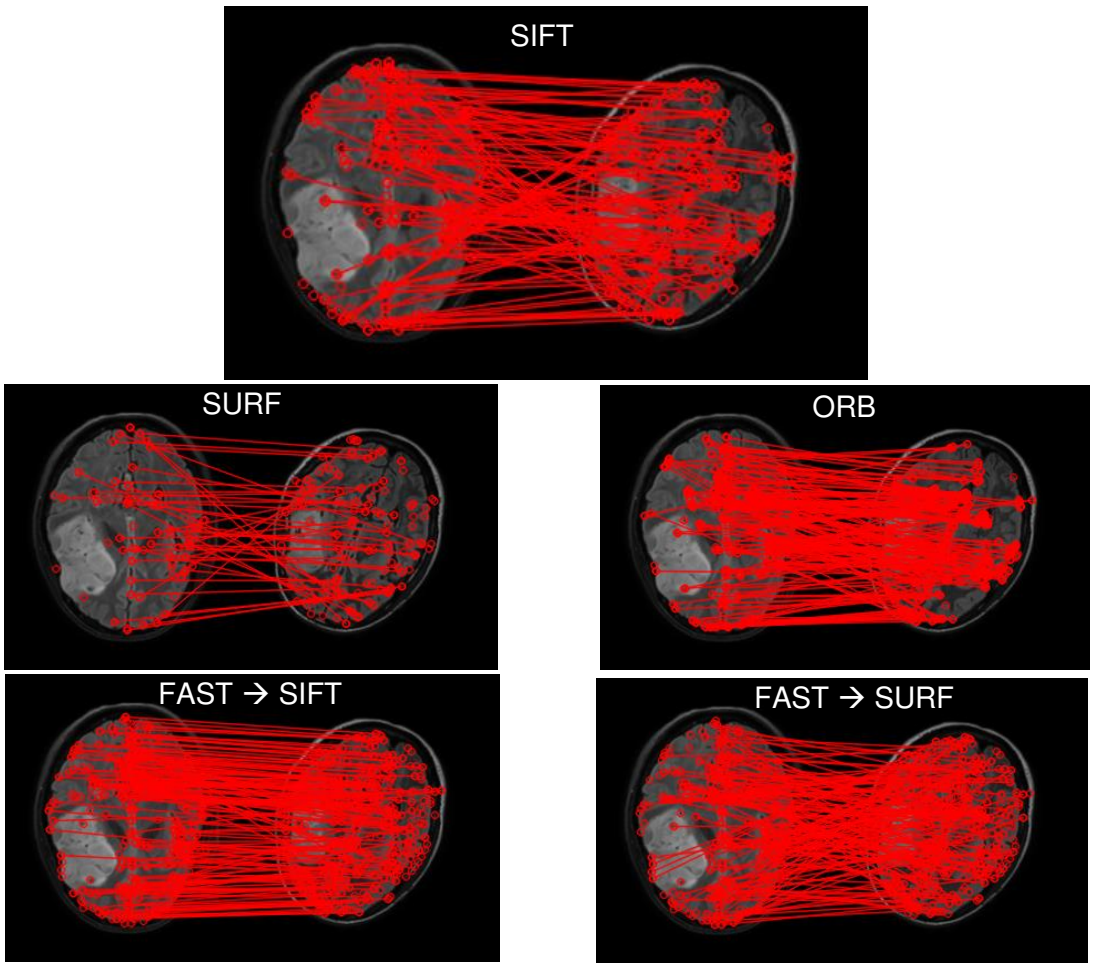


Figure 47: Result of the key point matching using different feature extractors

9.5.6 Filtering the matches

A lot of matches is found during the matching process. In addition to the good matches, there is a lot of fails. In this step, we filter the matches. The main prejudice of the filtering step is that first input image is not rotated a lot to second one input image, so lines between key points should be almost evenly. It is based on the supposition that key points have got very similar coordinates. Results of the filtration step are shown in the Figure 48.

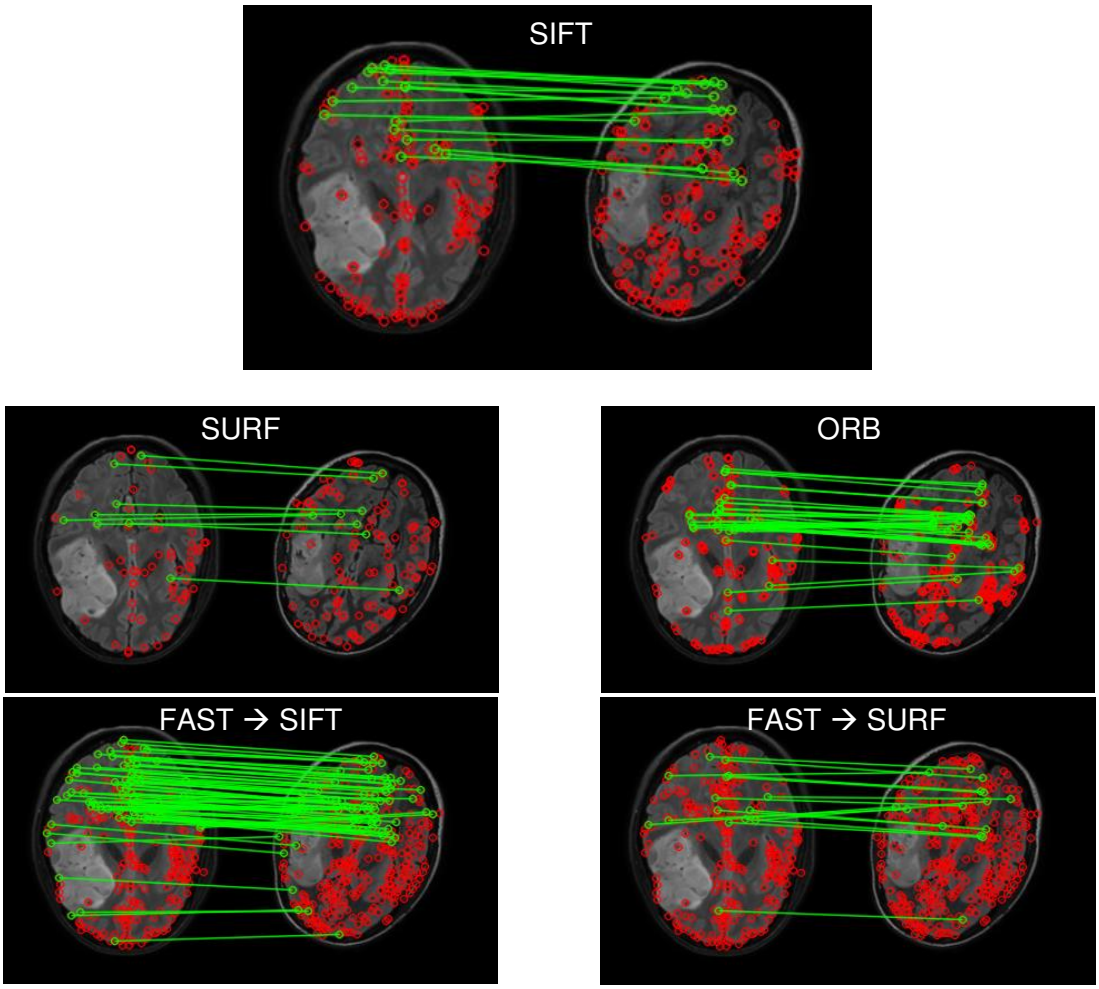


Figure 48: Result of match filtration step

9.5.7 Rigid transformation

Final step is rigid transformation. The affine matrix for the rigid transformation is found using matches from the previous step. Then the matrix is used to warp a perspective of the first input image. In the Figure 49 on the first image is visualized blue channel of input slice transposed over red and green channel of the reference slice from the Figure 43 and next images visualize results of the registration by all used combinations of the algorithms where the same red and green channels are transposed over the blue channel of registered – warped slice. As we can see In the Figure 49, some parts are blue – came from blue channel of input image and some parts are yellow – it is caused by red and green channel of reference slice. Actually, in the dataset are also available correct registrations of the input slice, so we applied the same visualization on correctly registered input (correct original) and rotated, warped and translated input slice which we registered. Results are shown in the Figure 50.

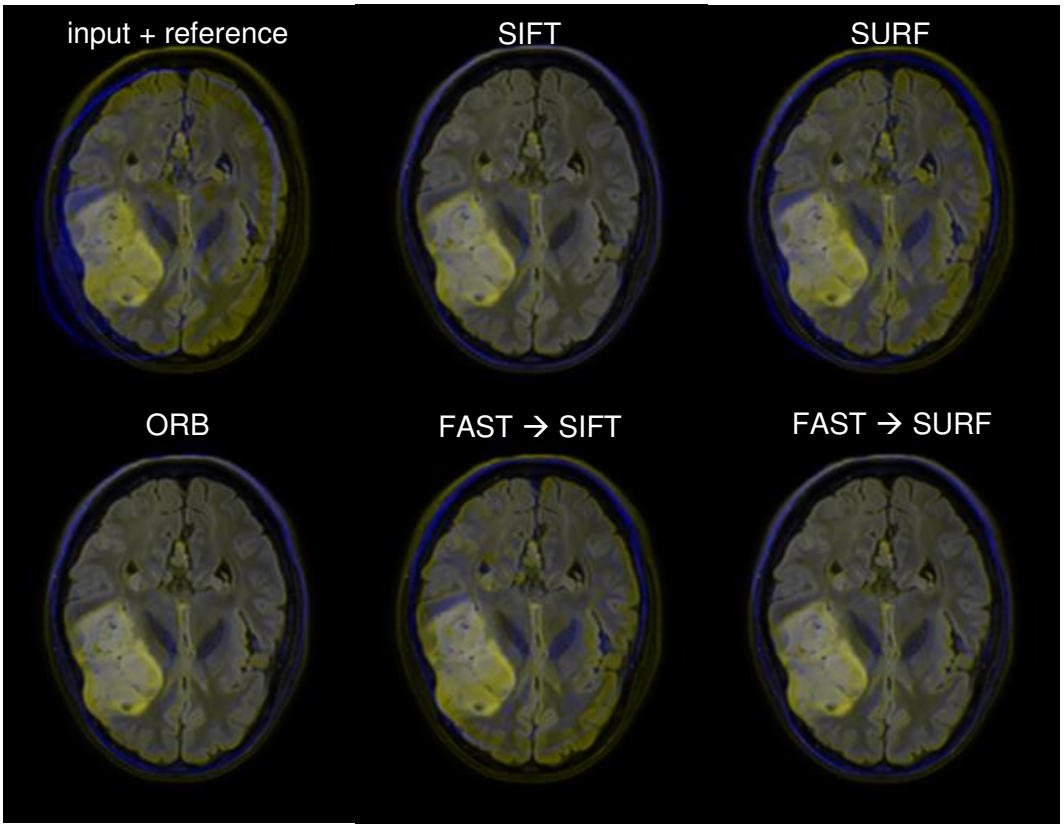


Figure 49: Comparison of the input (blue channel) transposed over the reference (red and green channel) slice before the registration and after

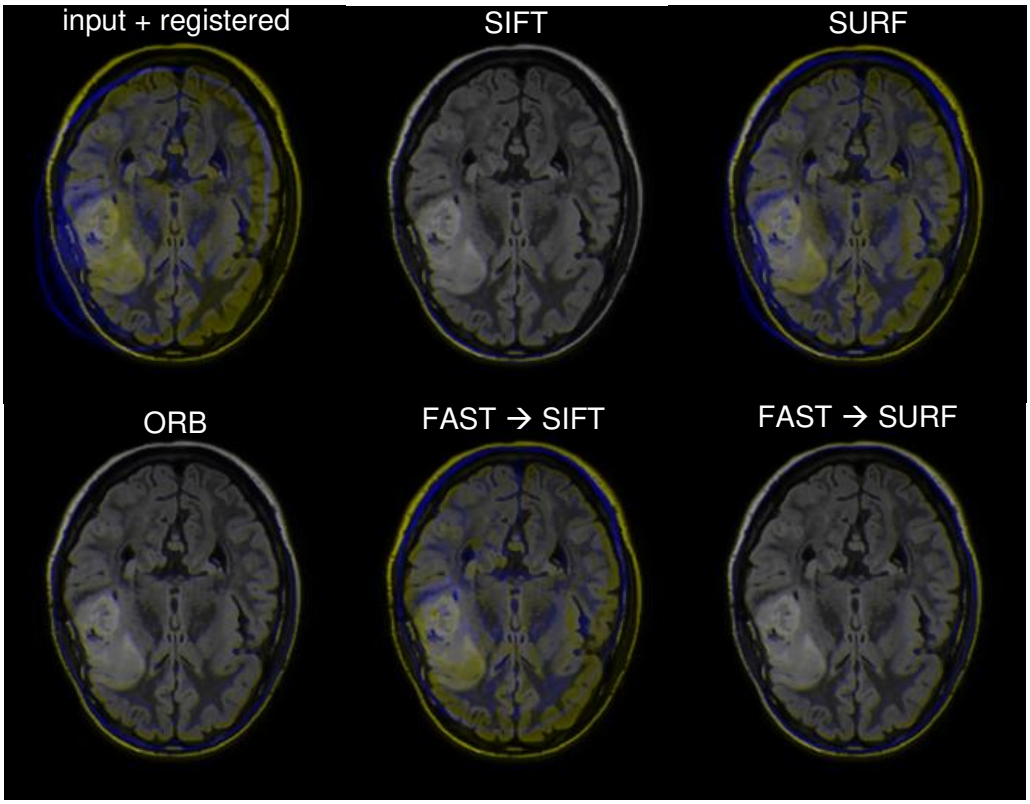


Figure 50: Comparison of the input (blue channel) transposed over the correct original input (red and green channel) slice before the registration and after

9.5.8 Testing

We tested the method with different combinations of algorithms and compared the results. We tried different registration masks (mask of the health brain and mask of the cranium) and also different combinations of key point detectors and descriptors. We tested combinations showed in the Table 1.

	registration mask	key point detector	descriptor
	health brain	SIFT	SIFT
	health brain	SURF	SURF
	health brain	ORB	ORB
	health brain	FAST	SIFT
	health brain	FAST	SURF
	cranium	SIFT	SIFT
	cranium	SURF	SURF
	cranium	ORB	ORB

Table 1: Tested combinations of algorithm

9.5.8.1 Dataset

While proposed method works only with 2D images, but our testing dataset – Siemens dataset consists of 3D volumes as is explained in the chapter 9.2.2, we decided to create own dataset. We generated 2D slices from 3D volumes. In the Siemens dataset examinations are registered, so we left 2D slices in original - correct original and we created new files which contains these slices, but in some different rotation, brain can be alse scaled or translated. We decided to choose slices of the first examination as a reference slices and all next examination as unique inputs. Reference slice is not scaled, rotated or translated. Finally, our dataset contains:

- 23372 2D unique inputs and also the same count of the correct original,
- 10011 2D reference images,
- tumor masks for all slices.

9.5.8.2 Testing flow

Testing flow is visualized using sequence diagram in the Figure 51. We process all images of the dataset in the loop. At first we preprocess the data – all input slices are preprocessed (by the way explained in the chapter 9.3) to the corresponding reference slice. Then inputs are registered to their corresponding reference slices – slice by slice and finally, we evaluate the registration by comparing histograms of the slices (explained in detail in the chapter 9.5.8.3)

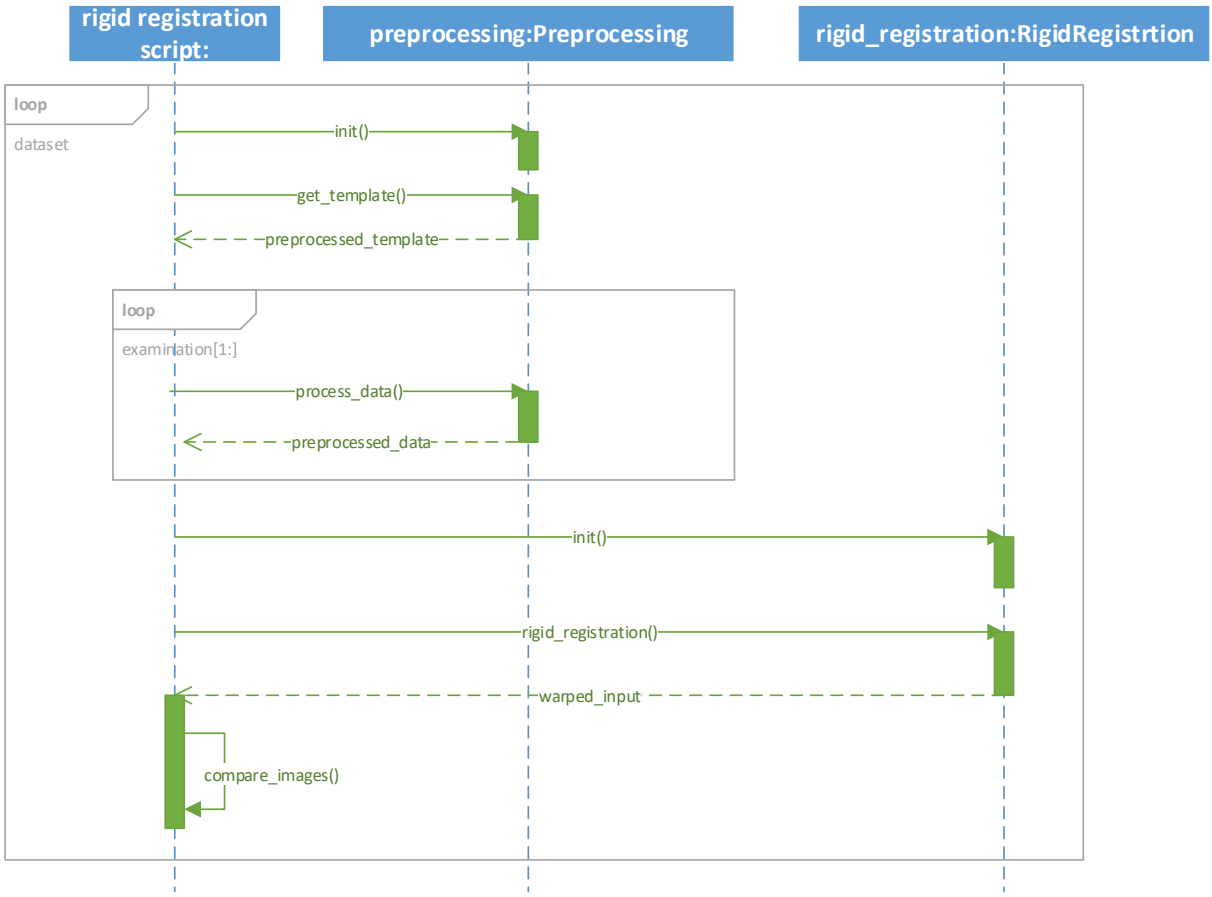


Figure 51: Flow of the testing process (UML notation)

9.5.8.3 Evaluation

For the evaluation of the registration we decided to compare slices:

- input with reference,
- warped input with reference,
- input with correct original and
- warped input with correct original.

Thanks these combinations we can see, if registration was correct.

Actually, we do not compare whole slices at once and we do not compare black background of the images (it is bloody). However, we span through the slice with the small rectangle as is shown in the Figure 52 and if we hit the foreground, we compute the histogram of the slices parts under the rectangle as is shown in the Figure 53. Then we compare the histograms using three metrics:

- correlation,
- intersection and

- Bhattacharyya distance.

Finally, we compute the average value of the metrics and this is the result of the comparison of two slices.

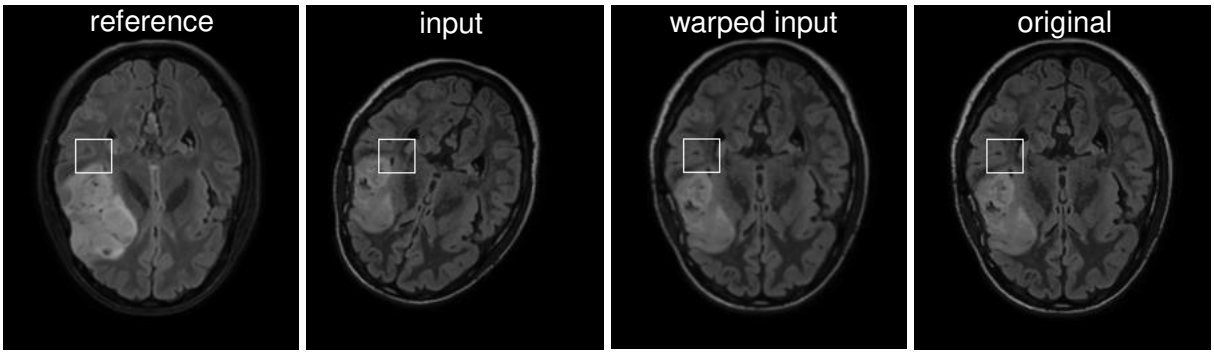


Figure 52: Actual rectangle for comparison on all slices

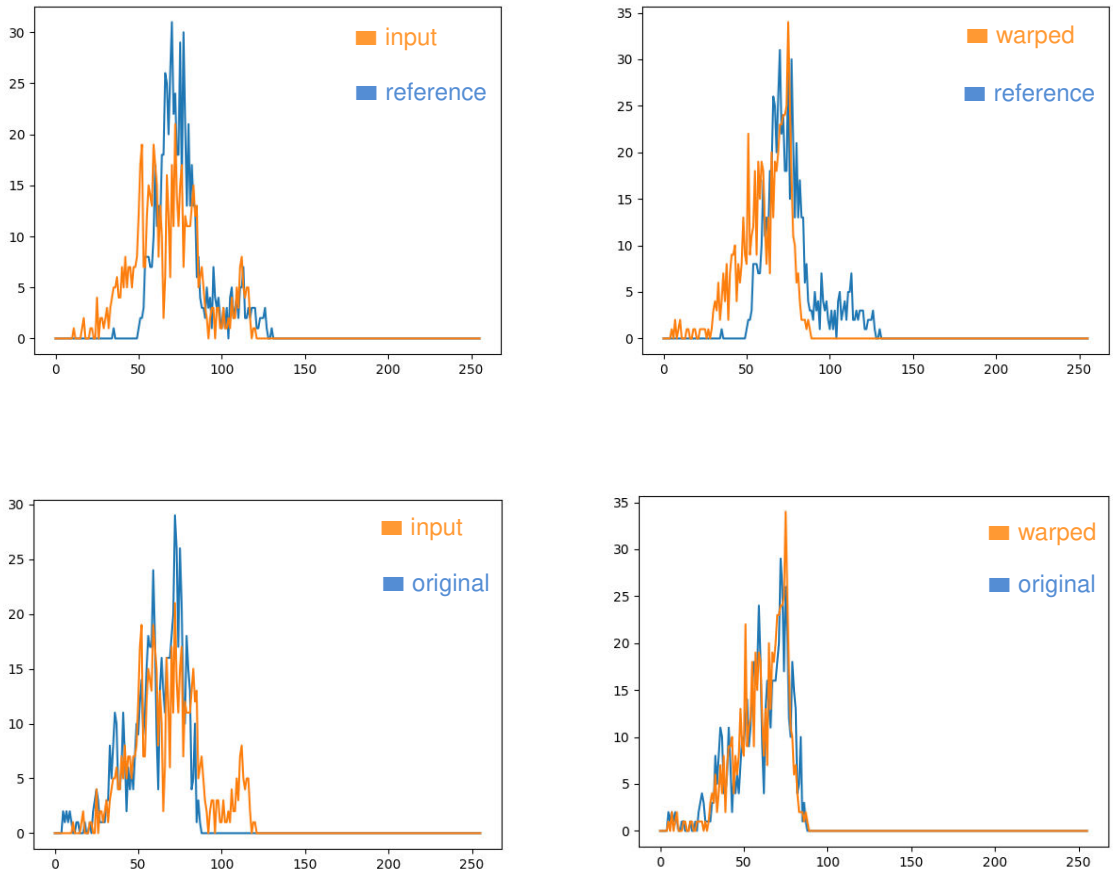


Figure 53: Histograms of rectangles from previous Figure 53

9.5.9 Results

While we have tested different configuration on the big dataset (23372 input slices), it is not possible to visualize all results in the one graph. We decided to use several visualizations to compare different configurations listed in the chapter 9.5.8.

One of the steps in rigid registration process is finding transformation matrix for affine (rigid) transformation. However, there are some cases, when the matrix was not found. In these cases the registration is unsuccessful and it fails. In the Figure 54 we can see the success of finding matrix of the different configurations.

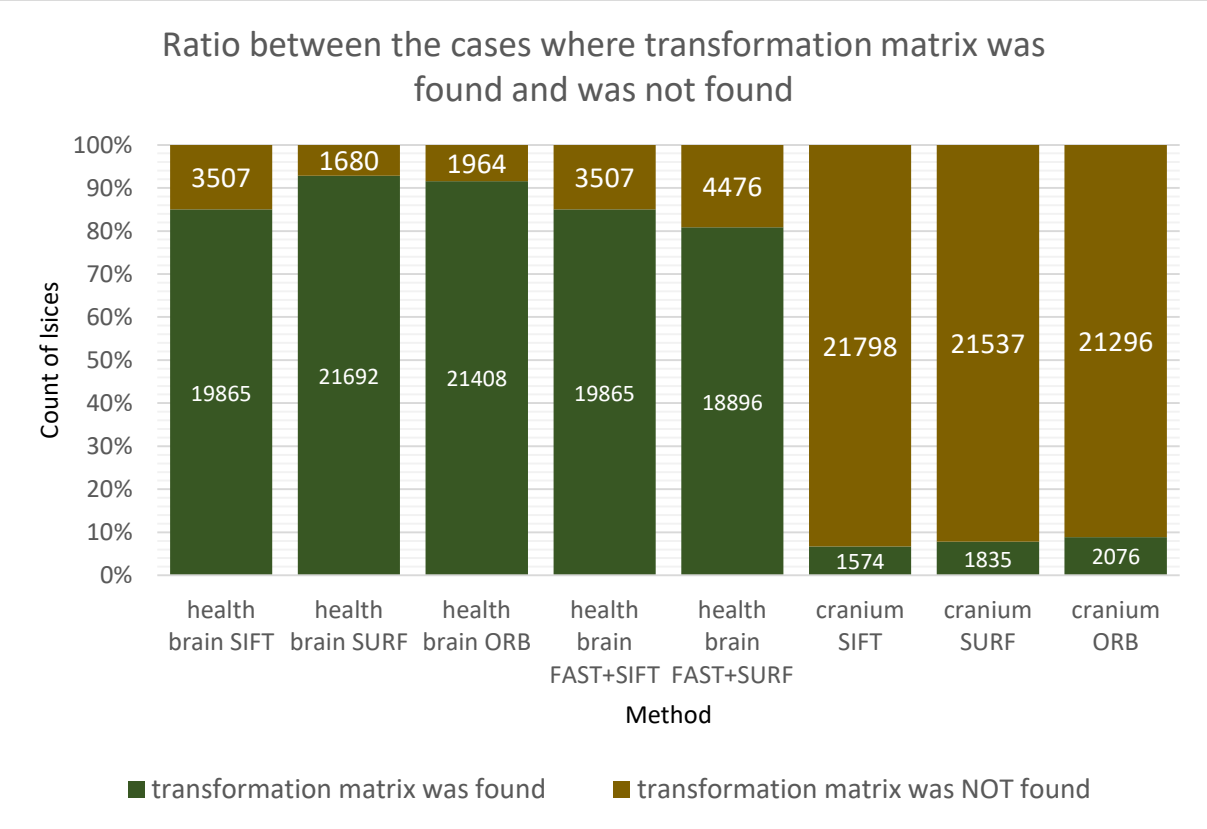


Figure 54: Success of finding transformation matrix by different configurations

When finding of the transformation matrix is successful, we can warp the slice and compute average correlations between the histograms of foreground slice parts as described in the chapter 9.5.8.3. If the correlation of the warped input data with correct original or reference data is higher than correlation of the input with correct original or reference data, then the registration was successful. In the Figure 55 we can see ratio between cases when the correlation of the warped (registered) input data and reference data was higher (lower) than correlation of the input data and reference data. In the Figure 56 we can see ratio between cases when the correlation of the warped (registered) input data and correct original data was higher (lower) than correlation of the input data and correct original data.

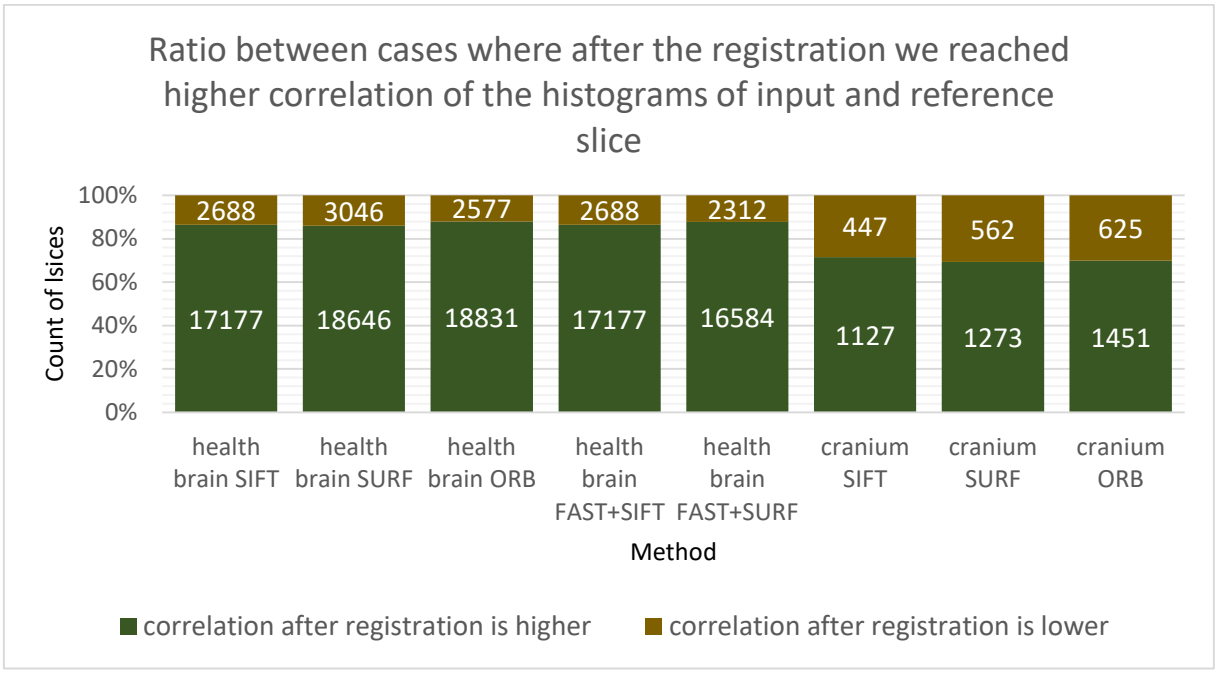


Figure 55: Ratio between cases when registration was successful (correlation between registered input and reference was higher that correlation between input and reference) and was not successful (vice versa)

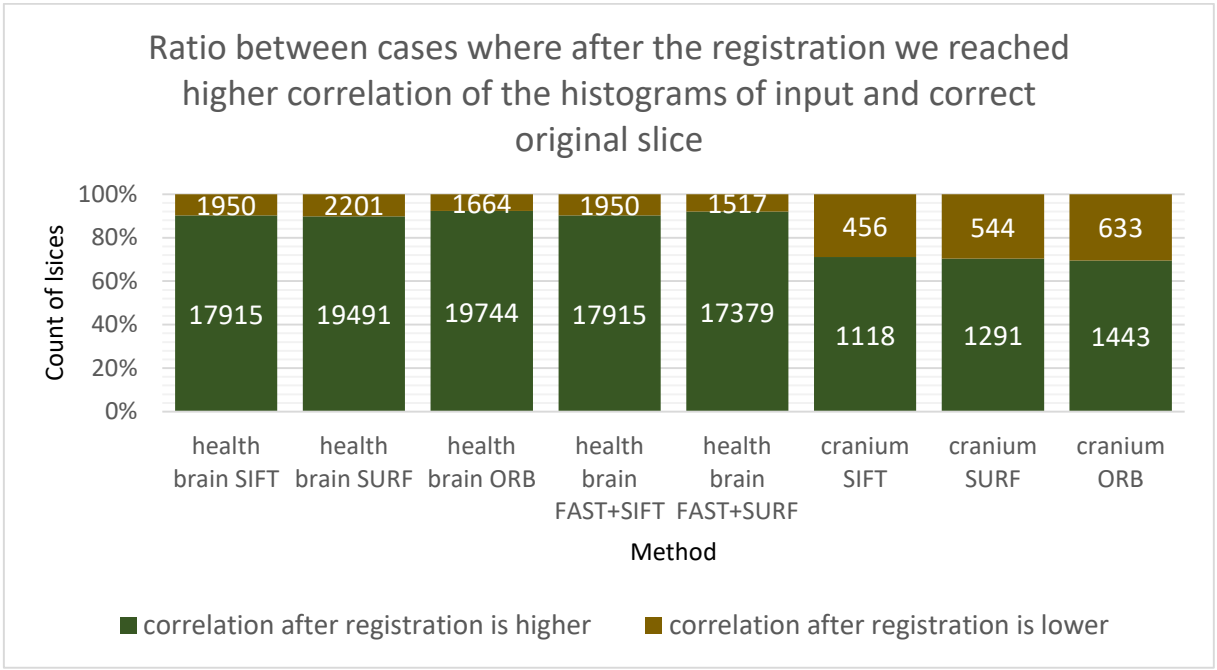


Figure 56: Ratio between cases when registration was successful (correlation between registered input and correct original was higher that correlation between input and correct original) and was not successful (vice versa)

For the different configurations we have compared average values of the histogram correlations between:

- input and reference vs. warped input and reference,
- input and correct original vs. warped input and correct original.

The results are shown in the Table 2 and visualized in one bar chart in the Figure 57.

	Average correlation between:			
	input and reference slice hist	warped and reference slice hist	input and correct original slice hist	warped and correct original slice hist
health brain SIFT	0.55925936	0.673267044	0.696120257	0.881003553
health brain SURF	0.558169989	0.679775182	0.685948337	0.885244291
health brain ORB	0.563983903	0.692641344	0.686401059	0.89787346
health brain FAST+SIFT	0.55925936	0.673267044	0.696120257	0.881003553
health brain FAST+SURF	0.581103639	0.703054977	0.705173425	0.901245473
cranium SIFT	0.586529999	0.65107211	0.702952447	0.792392358
cranium SURF	0.577200119	0.635127201	0.687570851	0.768805339
cranium ORB	0.566164306	0.625010991	0.683571548	0.765503716

Table 2: Comparison of average histogram correlations for different configurations

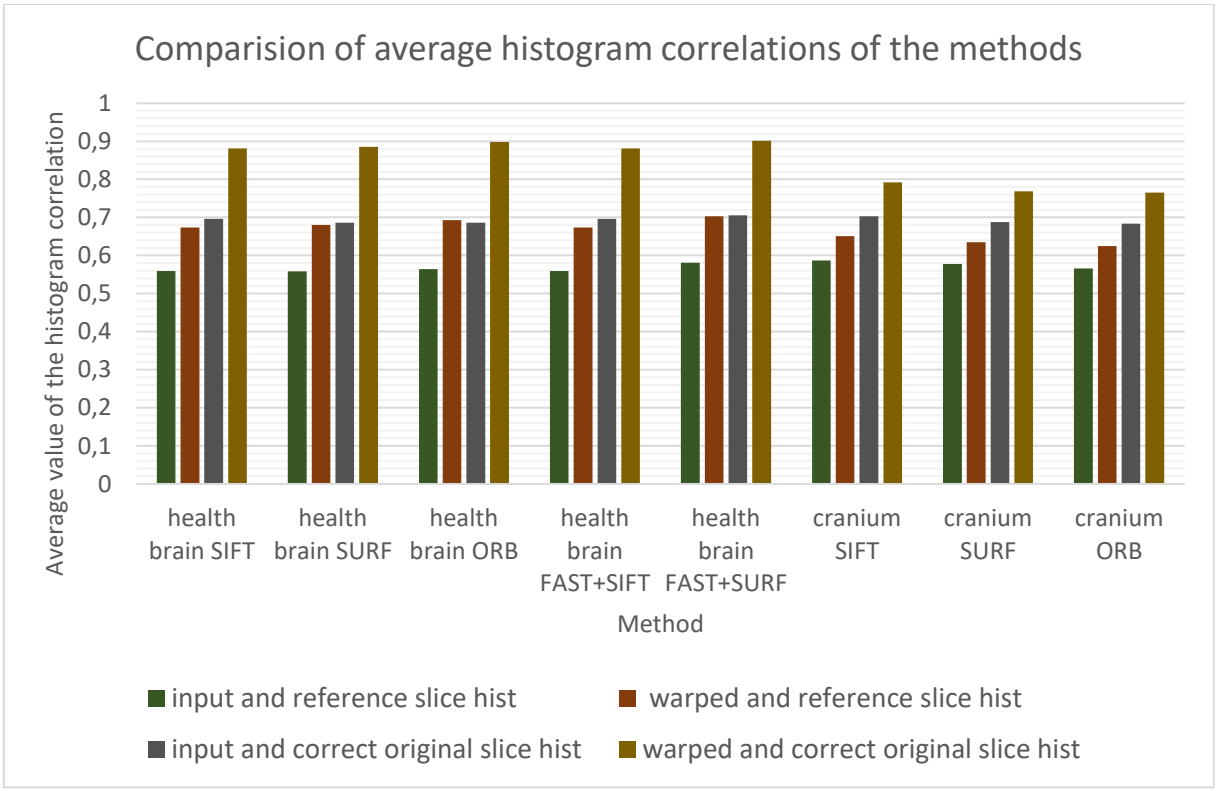


Figure 57: Comparison of average histogram correlations for different configurations

For different configuration we have computed minimum, bottom quartile, median, top quartile and maximum of the histogram correlations between:

- warped input and reference (visualized using box plot in the Figure 58)
- warped input and correct original (visualized using box plot in the Figure 59)

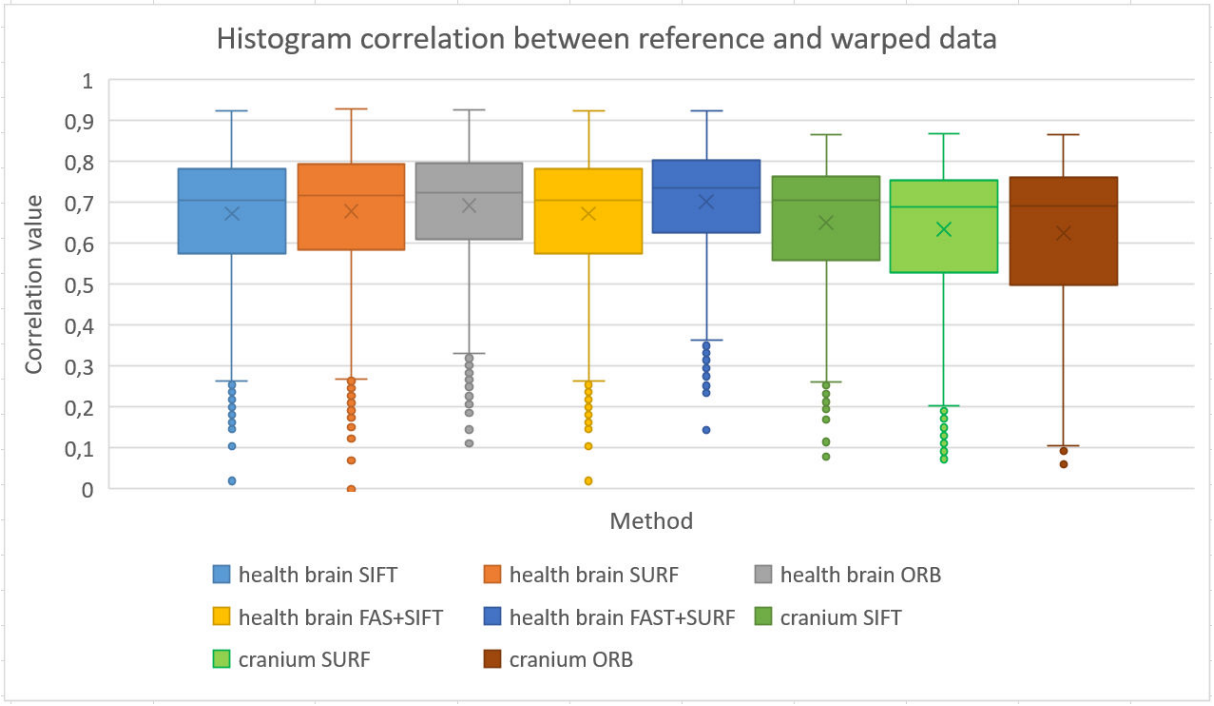


Figure 58: Histogram correlation between reference and warped data for different configurations

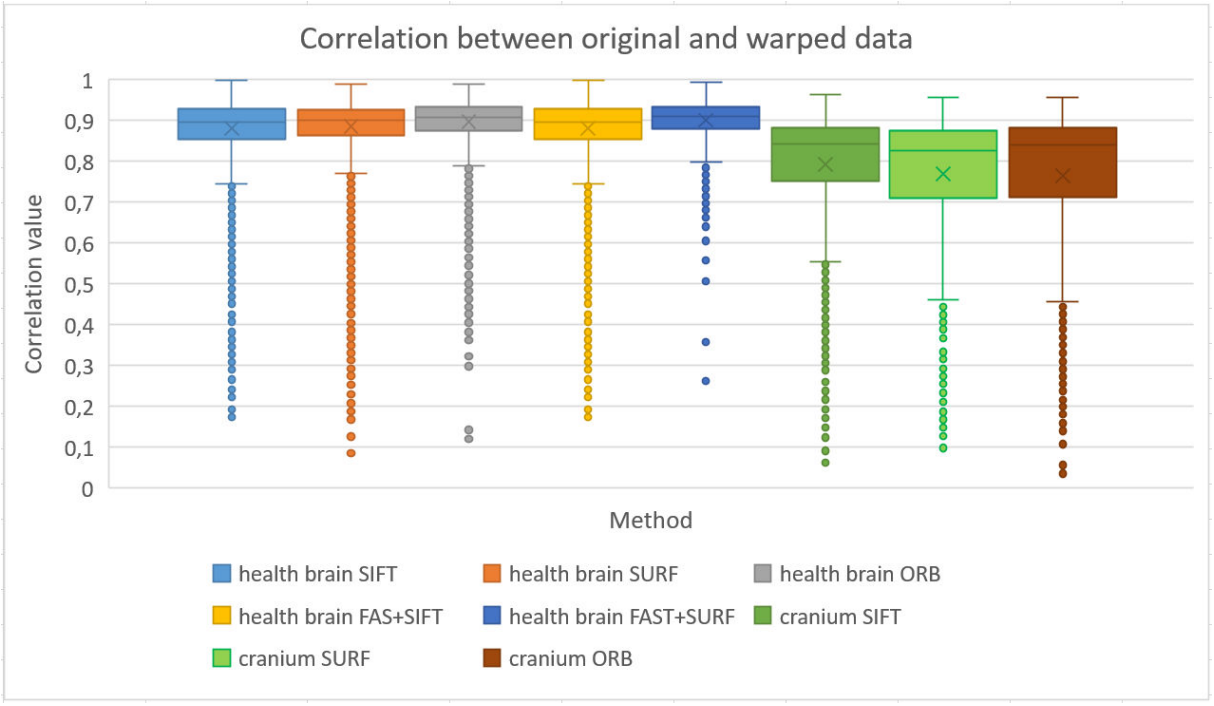


Figure 59: Histogram correlation between original and warped data for different configurations

For the different configurations we have compared average values of the histogram intersections between:

- input and reference vs. warped input and reference,
- input and correct original vs. warped input and correct original.

The results are shown in the Table 3 and visualized in one bar chart in the Figure 60.

	Average intersection between:			
	input and reference slice hist	warped and reference slice hist	input and correct original slice hist	warped and correct original slice hist
health brain SIFT	352.2922887	418.9958806	421.7702733	526.6356664
health brain SURF	351.3932037	422.8377119	415.9759987	529.1488422
health brain ORB	353.6720788	429.6340126	415.6346208	536.8878107
health brain FAST+SIFT	352.2922887	418.9958806	421.7702733	526.6356664
health brain FAST+SURF	364.7993639	435.2697225	427.487366	539.6562794
cranium SIFT	377.2203331	415.1668951	434.6328554	482.7449933
cranium SURF	374.0779159	408.2073459	426.9232801	470.5326106
cranium ORB	365.7259957	399.7655388	423.5275305	466.8360997

Table 3: Comparison of average histogram intersection for different configurations

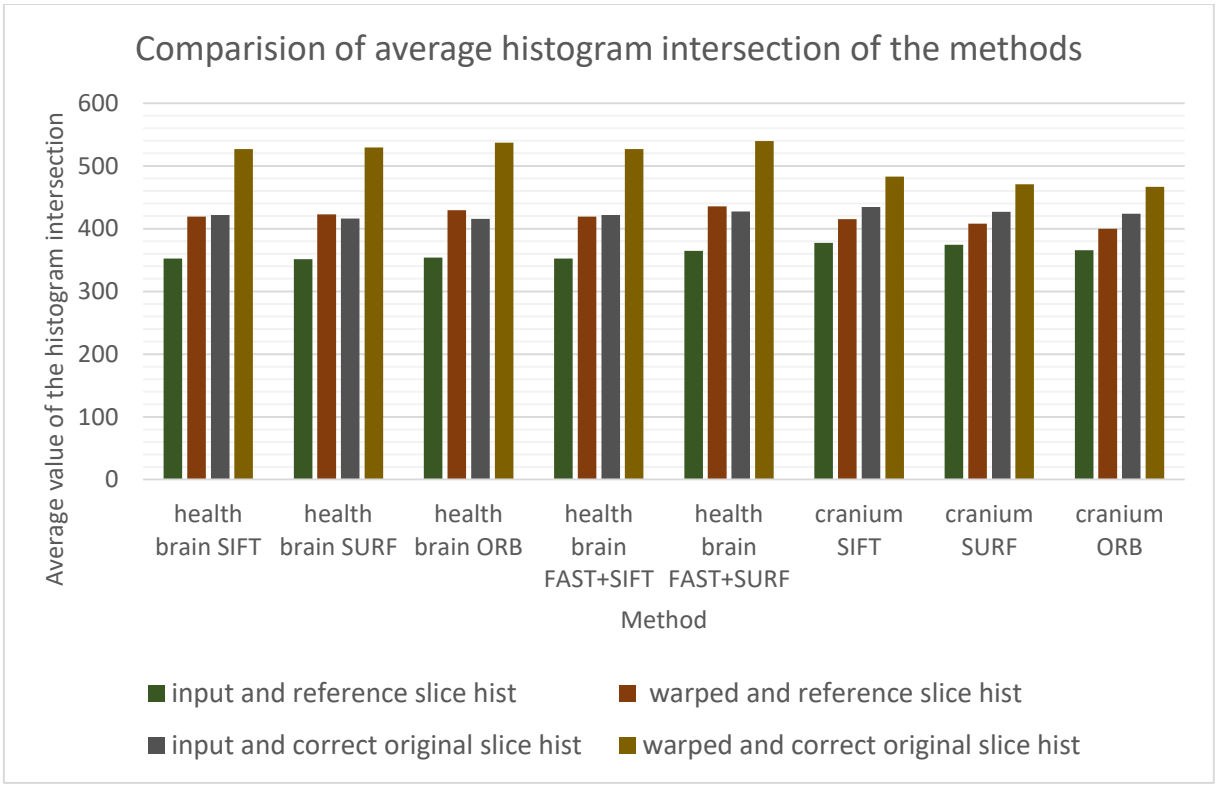


Figure 60: Comparison of average histogram intersection for different configurations

For different configuration we have computed minimum, bottom quartile, median, top quartile and maximum of the histogram intersections between:

- warped input and reference (visualized using box plot in the Figure 61)
- warped input and correct original (visualized using box plot in the Figure 62)

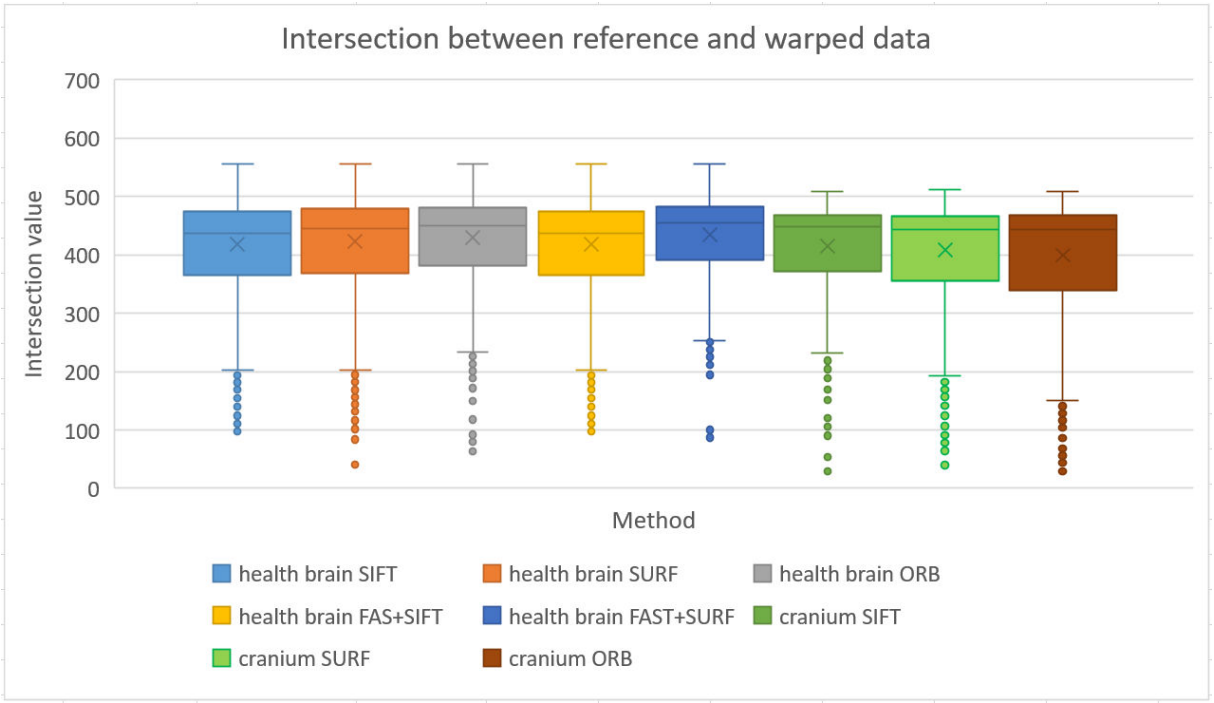


Figure 61: Histogram intersection between reference and warped data for different configurations

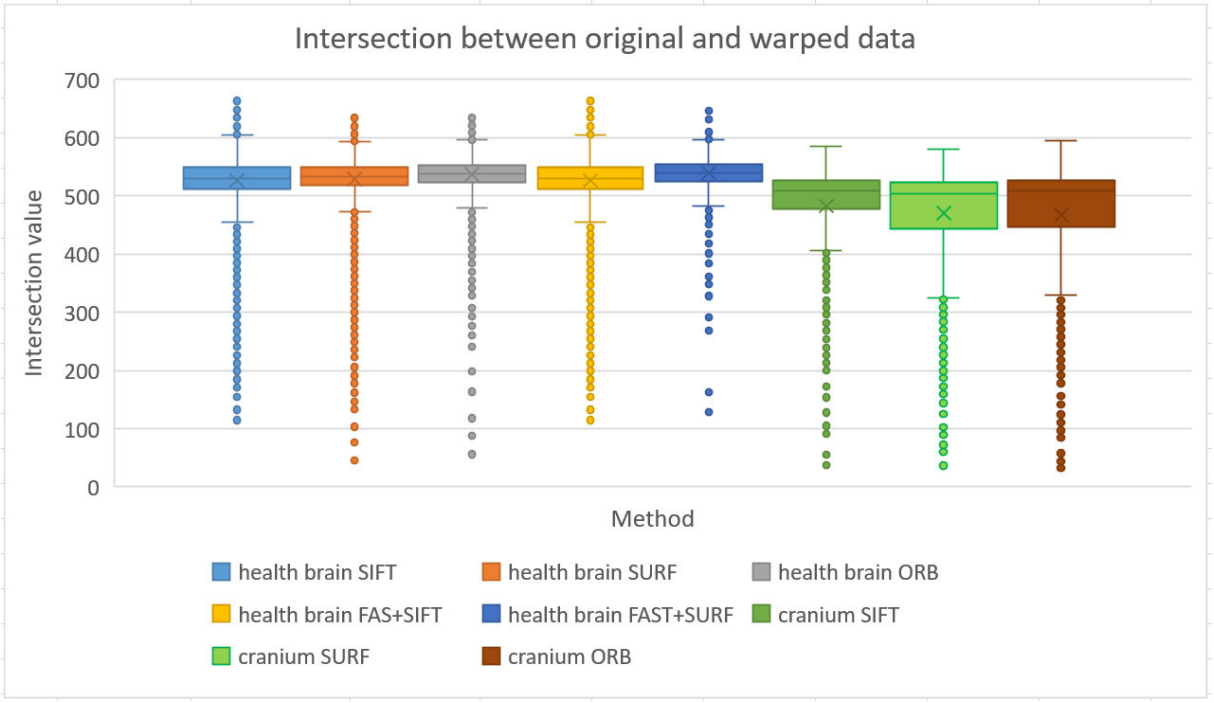


Figure 62: Histogram intersection between original and warped data for different configurations

For the different configurations we have compared average values of the histogram Bhattacharyya distances between:

- input and reference vs. warped input and reference,
- input and correct original vs. warped input and correct original.

The results are shown in the Table 4 and visualized in one bar chart in the Figure 63.

	Average bhattacharyya distance between:			
	input and reference slice hist	warped and reference slice hist	input and correct original slice hist	warped and correct original slice hist
health brain SIFT	0.496994897	0.409436171	0.40232794	0.268244571
health brain SURF	0.498047819	0.404393555	0.410093665	0.267288429
health brain ORB	0.494636076	0.395065031	0.410371469	0.257348448
health brain FAST+SIFT	0.496994897	0.409436171	0.40232794	0.268244571
health brain FAST+SURF	0.479674829	0.38757861	0.394690441	0.255031371
cranium SIFT	0.467723136	0.419414784	0.39071732	0.335073082
cranium SURF	0.472076045	0.428548565	0.401592521	0.350917864
cranium ORB	0.483003533	0.439819222	0.405216908	0.356111652

Table 4: Comparison of average histogram Bhattacharyya distance for different configurations

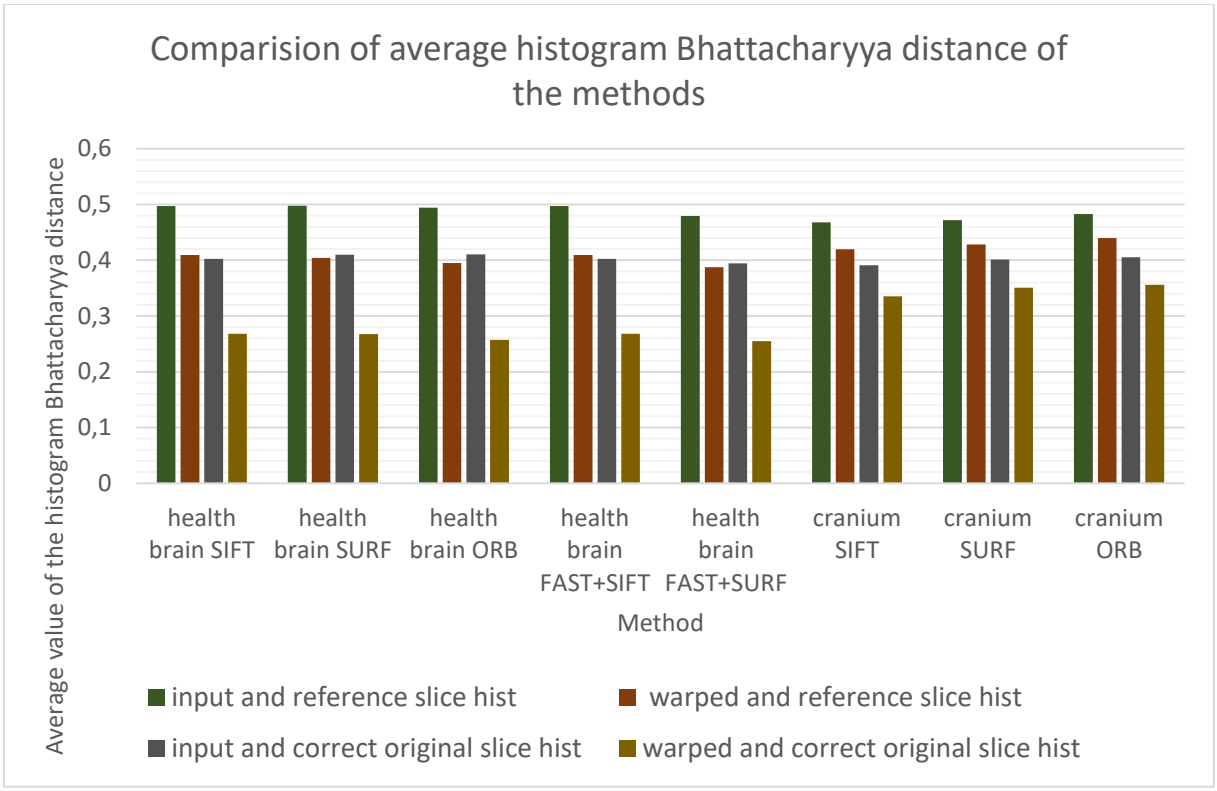


Figure 63: Comparison of average histogram Bhattacharyya distance for different configurations

For different configuration we have computed minimum, bottom quartile, median, top quartile and maximum of the histogram Bhattacharyya distances between:

- warped input and reference (visualized using box plot in the Figure 64)
- warped input and correct original (visualized using box plot in the Figure 65)

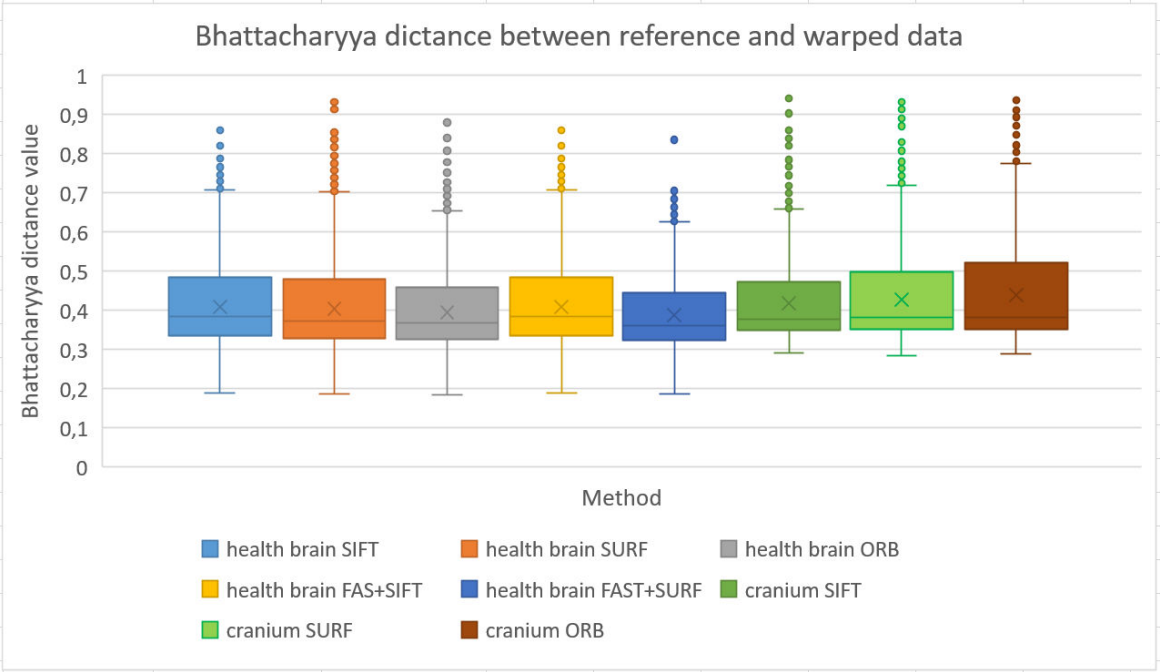


Figure 64: Histogram Bhattacharyya distance between reference and warped data for different configurations

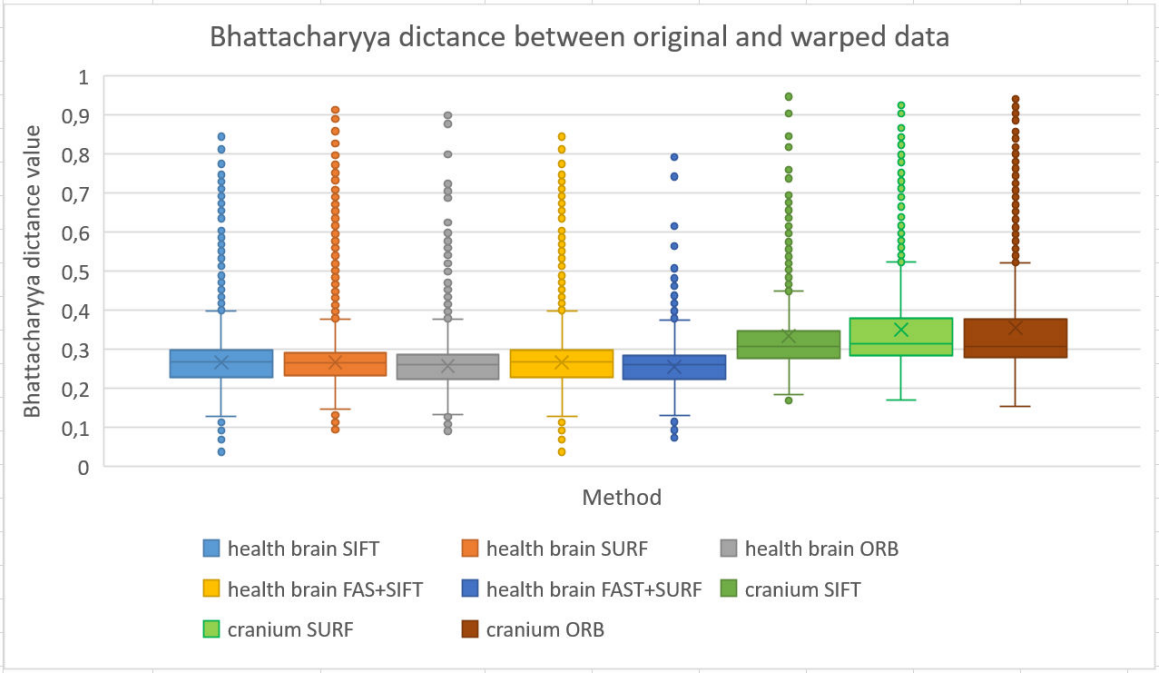


Figure 65: Histogram Bhattacharyya distance between original and warped data for different configurations

We can see in the charts and tables, that health brain ORB configuration reach the best results. Thus, we tried to watch closer on the results of this configuration. While we are working with big dataset, it is not possible to visualize results of all registrations. We created the histogram of correlations between:

- reference and input slices vs. reference and warped slices (Figure 66),
- correct original and input slices vs. correct original and warped slices (Figure 67).

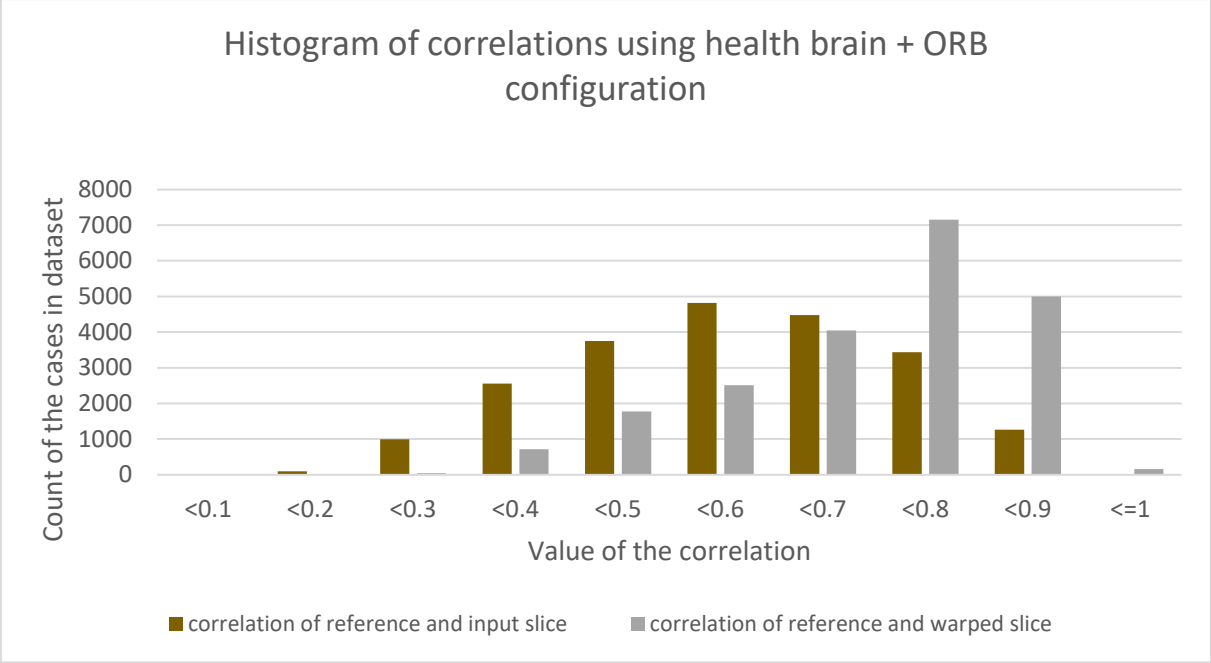


Figure 66: Histogram of correlations (health brain ORB configuration), reference vs. input

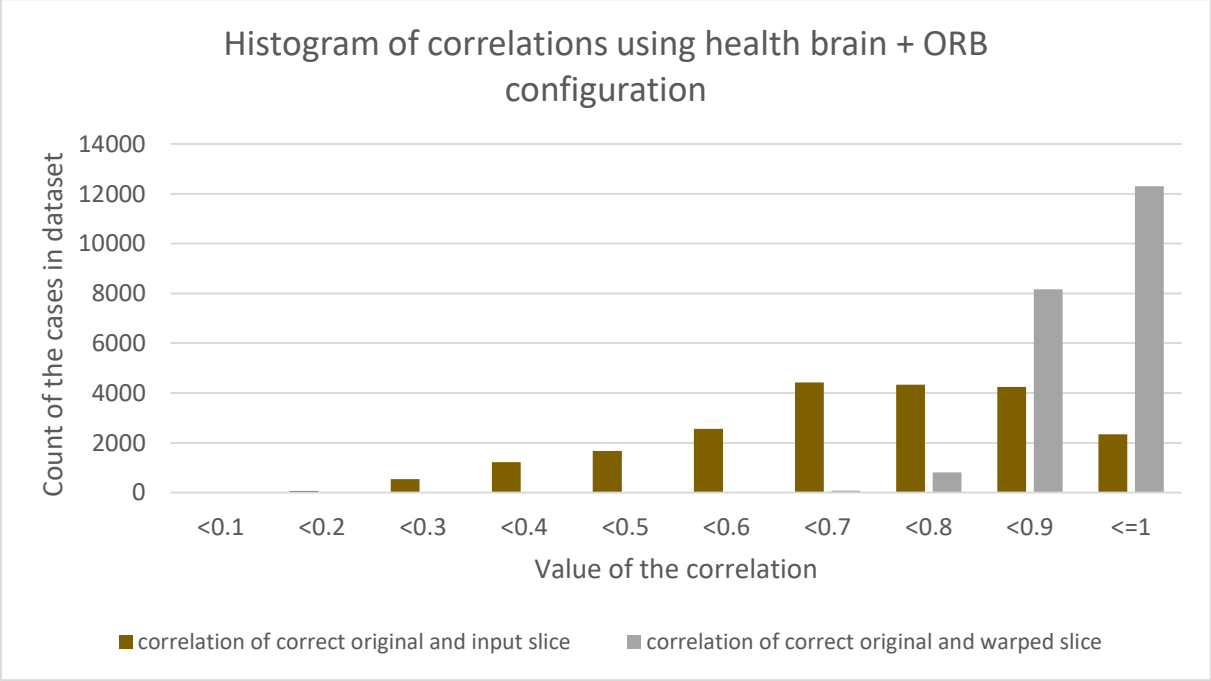


Figure 67: Histogram of correlations (health brain ORB configuration), correct original vs. input

We have also tried to compute differences between the correlations of:

- reference and input slices vs. reference and warped slices (visualized by histogram of differences in the Figure 68),
- correct original and input slices vs. correct original and warped slices (visualized by histogram of differences in the Figure 69).

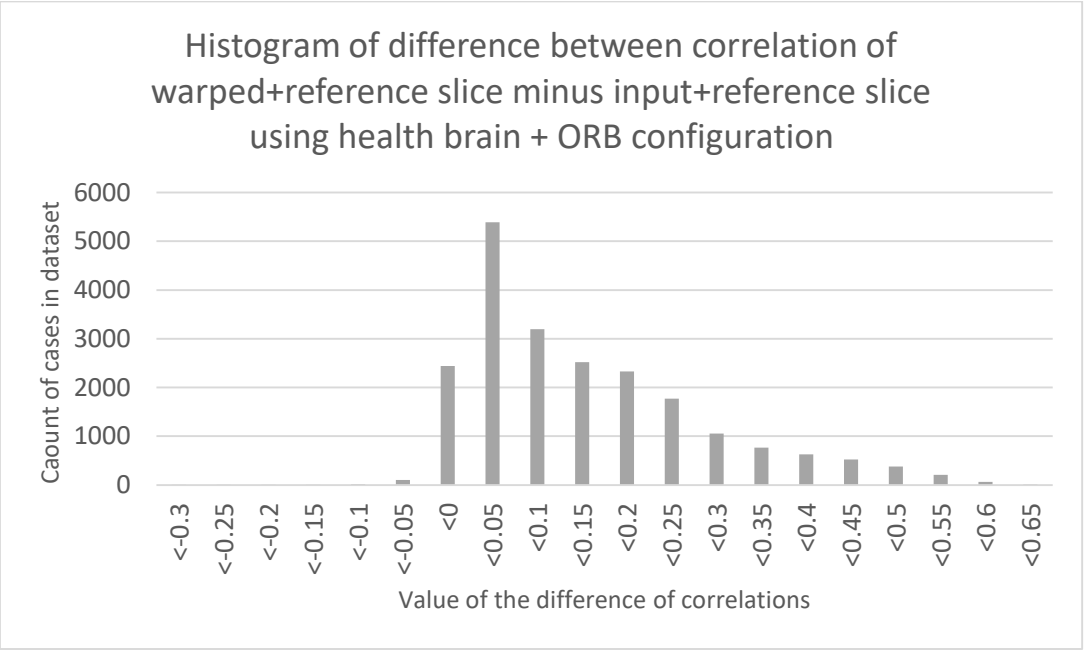


Figure 68: Histogram of correlation differences between reference minus input (health brain ORB configuration)

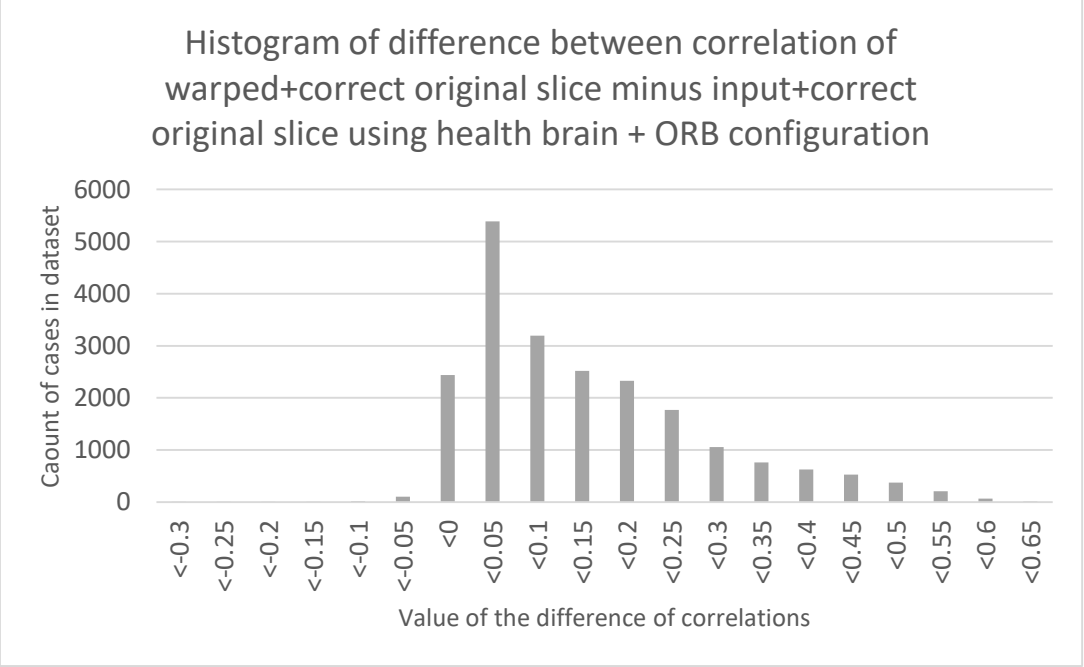


Figure 69: Histogram of correlation differences between correct original minus input (health brain ORB configuration)

In the Figure 66 and Figure 67 histogram distribution shows that histogram correlation between registered input and reference or correct original is in the most cases higher than correlation between input and reference or correct original. Thus registration is in the most cases successful. This is confirmed by the Figure 68 and Figure 69 where we can see that in the most cases difference between the histogram correlation of registered input with reference or correct original and between input with reference or correct original are in the most cases positive.

We have also compared time durations of the different configurations. Average time of processing 2 slices is presented in the Table 5 and visualized in the Figure 70.

	Average time duration
health brain SIFT	0.135198092
health brain SURF	0.021823367
health brain ORB	0.012575842
health brain FAST+SIFT	0.13260751
health brain FAST+SURF	0.01880375
cranium SIFT	0.109932226
cranium SURF	0.024158819
cranium ORB	0.010333754

Table 5: Average time duration of the methods

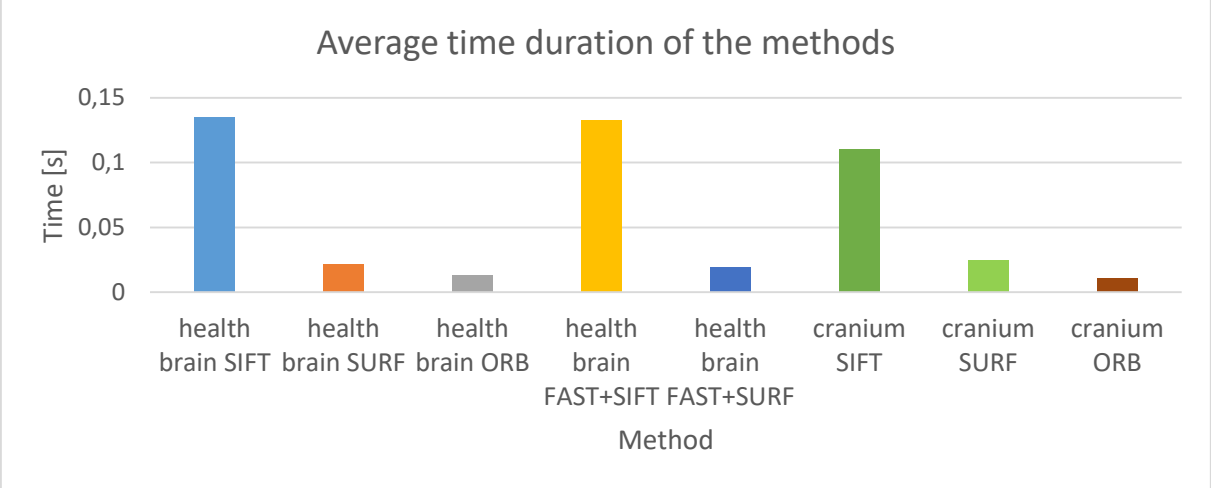


Figure 70: Average time duration of the methods

9.5.10 Implementation

Rigid registration can process 2D slice created from 3D volumes. Input for the method are two 2D slice – one reference and one input slice which will be registered to reference slice. Algorithms used in our proposed method of rigid registration are from OpenCV library. Also NumPy library is used to store slice and perform simple operations.

9.5.10.1 Structure of the source code presented by a class diagram

RigidRegistration class is organized in *logic* package and used by the *main* function as is shown in the Figure 71. *RigidRegistration class* has three important parameters:

- *registration_mask_type* – health brain mask or cranium mask
- *key_point_detector* – used detector for key point detection (SIFT, SURF, ORB, FAST)
- *key_point_descriptor* – used descriptor for computing feature vectors (SIFT, SURF, ORB)

And also implements several methods:

- *rigid_registration()* – method preprocess data – two input 2D slices
- *__create_mask_for_registration()* – creates mask for the registration
- *__detect_key_points_SIFT(), __detect_key_points_SURF(), __detect_key_points_ORB(), __detect_key_points_FAST()* – methods find key points on the slices
- *__calculate_descriptors_SIFT(), __calculate_descriptors_SURF() , __calculate_descriptors_ORB()* – methods computer feature vectors of the key points
- *__get_matches()* – returns all matches between the feature vectors
- *__get_good_matches()* – filter only good matches from all matches

In the source code is more parameters and methods, but they have got only help function.

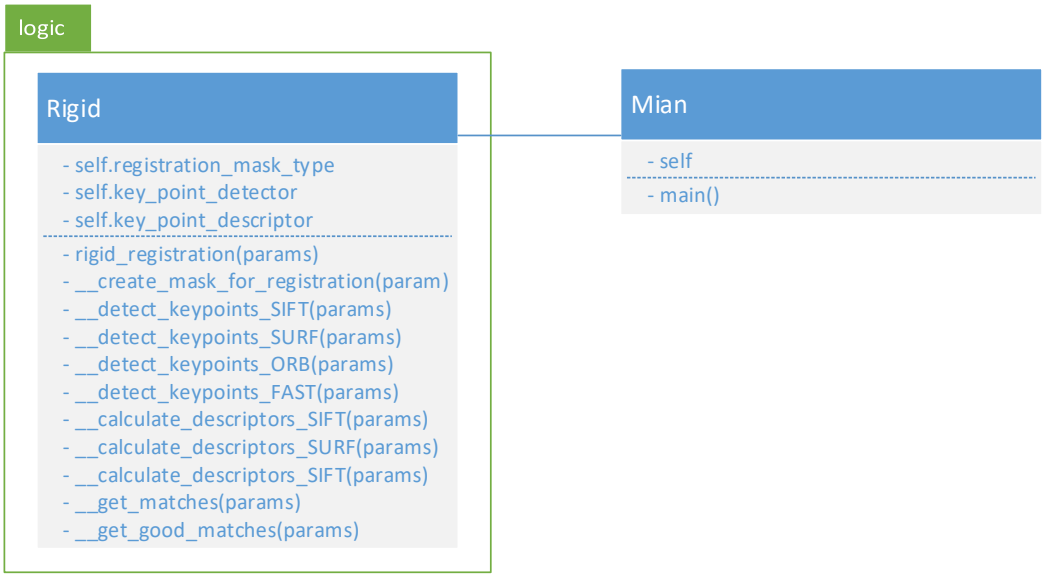


Figure 71: Class diagram of rigid registration (UML notation)

9.5.10.2 Interaction of the objects presented by a sequence diagram

In the Figure 72 is shown sequence diagram of the rigid registration. *RigidRegistration* is used by the *main*. First we have to initialize *RigidRegistration* and set configurations of the method. At first, we create registration mask for both input slices. Registration mask can be created for health brain or cranium. Next step is detection of key points on the both slices using SIFT, SURF, ORB or FAST detector. In the next step descriptors (SIFT, SURF or ORB) compute feature vectors for all key points

and then these vectors are matched. Resulting matches are filtered to get only good matches. Finally, matrix for the transformation is found and used to warp the input slice – register it to reference slice.

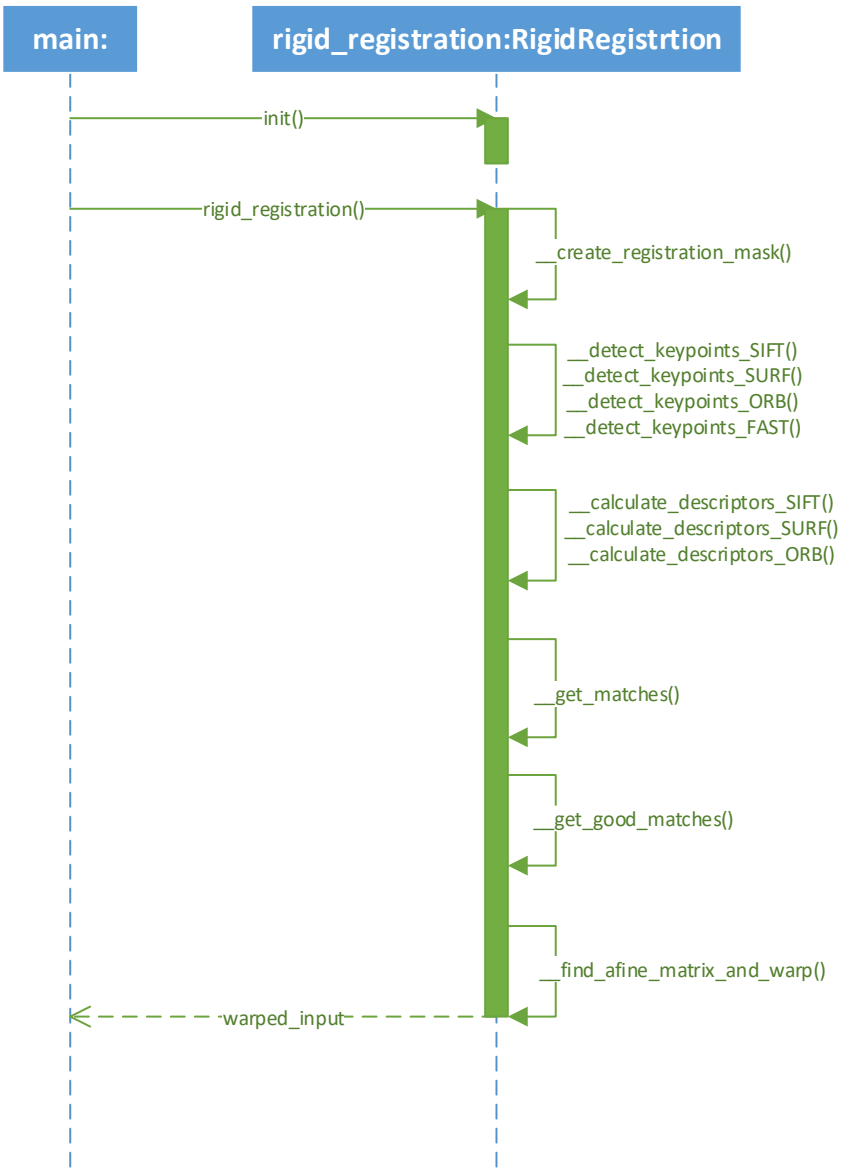


Figure 72: Sequence diagram of rigid registration (UML notation)

9.5.11 Discussion

In our approach we have tried to crate registration mask from health brain part or from the cranium. Registration mask created from cranium was problematic for our method. In the Figure 54 we can see that method was not able to find affine matrix for registration when we used mask of the cranium for registration. There were also some very successful registrations, but in general the method did not work well on the cranium part. This was caused by several problems:

- sometimes there was not enough key points detected so the method did not have enough matches after the finding matches step as is shown in the Figure 73

- sometimes only or mainly wrong matches of feature vectors were created as is shown in the Figure 74

- another times in the matches filtering step, our algorithm did not filter wrong matches properly as is shown in the Figure 75

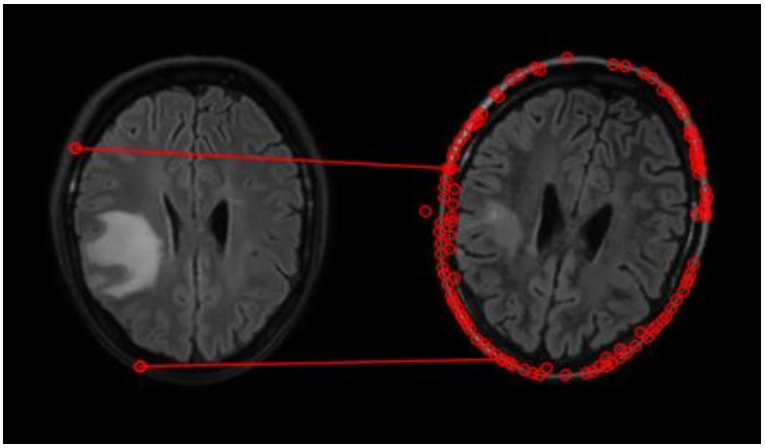


Figure 73: Unsuccessful segmentation caused by not enough key points detected

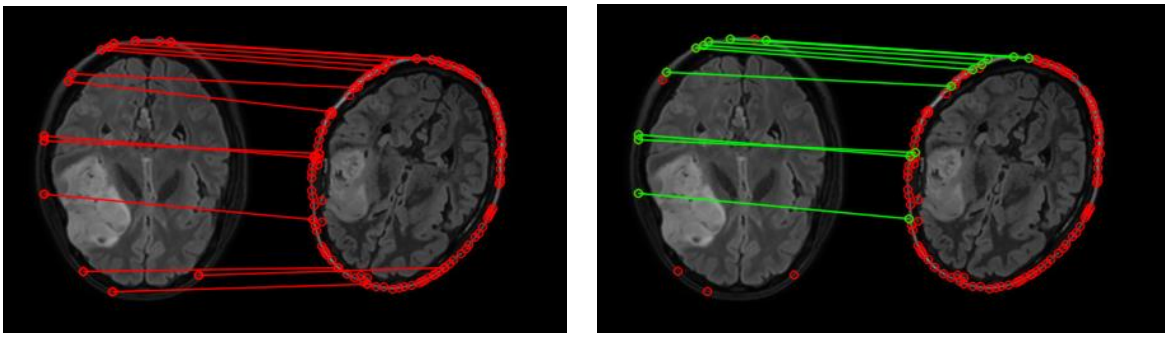


Figure 74: Unsuccessful segmentation caused by creating mainly bad matches between the feature vectors

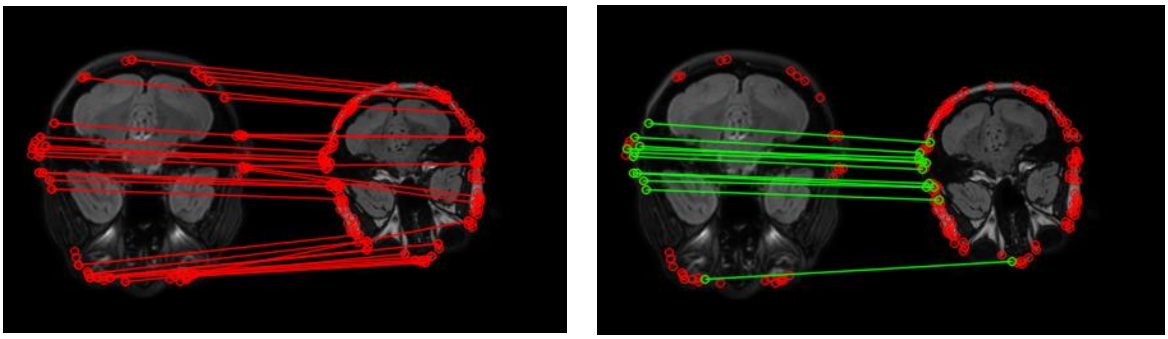


Figure 75: Unsuccessful segmentation caused by wrong matches filtering

Our proposed method works better when we register slices by the health brain parts and the majority of slices were registered successfully as we can see in the Figure 55 and Figure 56. From the charts from the Figure 57 to Figure 65 we can see that the best results reached health brain ORB configuration and comparatively also health brain FAST-SURF configuration. However, there are still

cases when the method did not register slices successfully. In the Figure 76 we can see some examples of successful registrations using health brain ORB configuration and in the Figure 77 we can see on the other side cases of unsuccessful registration using the same configuration. Failure of the method in the testing slice was caused by matching bad feature vectors in matching step and northerly filtration of the matches in filtration matches step.

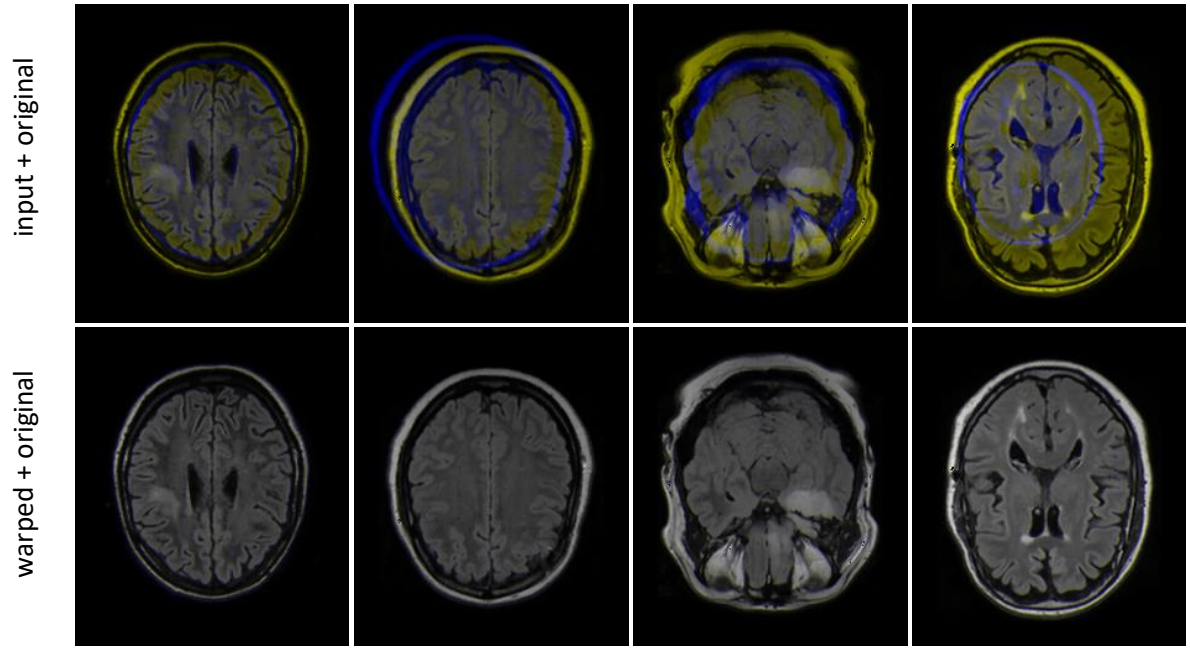


Figure 76: Examples of successful registration: merged blue channel of input (top row) or warped input (bottom row) with red and green channel of correct original

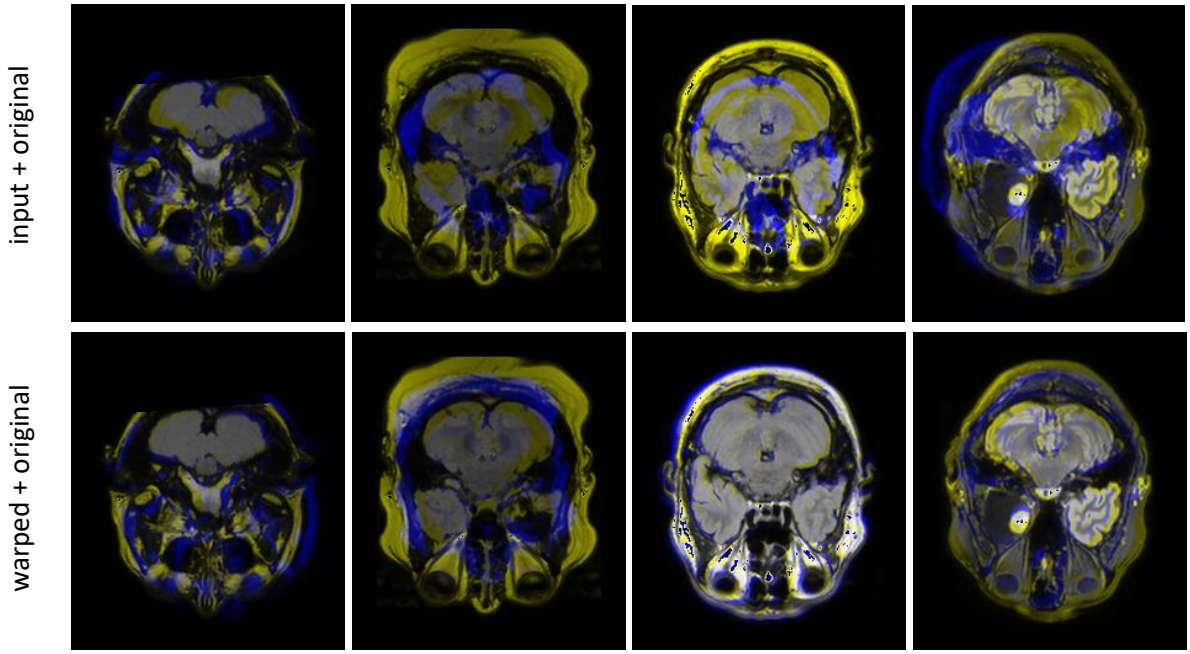


Figure 77: Examples of unsuccessful registration: merged blue channel of input (top row) or warped input (bottom row) with red and green channel of correct original

9.6 Non-rigid registration

Non-rigid registration is crucial step for the observation of the brain changes caused by the tumor. Tumor grows and changes irregularly depending on the surrounding health brain structures and it pushes on those surrounding structures. Thanks the non-rigid registration we should be able to track changes of the health brain parts but also of the tumor. There are several approaches which try to deal with the similar problem. Many of them use brain atlases for non-rigid registration and simulation of the tumor growth. They follow different goals, such as tumor segmentation or pre-operative and post-recurrence scan registration. Chapter 8.2 Registration is devoted to the mentioned methods. In our work we decided to implement and test different approach, because our goal is a little different.

Basic flow is shown in the Figure 78. Input for the method are 3D volumes. However, we are using algorithms from the library which can process only 2D images, so our method works slice by slice. In the first step, the method finds optical flow using Farneback or Lukas-Kanade optical flow algorithm. In the next step we compute corresponding points from the flow. This is important mainly for the Farneback algorithm, because it returns dense flow. Lukas-Kanade returns a set of points, we only change their structure for the next step – visualization.

Final step is visualization of the changes. We tested different approaches to the visualization:

- paint tracks,
- paint HSV from dense flow and
- image morphing.

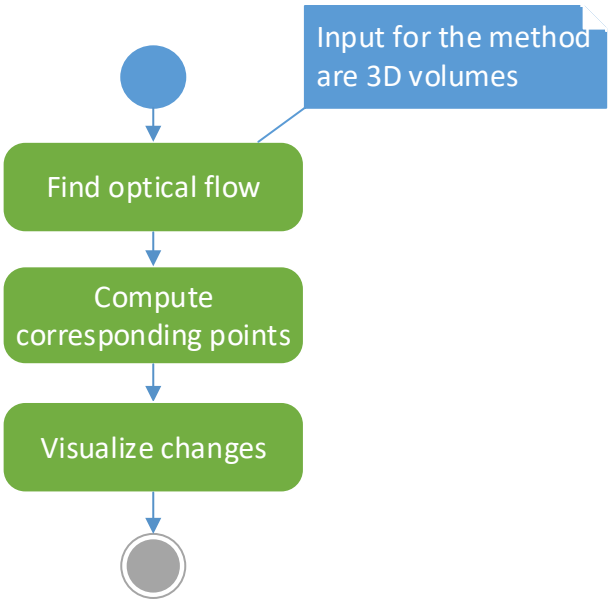


Figure 78: Flow of non-rigid registration represented by activity diagram (UML notation)

9.6.1 Input

Input for the proposed method is several examinations of one patient – several 3D MRI volumes captured in different time – in some regular period. Actually, the method can process 3D data, but it still works slice by slice in axial direction. We try to register old examination with actual examination. In this chapter we present the steps of the method on the slices 93 from the volumes from the BRATS 2015 dataset - pat_153 examinations 0109 (old examination) and 165 (actual examination). We work with preprocessed images using the preprocessing described in the chapter 9.3.Example of the input is shown in the Figure 79.

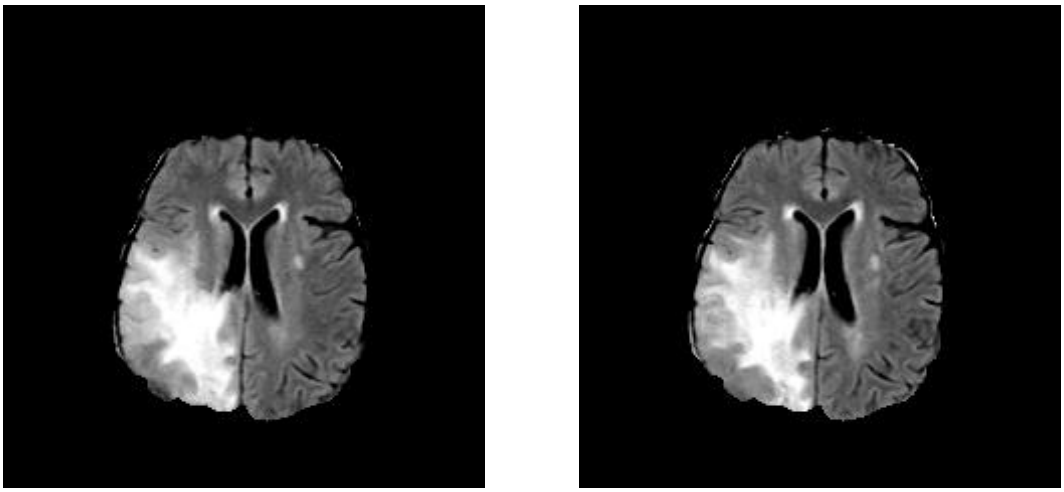


Figure 79: preprocessed axial MR slice 93 from BRATS 2015, pat_153,
LEFT (old): examination 0109, RIGHT (actual): examination 165

9.6.2 Finding optical flow

As we can see in the Figure 79, slices look really similar nevertheless the fact that they came from different examinations of one patient. In our approach we have tested method based on the key points and description of the key point features, but the results were bad. The main reason was that proposed method had find matches between the feature vectors of the very distant and disagreeable key points. It is because some parts of the brain have got very similar intensities.

Rather than reduce the area of the finding the best matches for the key points we decided to test optical flow algorithms. We supposed that they are better suited for the problem. They are usually used to track the motion in the videos – in consecutive frames. In our case we do not have consecutive frames, but we can replace them by corresponding slices from following examinations.

In general, optical flow algorithms work on two basic premises:

- intensities of the object do not change between two consecutive frames – in our case two (or more) corresponding slices from the two (or more) examinations followed by each other (this is another reason why we head to preprocess the data)

- motion of the neighboring pixels is similar

We decided to use two different algorithms and compare their results:

- Lukas-Kanade optical flow algorithm – algorithm was invented by Bruce D. Lucas and Takeo Kanade. The algorithm uses mentioned basic premises. It combines information from several nearby pixels by creating small patch and the premise is that all pixels under the patch have got the same motion. Then the algorithm checks the similarities of the intensities on the next frame and thanks this it can compute the motion of these patches.

- Farneback dense optical flow algorithm – algorithm was invented by G. Farneback. Farneback's algorithm computes dense optical flow on dense grid of the points. At the beginning, the algorithm approximates each neighborhood of two following frames by quadratic polynomials. In the next step, quadratic polynomials are considered, a new signal is constructed by a global displacement. Finally, the coefficients in the quadratic polynomials' yields is used to calculate this global displacement by equating.

Those algorithms are suitable for the problem, because they can compute only small motions between two frames and this is actually what we need.

9.6.3 Computation of the corresponding points

In this step we have to compute the corresponding points. Output from the previous methods is an optical flow.

Actually Lukas-Kanade returns flow as corresponding points, thus we only have to change the structure for the next steps.

However, Farneback algorithm returns dense optical flow – motion of the all pixels from the frame. This is too much information for the next processing, so we choose only every x^{th} pixel from the motion vectors we compute the coordinates of the corresponding points in the next frame.

9.6.4 Visualization of the changes

Final step of the non-rigid registration is visualization of the changes. We have tested different techniques of the visualization:

- paint tracks,

- HSV from dense flow and

- image morphing.

9.6.4.1 Paint tracks

Paint tracks is the simplest visualization method. It is based only on the painting lines between the corresponding points. In the Figure 80 are shown changes between the slices from the Figure 79 using paint tracks technique. We can see in the figure that visualization of corresponding points created by the Farneback have more arrows and on the other side visualization of corresponding points created by Lukas-Kanade algorithm contains only several dots whole dots represents places where algorithm

detected the motions. This is caused by the fact that Farneback calculates dense flow and Lukas-Kanade only try to detect the motion and this is probably too hard task for Lukas-Kanade algorithm.

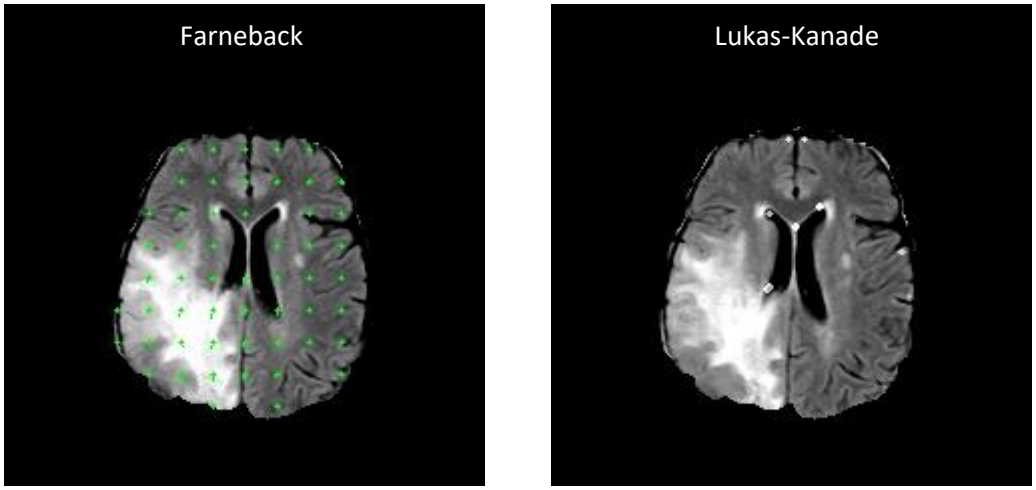


Figure 80: Visualization of changes between examination using paint tracks visualization technique

9.6.4.2 HSV from dense flow

Next visualization technique is to compute HSV image (hue, saturation, value) from the dense flow. This technique may be used only for flow computed by Farneback optical flow algorithm because it returns dense flow (Lukas-Kanade do not return dense flow).

Flareback returns a flow matrix with the same size as the input frames. Each element in the matrix is a value that represents the displacement of the concrete pixel between actual and previous frame. Thus, we compute angle of the movement and magnitude of the movement. Then we use these values to create HSV image where angle is H (hue) and magnitude is V (value). S (saturation) is constantly set to 255. Finally, we convert HSV image to RGB. Result is shown in the Figure 81.

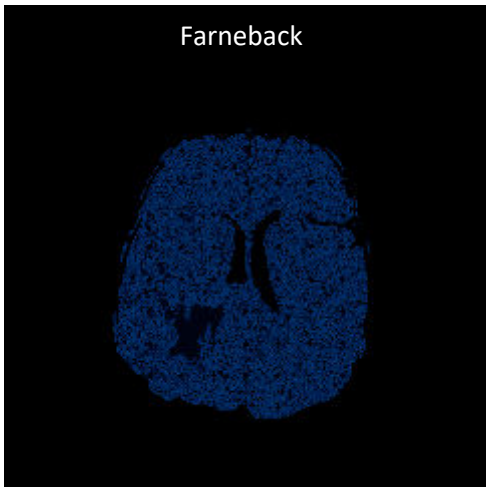


Figure 81: Visualization of changes using HSV image

9.6.4.3 Image morphing

Last visualization technique is called image morphing. This technique allows us to create animations and animate changes between the examinations. Actually, we could create only simple animation using classic animation - only slice by slice, but we decided to try create additional images using image morphing technique. Corresponding points from the previous step are used to create additional frames for the smooth and clearly animation.

Actually we could create additional animation using only simple blending of the two slices according to the formula:

$$N(x, y) = (1 - \alpha) * O(x, y) + \alpha * A(x, y),$$

where $N(x, y)$ is pixel from the new additional image, α is parameter which controls blending, $O(x, y)$ is pixel from the old image and $A(x, y)$ is pixel from the actual image.

However, image morphing allow to create more clearly animation.

Corresponding points are two sets of the points as is shown in the Figure 82 a). One set is from old slice and second one are corresponding points from the actual slice. In the first step we decided to add four more points to both sets. These additional points are top of bounding rectangle of the brain. Thus we create a bounding rectangle of the brain as is shown in the Figure 82 b) Then we extract the tops of the rectangle (Figure 82 c)) and add them to the sets as is shown in the Figure 82 d).

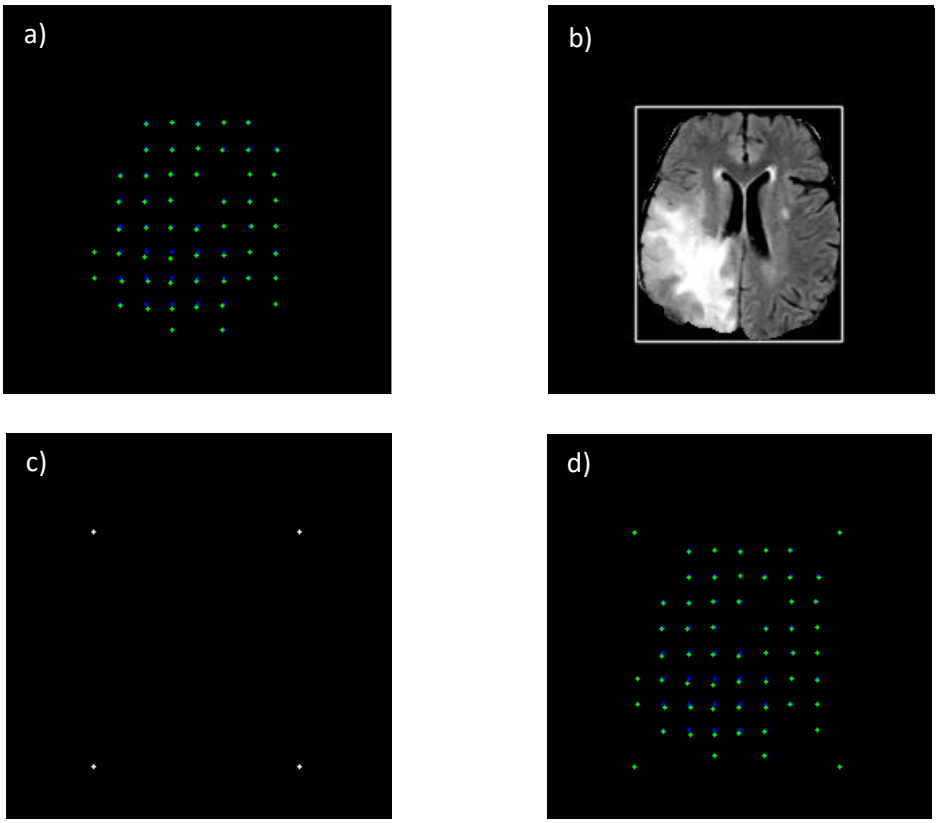


Figure 82: Sets of corresponding points from old and actual slice and creation of additional points

In the next step we calculate the average position of the corresponding points using the formula:

$$x_n = (1 - \alpha)x_o + \alpha x_a$$

$$y_n = (1 - \alpha)y_o + \alpha y_a,$$

where x_n and y_n are new coordinates of the point, x_o and y_o are coordinates of old slice, x_a and y_a are coordinates of the actual slice and α is parameter which controls blending.

Result is third set of points as is shown in the Figure 83.

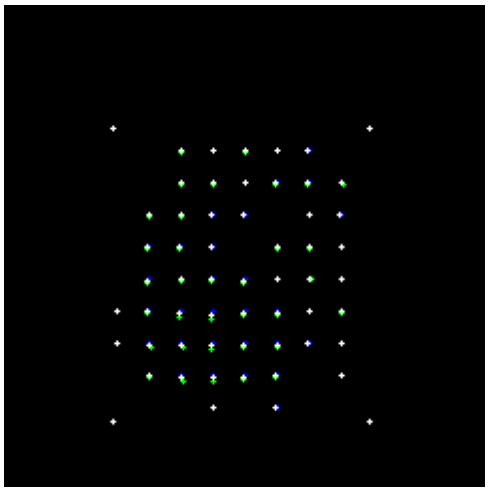


Figure 83: Sets of corresponding points (green and blue) and set of average corresponding points (white)

Next step is computation of the triangles between the points from the new set of points created in previous step. We decided to use Delaunay triangulation. Triangles are created in the iterations. In the Figure 84 a) is visualized result of the algorithm. From the triangles we compute corresponding triangles in the set of points from old slice (Figure 84 b)) and also set of points of the actual slice (Figure 84 c)).

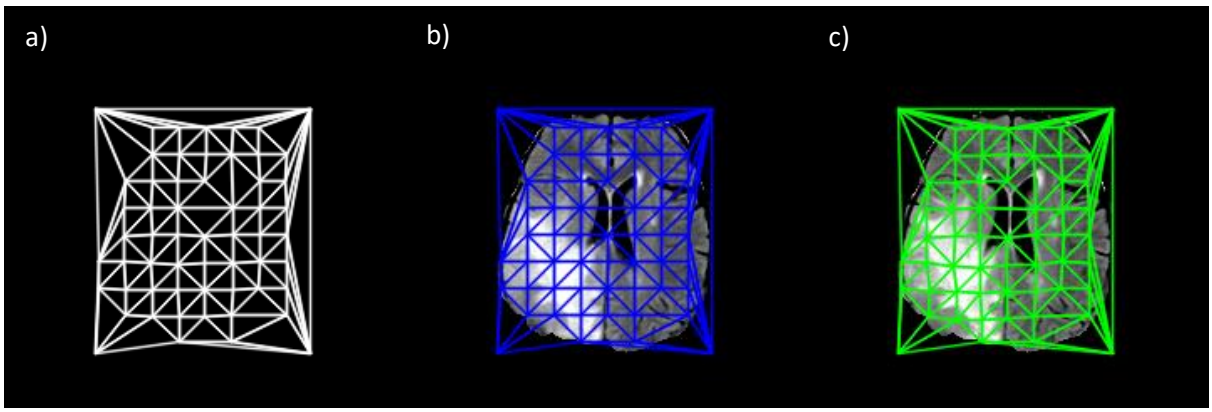


Figure 84: Result of Delaunay triangulation

Finally, we can create new additional image for the animation. We warp all triangles from old slice (Figure 84 b)) and all triangles from actual slice (Figure 84 c)) to the tringles created by Delaunay

(Figure 84 a)). For every triangle we compute affine matrix and we transform it rigidly. Resulting additional image is created by blending all warped corresponding triangles using the formula from the page 118.

9.6.5 Testing

We tested the method using only Farneback optical flow, because Lukas-Kanade algorithm did not looked well. We tested results of the method for different modalities shown in the Table 6.

	modality
	flair
	T1
	T1c
	T2

Table 6: Tested modalities

We also tested different settings of searching window (w) in Farneback algorithm and different size of step between the points (s) calculated from the flow in the step Computation of the corresponding points (chapter 9.6.3). Everything was tested on the flair modality. Table 7 shows tested configurations of the method.

	searching window size (w)	step between points (s)
	15	16
	29	16
	39	16
	15	8
	15	32

Table 7: Different configurations of the method

9.6.5.1 Dataset

Non-rigid registration was tested on BRATS 2015 dataset. However not all patients in the dataset have got more examinations, so we choose from the dataset only patients with more than one examination. Together it is 20 patients and 95 examinations. On average 4,75 examination for one patient – some patients have only 2 examinations but other have more.

9.6.5.2 Testing flow and evaluation

Testing flow is visualized using sequence diagram in the Figure 85. We process all patients from the dataset in the loop. For every patient we preprocess the data using preprocessing method from the chapter 9.3. Then we push all examinations from one patient to the non-rigid registration algorithm and it computes corresponding points for all examinations.

For the evaluation it is necessary to transform volumes non-rigidly. For the transformation we decided to use similar process as image morphing described in the chapter 9.6.4.3. Actually, the process is the same except for the last step. We do not create new image instead of that we transform triangles from the old slice to the triangles from the actual slice. Finally, we compare slices using the same method as in rigid registration testing (explained in the chapter 9.5.8.3). We compute average correlation, intersection and Bhattacharyya distance for:

- histograms of slices from old examination and actual examination,
- histogram of slices from transformed (non-rigidly) old examination and actual examination.

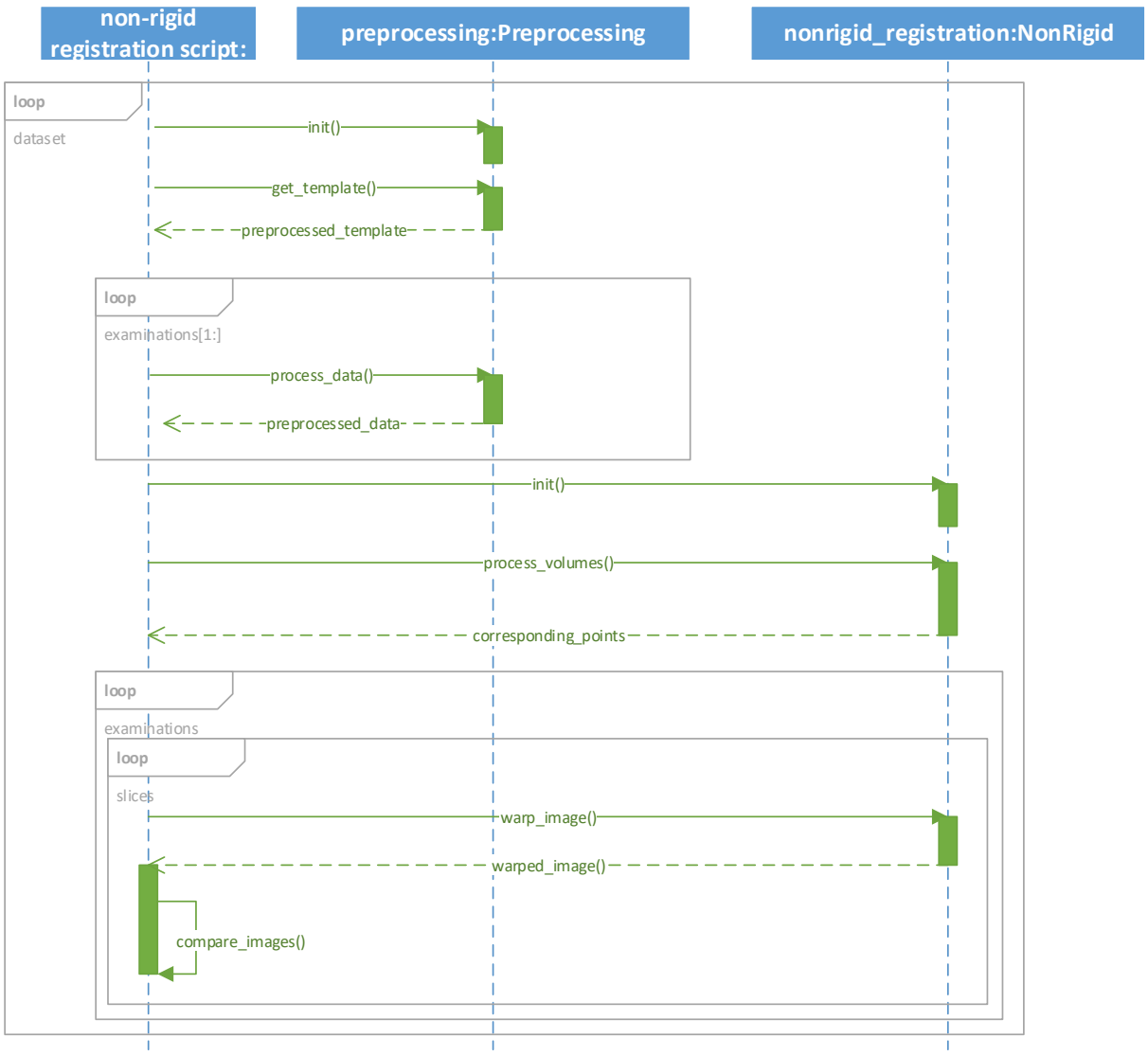


Figure 85: Flow of the testing process (UML notation)

9.6.6 Results

We have tested non-rigid registration on the dataset consists of 20 patients with more than one examination. While we evaluated the results slice by slice, not for whole volumes, it is not possible to visualize all results in one graph because it is a lot of data. We used several visualizations to compare:

- results for different modalities and
- different configurations of the method.

9.6.6.1 Results for different modalities

In the Figure 86 we can see ratio between cases when the correlation of the transformed (registered) old slices and actual slices was higher (lower) than correlation of the input old slices data and actual slices.

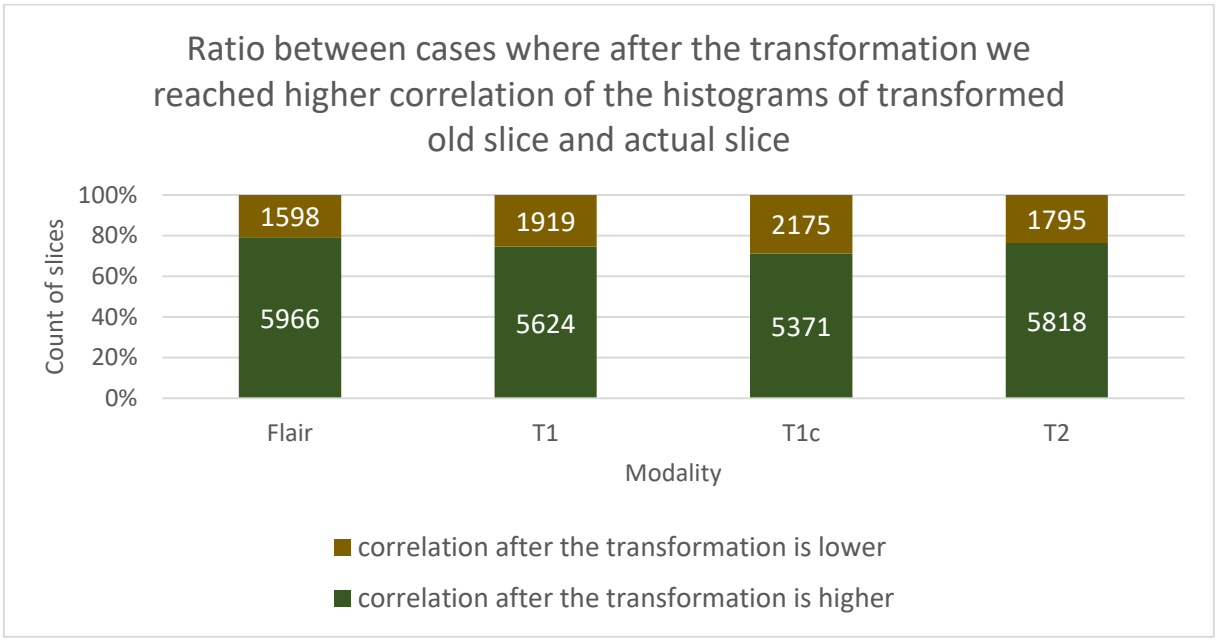


Figure 86: Ratio between cases where after the transformation we reached higher histogram correlation of the transformed old slice with actual slices

For the different tested modalities, we have compared average values of the histogram correlations between old slice and actual slice versus transformed old slice and actual slice. The results are presented in the Table 8 and visualized in the Figure 87.

	actual and old original	actual and transformed old original
Flair	0.719322477	0.744422397
T1	0.644726762	0.667428015
T1c	0.700108284	0.714934625
T2	0.713350622	0.731279275

Table 8: Comparison of average histogram correlations for different modalities

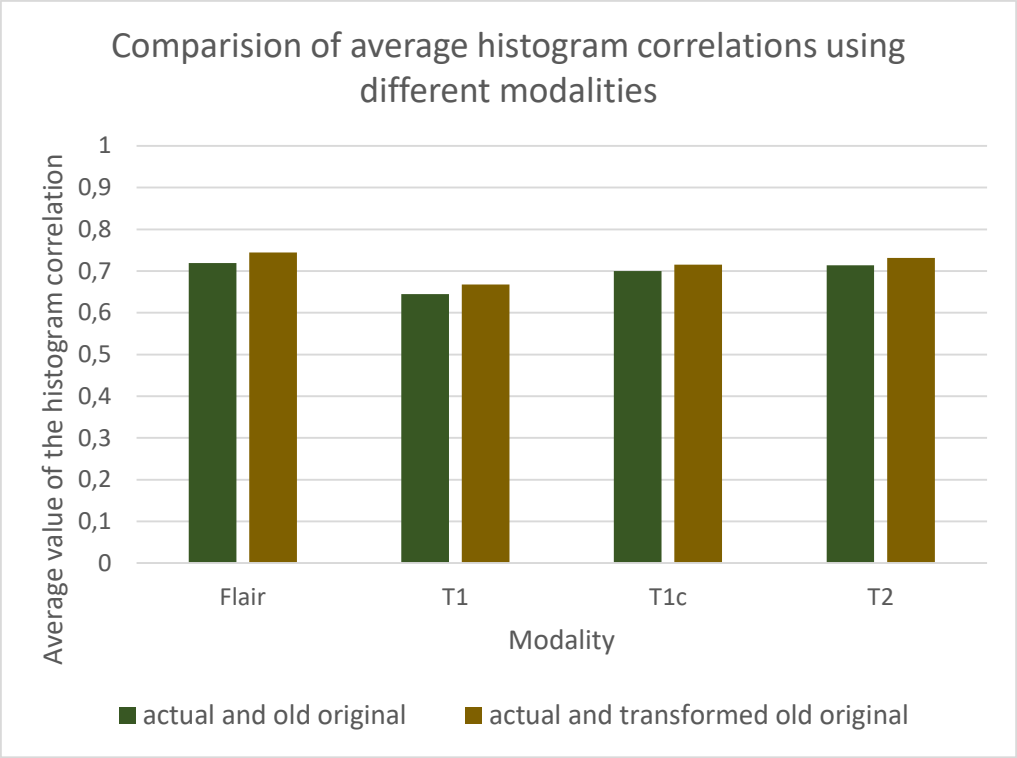


Figure 87: Comparison of average histogram correlations for different modalities

We have computed minimum, bottom quartile, median, top quartile and maximum of the histogram intersections between transformed old slice and actual slice. Results are visualized in the Figure 88.

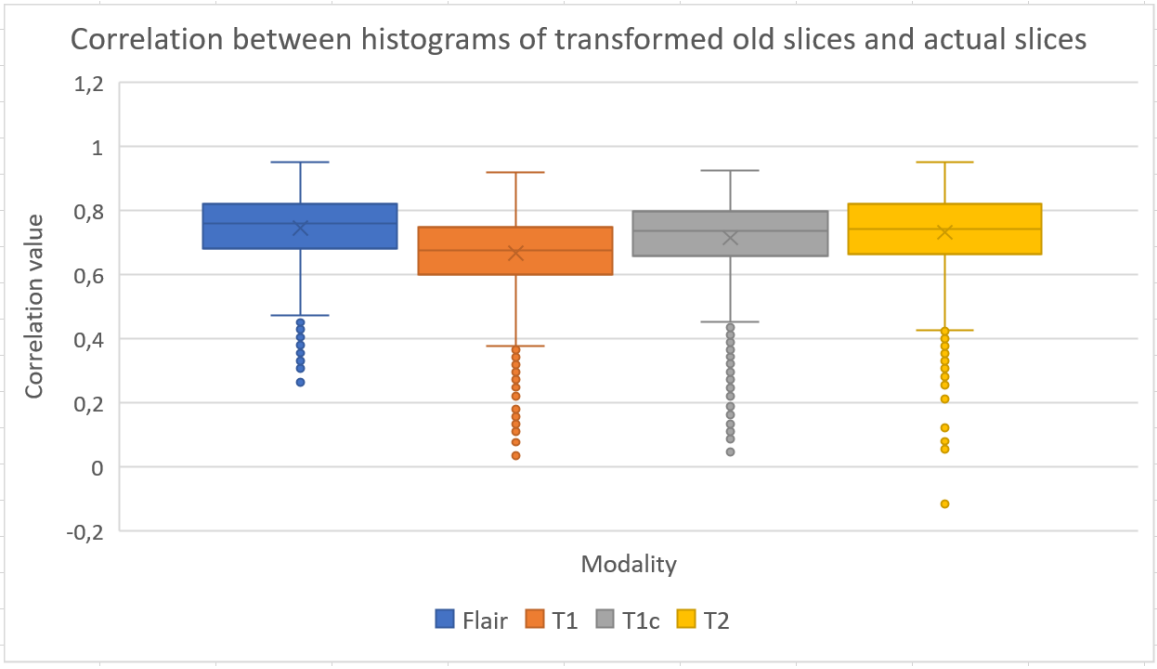


Figure 88: Histogram correlation of transformed old slice and actual slice

For the different tested modalities, we have compared average values of the histogram intersection between old slice and actual slice versus transformed old slice and actual slice. The results are presented in the Table 9 and visualized in the Figure 89.

	actual and old original	actual and transformed old original
Flair	425.8967622	443.1579404
T1	391.8223372	412.8192378
T1c	418.0144272	431.2574375
T2	440.5532608	453.7109507

Table 9: Comparison of average histogram intersections for different modalities

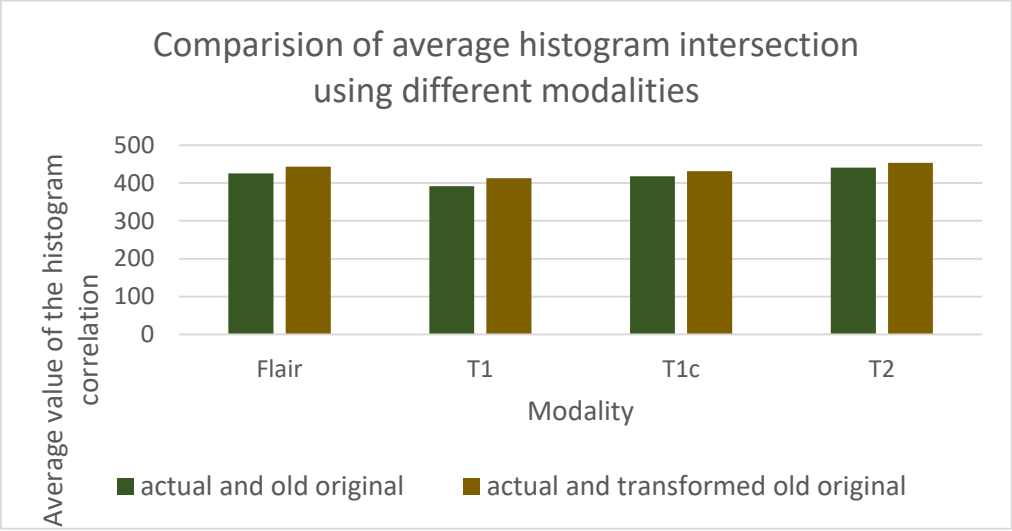


Figure 89: Comparison of average histogram intersections for different modalities

We have computed minimum, bottom quartile, median, top quartile and maximum of the histogram intersections between transformed old slice and actual slice. Results are visualized in the Figure 90.

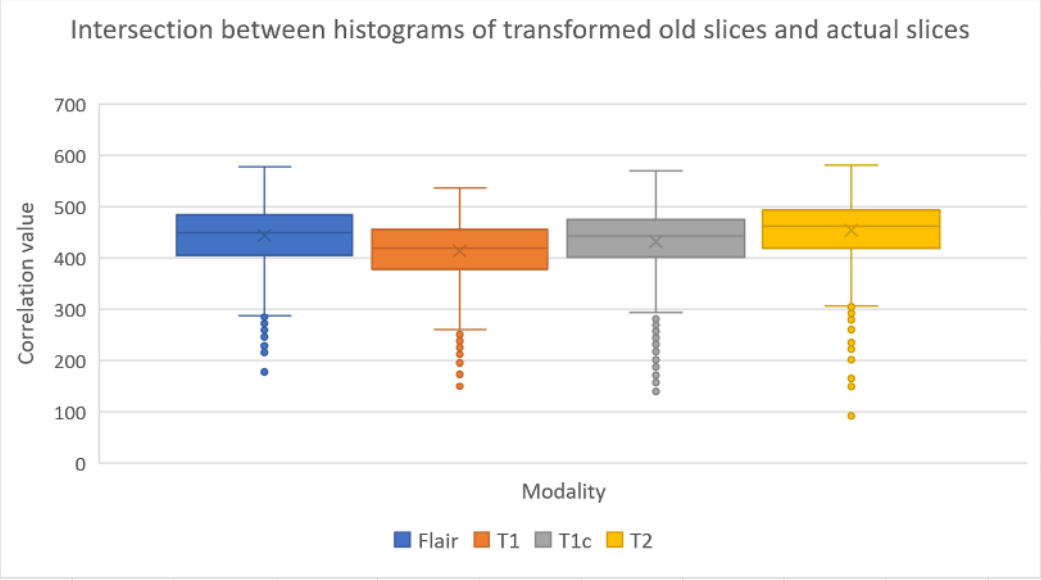


Figure 90: Histogram correlation of transformed old slice and actual slice

For the different tested modalities, we have compared average values of the histogram Bhattacharyya distance between old slice and actual slice versus transformed old slice and actual slice. The results are presented in the Table 10 and visualized in the Figure 91.

	actual and old original	**actual and transformed old original**
Flair	0.483100397	0.447198571
T1	0.53928495	0.497564994
T1c	0.491556665	0.46384179
T2	0.456074306	0.432324678

Table 10: Comparison of average Bhattacharyya distances for different modalities

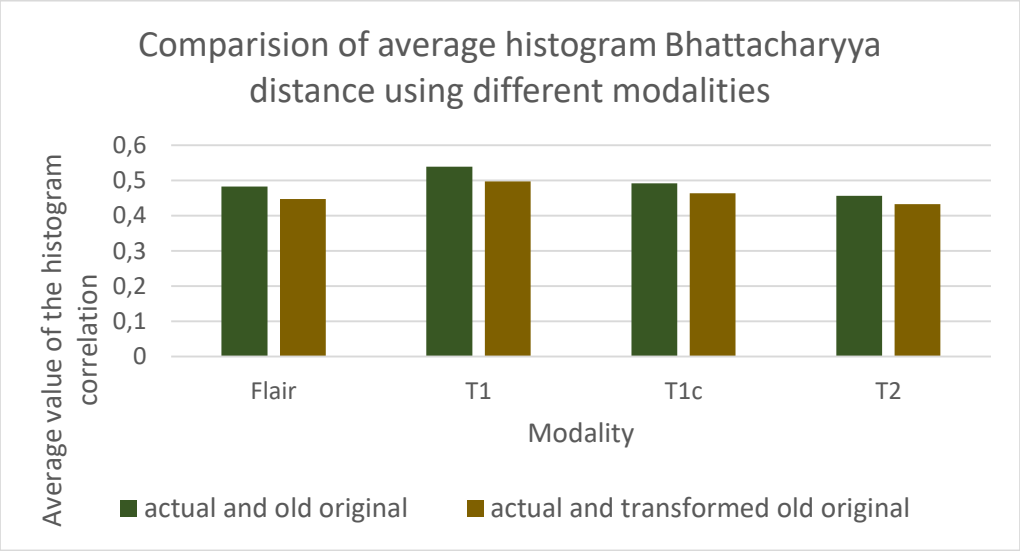


Figure 91: Comparison of average Bhattacharyya distances for different modalities

We have computed minimum, bottom quartile, median, top quartile and maximum of the histogram Bhattacharyya distances between transformed old slice and actual slice. Results are visualized in the Figure 92.

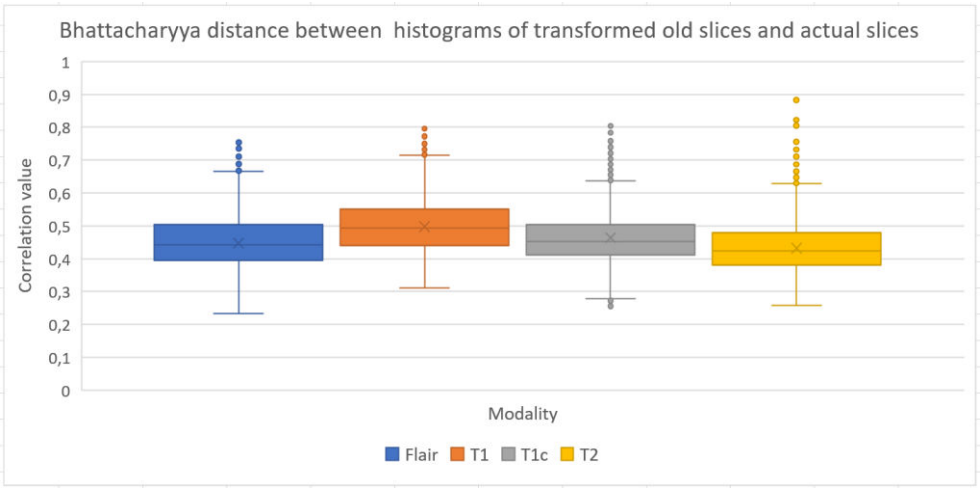


Figure 92: Histogram correlation of transformed old slice and actual slice

9.6.6.2 Results for different configurations of the method

In the Figure 93 we can see ratio between cases when the correlation of the transformed (registered) old slices and actual slices was higher (lower) than correlation of the input old slices data and actual slices.

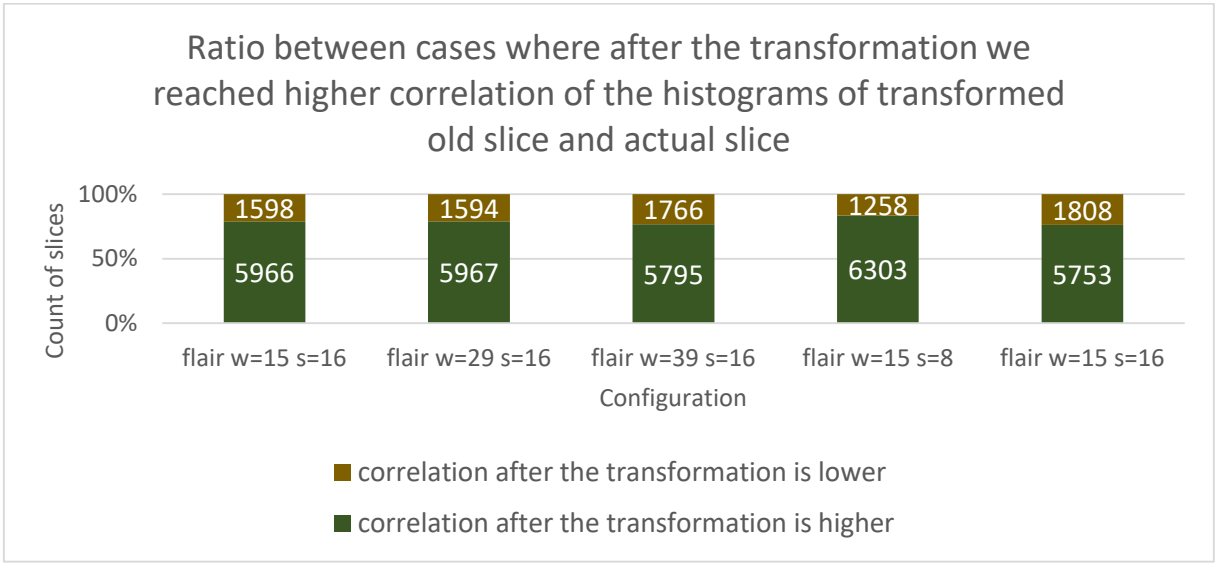


Figure 93: Ratio between cases where after the transformation we reached higher histogram correlation of the transformed old slice with actual slices

For the different tested configurations, we have compared average values of the histogram correlations between old slice and actual slice versus transformed old slice and actual slice. The results are presented in the Table 11 and visualized in the Figure 94.

	actual and old original	actual and transformed old original
flair w=15 s=16	0.719322477	0.744422397
flair w=29 s=16	0.719347649	0.740349981
flair w=39 s=16	0.719347649	0.737536019
flair w=15 s=8	0.719347649	0.745301948
flair w=15 s=16	0.719347649	0.742616476

Table 11: Comparison of average histogram correlations for different configurations

In the Table 11 and in the Figure 94 we can see that different configurations returns very similar results, but we also try to compute minimum, bottom quartile, median, top quartile and maximum of the histogram correlation between transformed old slice and actual slice. Results are visualized in the Figure 95. This confirms the previous findings, so we do not visualize intersection metric and Bhattacharyya matric in this evaluation.

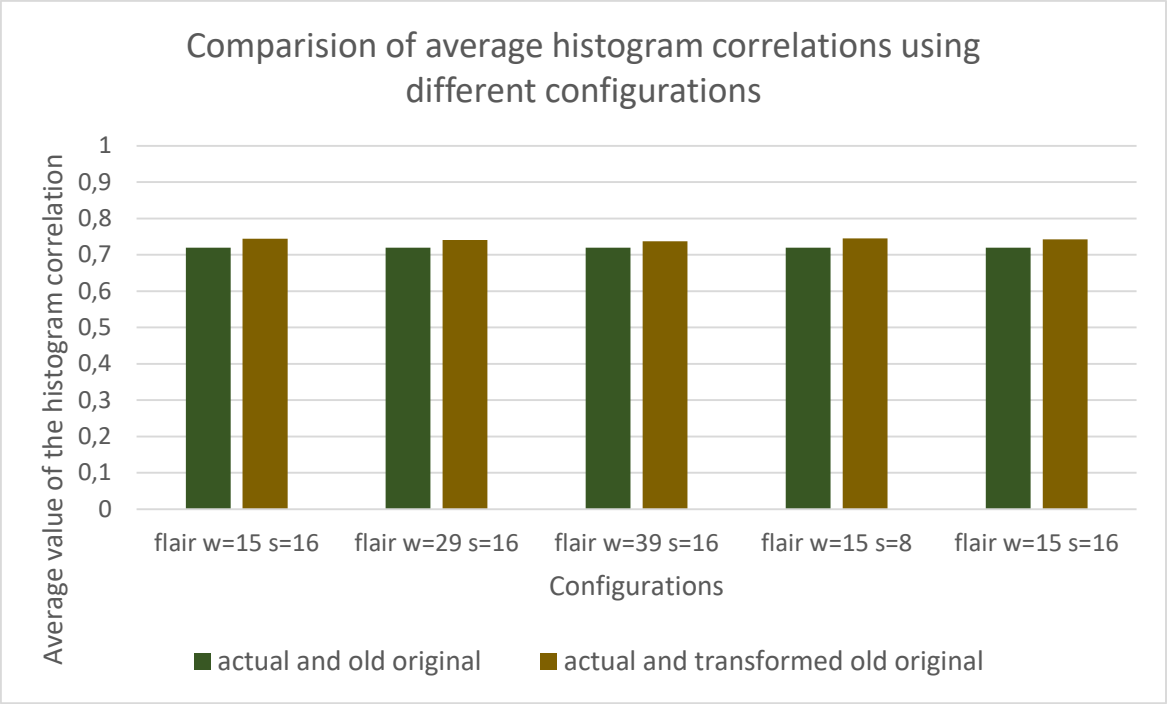


Figure 94: Comparison of average histogram correlations for different configurations

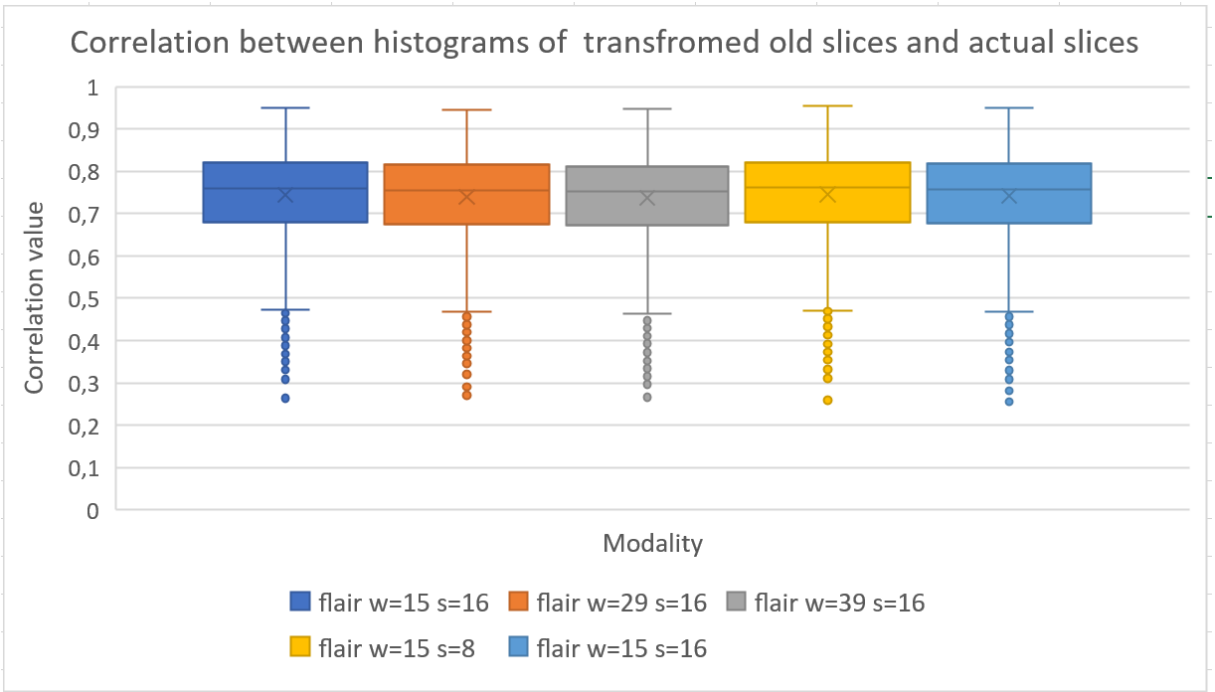


Figure 95: Histogram correlation of transformed old slice and actual slice

We have also created the histogram of correlations for the basic configuration of the method. In the Figure 96 we can see comparison of histogram correlation distribution for old slice and actual slice versus transformed old slice and actual slice.

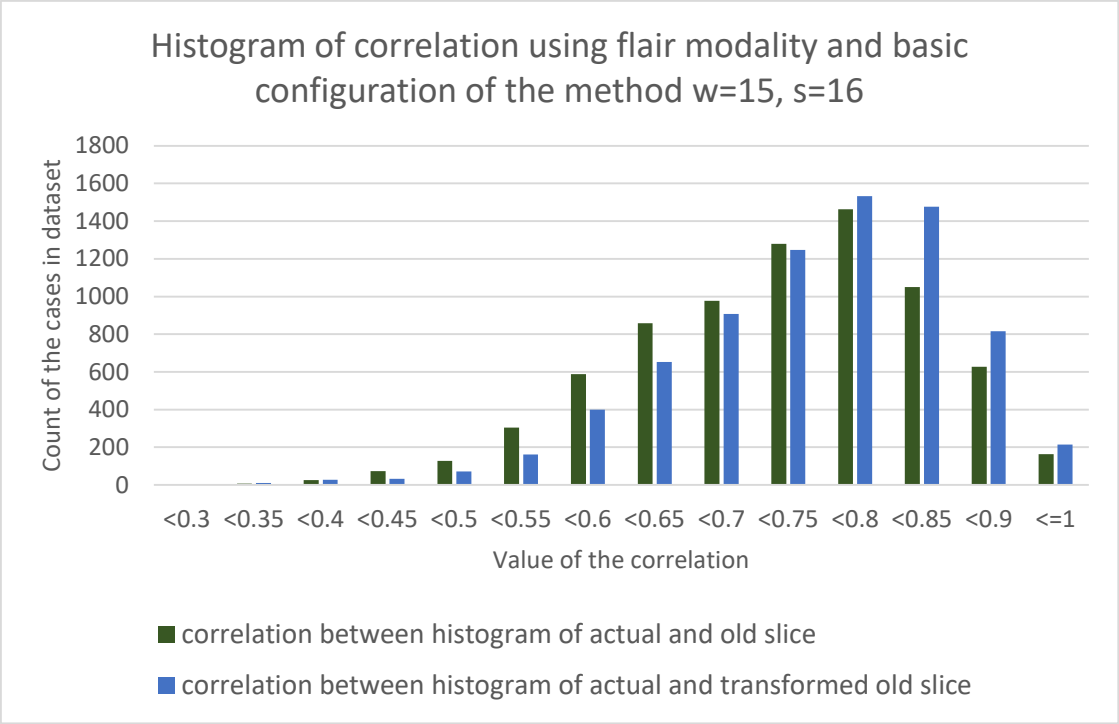


Figure 96: Histogram of correlations (flair w=15, s=16)

9.6.7 Implementation

Non-rigid registration can process 3D volumes, but it computes optical flow slice by slice. Input for the method are all examination of one patients = 3D data. Algorithms used in our proposed method of non-rigid registration are from OpenCV library. Also NumPy library is used to store data and perform simple operations.

9.6.7.1 Structure of the source code presented by a class diagram

NonRigid class is organized in *logic* package and used by the *main* function as is shown in the Figure 97. *NonRigid class* has one important parameter:

- *registration_ type* – Farneback or Lucas-Kanade method

And also implements several methods:

- *process_volumes()* – processes volumes form examinations from one patient
- *process_frames_farneback()* – processes all corresponding slices of the examination using Farneback optical flow algorithm
- *process_frames_lk()* – processes all corresponding slices of the examination using Lukas-Kanade optical flow algorithm
- *warp_images()* – warps images for the evaluation
- *morph_images()* – creates image for image morphing

- *delaunay_triangulation()* – computes triangles from the points
- *apply_transfrom()* – performs transformation between input triangles

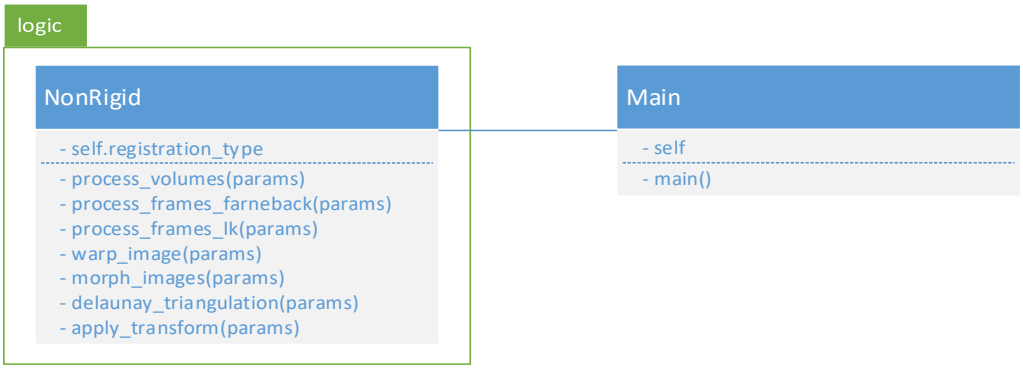


Figure 97: Class diagram of non-rigid registration (UML notation)

9.6.7.2 Interaction of the objects presented by a sequence diagram

In the Figure 98 is shown sequence diagram of the non-rigid registration. *Nonrigid* is used by the *main*. First we have to initialize *NonRigid* and set configurations of the method. Then the processing of the volumes can start. Volumes are processed slice by slice using Farneback optical flow method or Lukas-Kanade optical flow method – depending on the configuration. When corresponding points are found, we can morph images for the visualization of the tumor changes.

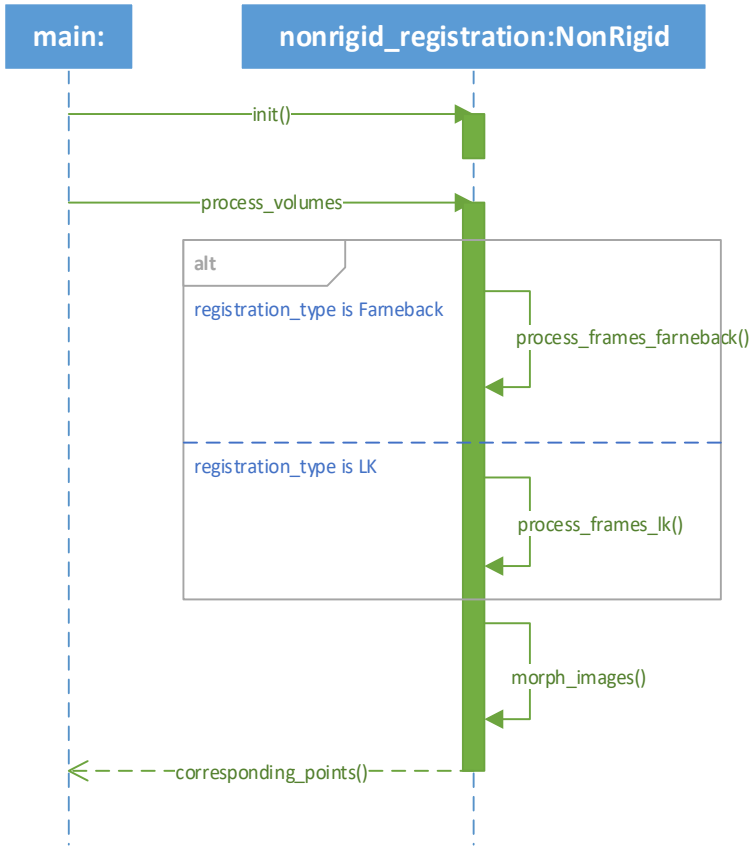


Figure 98: Sequence diagram of non-rigid registration (UML notation)

9.6.8 Discussion

Results presented in tables Table 8 - Table 10 and figures Figure 86 - Figure 92 show that modality can affect the results of the proposed method. We reached the best results using flair modality and just behind it T2 modality. On the other side when we used T1 modality results were the worst. Number of cases where we reached higher correlation after the transformation of the old slice is not such influences by the modality. However, in the Figure 86 we can see that the best results was reached using flair modality.

The results presented in the Table 11, Figure 94 and Figure 95 show that configuration of the method cannot improve the correlation between the old slice + actual slice and transformed old slice + actual slice. However, in the Figure 93 we can see that when we use smaller step for the computation of the corresponding points, more old slices after the transformation reach higher value of the correlation witch actual slices than other configurations.

In the Figure 96 we can see that using basic configuration of the method correlation between the old frame and actual frame is higher after the transformation of the old frame, thus method work well for our problem.

9.7 Visualization tool

We have created simple tool to visualize 3D volumes. The tool can process 3D data format and visualize brain MRI scans in 2D and 3D view. It also provides simple interaction. We can switch on/off the visualization of the tumor segmentation in both 2D and 3D views. Preparation of the visualization consists of several steps as is shown in the Figure 99.

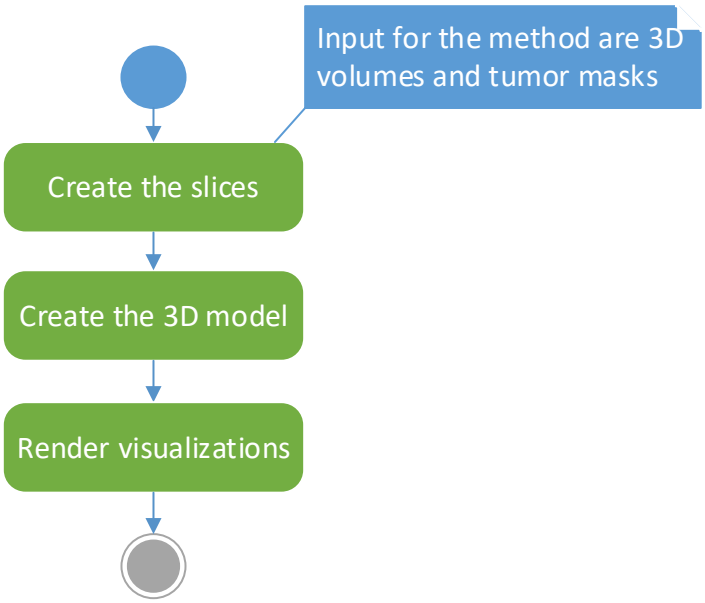


Figure 99: Basic steps of the visualization

9.7.1 Creating the slices

In the first step, we have to create slices to visualize MRI volumes in 2D. All medical software products usually use 3 different 2D views of 3D data based on the direction where we are looking from:

- axial,
- coronal and
- sagittal.

Not all data have got good resolution in all directions, but we decided to show all views. In the Figure 100 is shown an example of 2D slices in all directions.

We are creating slices directly from the volume. When the user requests some slice (by interaction in the tool) then we only scroll through them. We are using special matrixes, different for every direction, to get slices from the volume:

- Matrix for axial view: $m = \begin{vmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{vmatrix}$, where x, y, z is position of the volume center.
- Matrix for coronal view: $m = \begin{vmatrix} 1 & 0 & 0 & x \\ 0 & 0 & 1 & y \\ 0 & 1 & 0 & z \\ 0 & 0 & 0 & 1 \end{vmatrix}$, where x, y, z is position of the volume center.
- Matrix for sagittal view: $m = \begin{vmatrix} 0 & 0 & -1 & x \\ 1 & 0 & 0 & y \\ 0 & 1 & 0 & z \\ 0 & 0 & 0 & 1 \end{vmatrix}$, where x, y, z is position of the volume center.

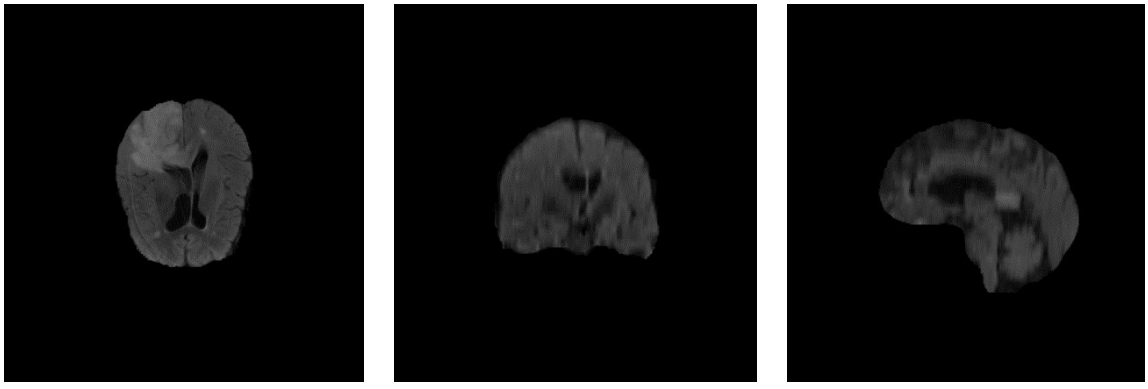


Figure 100: Axial, coronal and sagittal view of the brain MRI scan

9.7.2 Creating 3D model

If we want to create the 3D model of the brain and separately of the tumor we have to segment those parts from the volume to the two independent arrays. Segmentation of the whole brain is simple

because background is black and everything other is brain, so we can simply use threshold greater than zero to segment foreground – brain. To segment tumor from the volume we use mask of the tumors from the datasets. Masks from the BRATS dataset contains 5 different labels, so we have to unit them if we want to segment whole tumor.

Then we can use those segments as an input for the ray casting method which can create 3D model of the brain and also of the tumor. Different settings are implemented in the visualization tool to visualize whole brain, tumor only or location of the tumor in the brain. An example of the model created by ray casting method tool is shown in the Figure 101, where we can see health brain with 7% opacity and tumor with none opacity.

Ray casting algorithm is a simple algorithm which can generate 3D model from the volume. The algorithm is based on the fact that we send a ray from some point. The ray crosses the volume at some point and then the value of the point is rendered on the canvas.

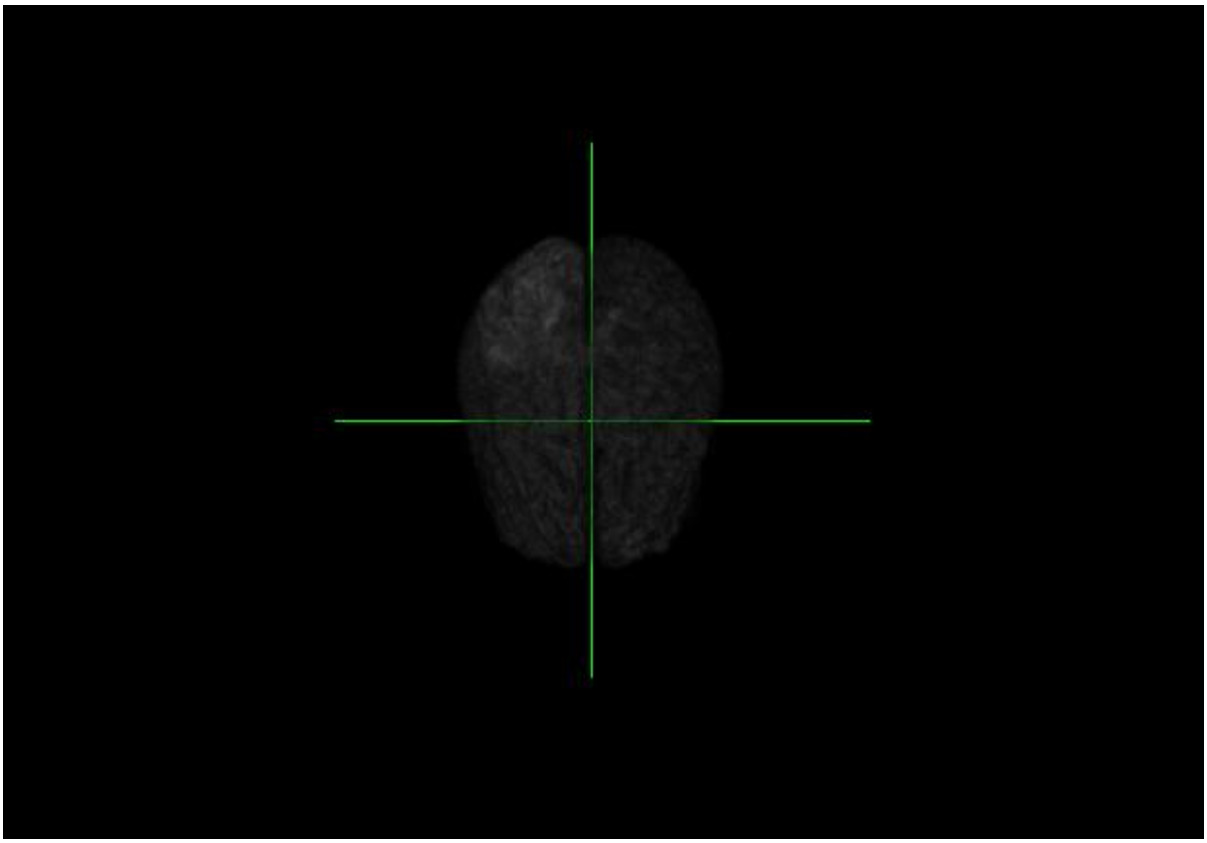


Figure 101: 3D model of the brain

9.7.3 Rendering of the visualizations

Final step is initialization of the GUI based on the settings from the previous steps. Initial GUI of our desktop application consists of four canvases and side panel with some setting options as is shown in the Figure 102. Three canvases contain axial, coronal and sagittal 2D slices. Fourth canvas contains 3D model of the brain and tumor and three axes which show where in the brain are slices on the canvases located.

User can translate and zoom the slices. When user scroll through the slices, axes in the 3D model change depending on the direction of scrolling. Application window also contain two histograms – one for health brain structures and second one for tumor structures.

In addition, application allow to display masks of the tumor on the slice views as is shown in the Figure 103. User can also highlight tumor in the 3D model by choosing different settings on the side panel. Figure 104 shows different views of the highlighted tumor in the 3D model.

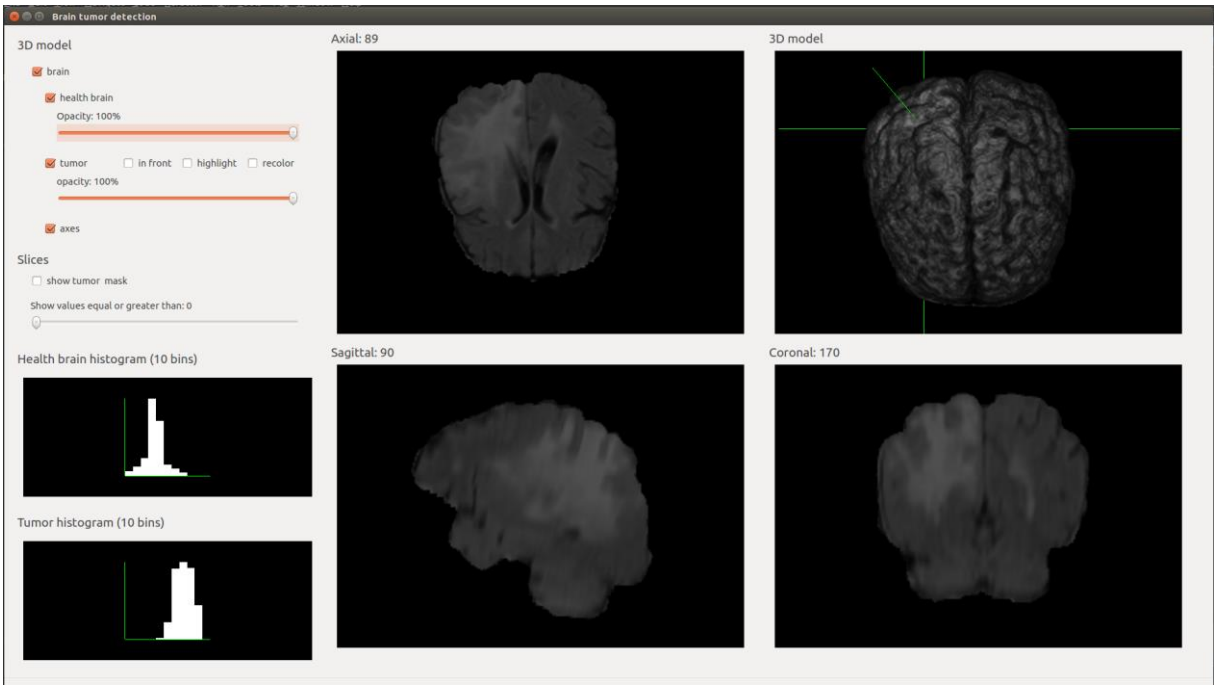


Figure 102: Initial window of the application

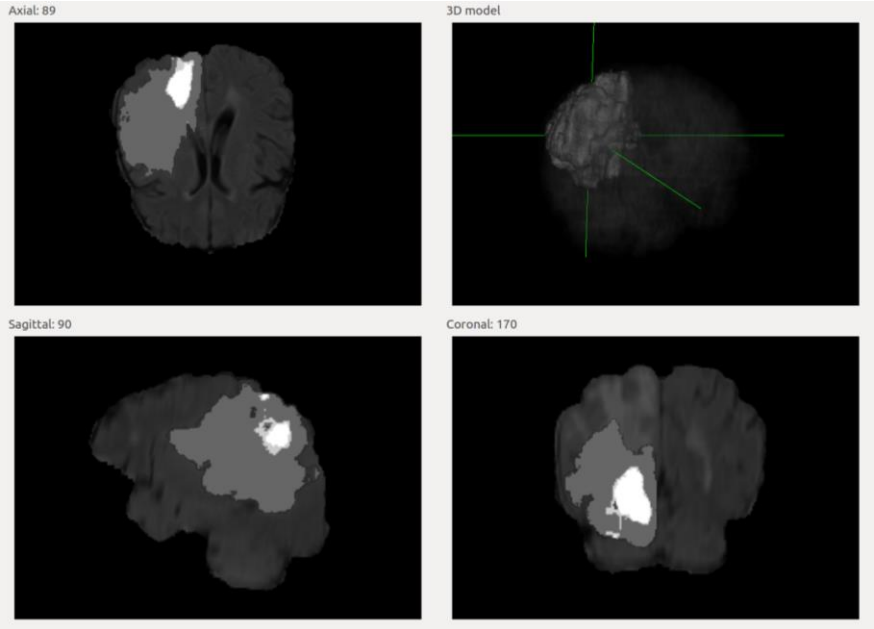


Figure 103: Visualization of the segmentation mask

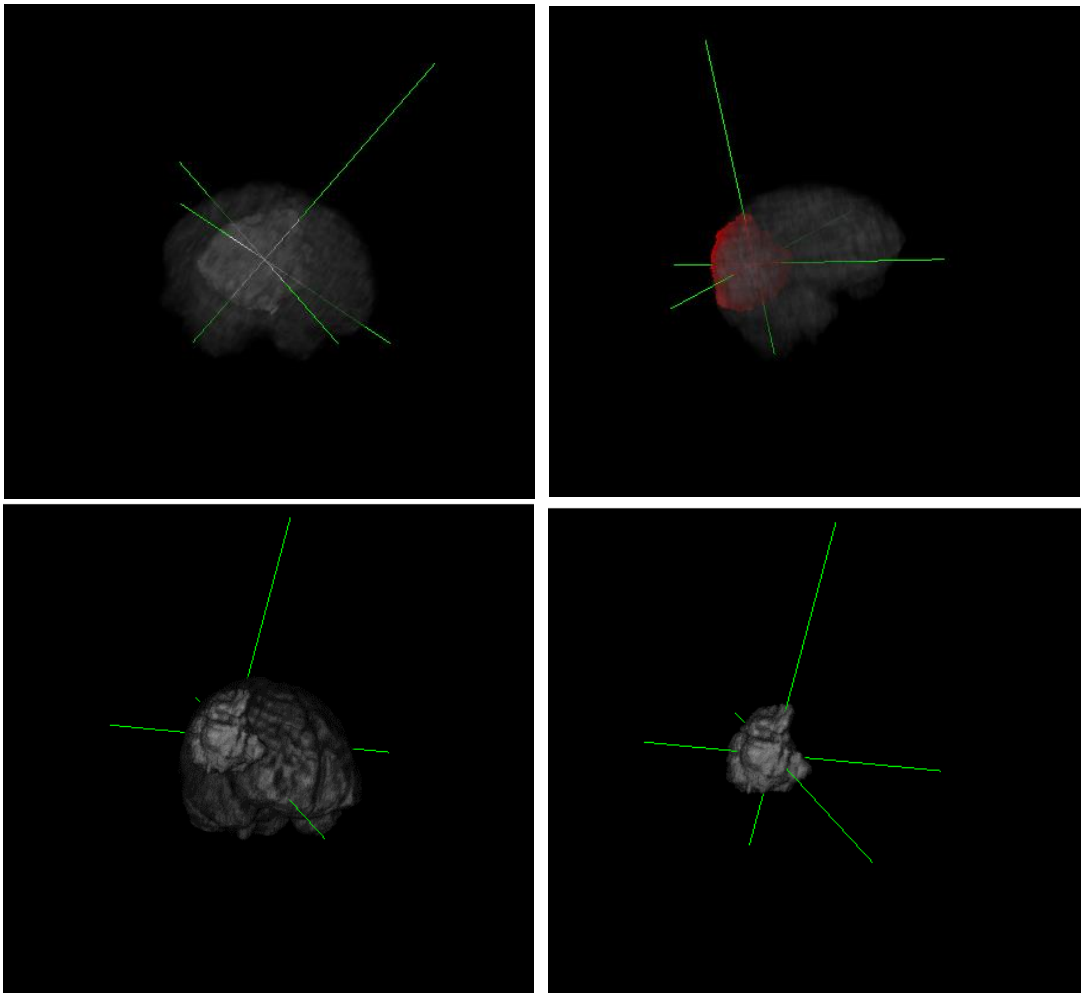


Figure 104: Different ways to highlight the tumor in the 3D model

9.7.4 Implementation

We are using VTK library and its functions to create the windows and also interaction. It is only a simple library – it can create only one independent window when we run the program. Thus, we connected VTK with PyQt. Thanks this we were able to create more sophisticated layout of the GUI using special widgets which permit to add simple VTK windows.

9.7.4.1 Structure of the source code presented in class diagram

Visualization is organized to the more classes in the *visualization* package. And also uses some functions from *Main class*. Class diagram of the visualization is shown in the Figure 105.

Main class does not have any important parameters but implements several methods for user interaction:

- *showTumorVolume()* – shows tumor model in the canvas
- *showBrainVolume()* – shows brain model in the canvas
- *setHealthBrainOpacity()* – changes opacity of the brain in 3D model

- *setTumorOpacity()* – changes opacity of the tumor in 3D model
- *tumorToFront()* – moves tumor in front of the brain in 3D model
- *highlightTumor()* – changes brightness of the tumor in 3D model
- *redolorTumor()* – changes collor of the tumor in 3D model
- *showTumorMask()* – shows mask of the tumor in the slices
- *showAxes()* – show axes in 3D model canvas

GUI class implements two methods:

- *setupUi()*, *retranslateUi()* which create GUI – window of the application:

Axe class implements one method:

- *paintLine()* – method paint line into the canvas of 3D model depending on the numbers of visualized slices

Histogram class implements one method:

- *makeHistogram()* – computes histogram of input data and paint it to histogram canvas

Volume class has two important parameters and implements several methods:

- *tumor* – data to create 3D model of the tumor
- *brain* – data to create 3D model of the brain
- *createTumorActor()* - creates 3D model of the tumor
- *createBrainActor()* – creates 3D model of the brain
- *volumeRender()* – renders volume to the canvas

Slices class implements several methods:

- getBrainSlices() - creates slices from 3D data
- *getTumorSlices()* – creates slices of the masks of tumors from 3D data
- *setInteraction()* and *addMouseMoveCallback()* - handle additional interaction to scroll thought slices

Figure 105: Class diagram of visualization (UML notation)

9.7.4.2 Interaction of the objects presented by the sequence diagram

In the Figure 106 is shown sequence diagram of the visualization. When user run the program main function starts and process all data using the method. Then it is time for the visualization. At first we create GUI elements for the user interface (window, canvases, labels, buttons). In the next step we crate 3D model of the brain and tumor for 3D visualization of the volume. We also create slices in axial, coronal and sagittal direction for both, brain and tumor again. In this step we have to set interaction too. It allows user scroll through the slices in the GUI. Then we can create axes for the canvas with 3D model visualization. The axes show positions of the slices in the brain. Finally, we create histogram of health brain and histogram of tumor. The result is a GUI returned to the user.

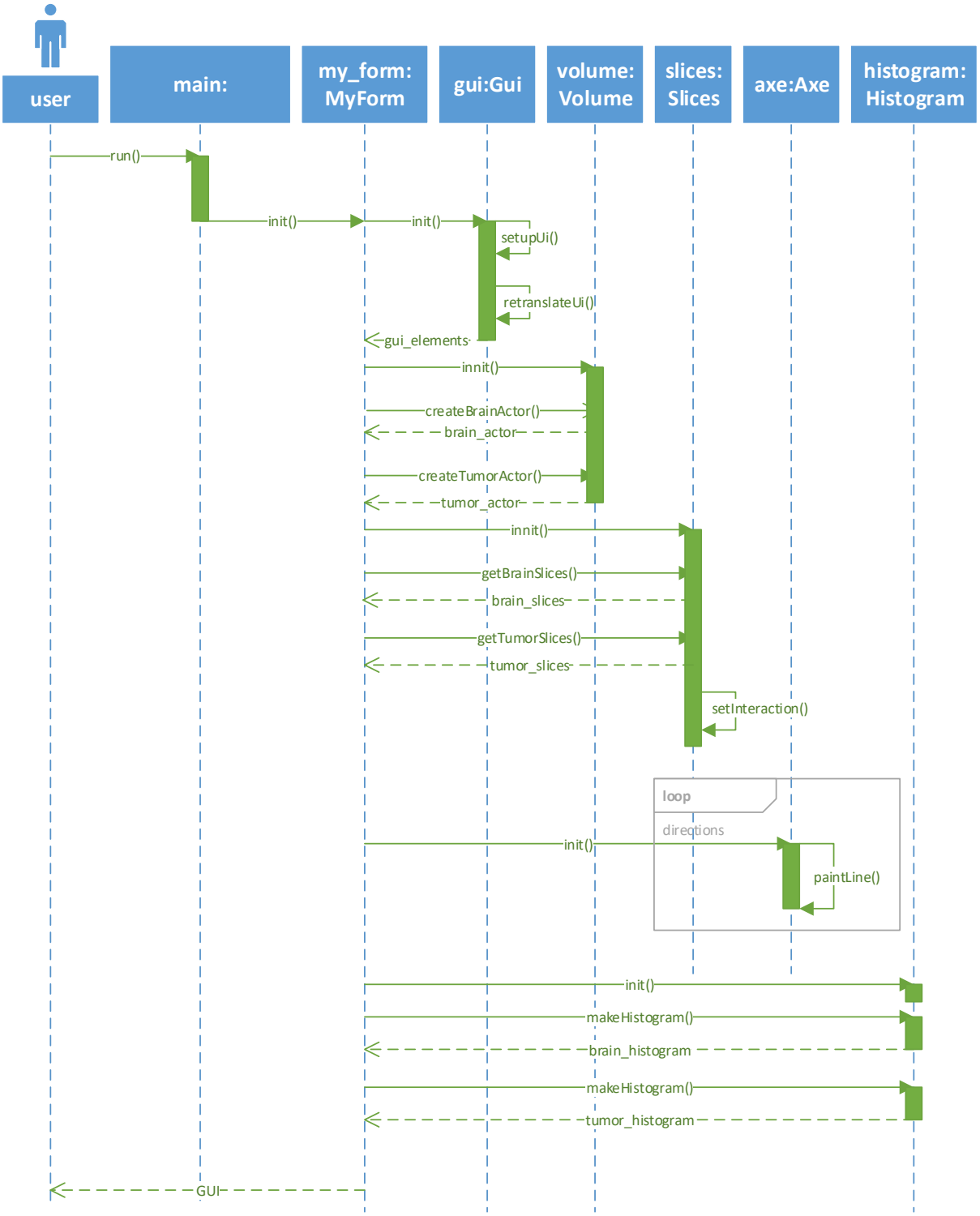


Figure 106: Sequence diagram of visualization (UML notation)

9.8 Implementation in general

9.8.1 Used development tools

Sources are written in Python 3.5 programming language and several libraries are used for proposed methods. Preprocessing use SimpleITK and NumPy– images and volumes are represented as arrays and performed operations are fast. Tested segmentation method based on convolutional neural networks uses Tensorflow and Keras libraries. Both registration methods use algorithms from OpenCV library. Finally, visualization tool is implemented using VTK, to render visualizations, and PyQT with plugin for VTK, to provide GUI. Matplotlib and Pandas are used for evaluation.

9.8.2 Structure of the source code presented by a class diagram

Source code is organized to the classes and packages as is shown in the Figure 107. Main class contains main function and basic functions for visualization – to set visibilities, opacities of the brain and tumor and also to visualize axes.

Logic package contains classes for preprocessing, rigid and non-rigid registration and also for reading data from disk. Some source codes are also used in the scripts for testing and dataset generating, but they are not involved in the class diagram – there is only collapsed *scripts package*. *Logic package* consists of these classes:

- *Reading class* is used by the main function, but mainly it is used by the scripts for testing and dataset generating. It reads one examination of the patient stored on the disk in specific hierarchy.

- *Preprocessing class* preprocess the volumes and was explained in detail in the chapter 9.3. Preprocessing is used in the main function to preprocess volumes and also in mentioned scripts.

- *Rigid class* implements functions for rigid registration of the slices and was explained in the chapter 9.5. It was not necessary to use rigid registration in main, because we were working with registered data, so it is used only by the scripts for testing.

- *NonRigid class* consists of methods for non-rigid registration of the volumes. It can process whole volume, but it works slice by slice. It is explained in detail in the chapter 9.6.

Object package contains only two classes which represent some objects:

- *Examination class* stores all data from one patient examination – name, modalities and results of non-rigid registration.

- *Volume class* stores volumetric data and basic information from the data such as origin, spacing, size and direction.

Visualization package contains all important classes for visualization tool – GUI. They are explained in detail in chapter 9.7, but in general:

- *Gui class* generates basic layout of the visualization tool.

- *Slices class* implements function for generating 2D slices from 3D volume in axial, sagittal and coronal view and also functions for the interaction which allows to scroll through slices.
- *Volume class* implements function for generating 3D model of the brain from volumetric data.
- *Histogram class* computes and paints the visualization of the histogram from the data (for the brain part and tumor part)
- *Axe class* contains methods for visualizing and moving axes in the 3D view depending on number of slices displayed in all 2D slice views.

Figure 107: Class diagram in general (UML notation)

9.8.3 Interaction of the objects presented by a sequence diagram

In the Figure 108 is shown sequence diagram. After the reading the examination of the patient, data are preprocessed using preprocessing methods mentioned in the chapter 9.5. First examination is used as target and all next are reference examiations. We have got rigidly registered data, but if we did not

have then all examination would be registered using rigid registration methods as is explained in the chapter 9.5. Then to track the tumor changes, we have to register data non rigidly – find corresponding points between the examinations. Thus, we use methods of the non-rigid registration mentioned in chapter 9.6. Finally, we can visualize data using visualization tool as is explained in the chapter 9.7. This return the GUI to the user.

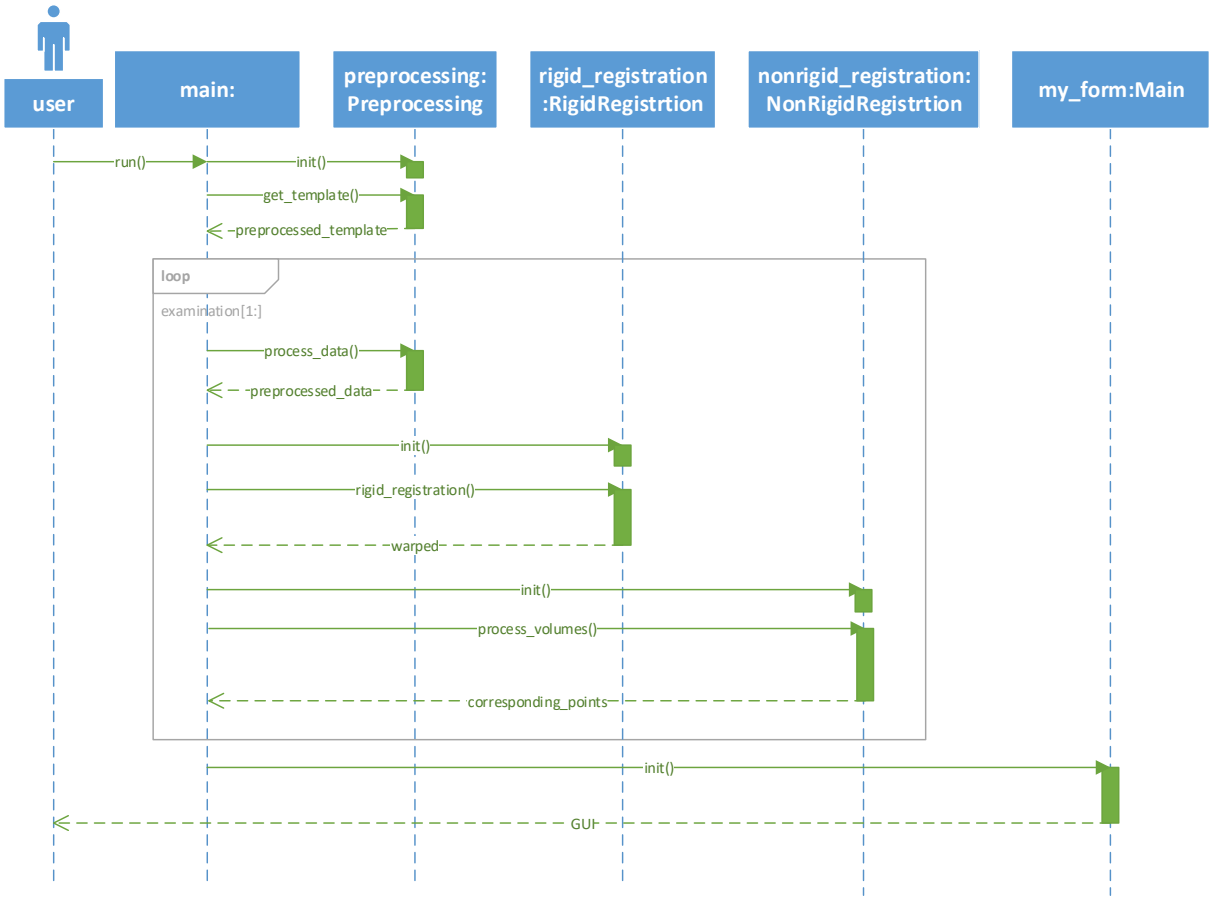


Figure 108: Sequence diagram in general

10 Conclusion

The main goal of the work was to develop methods appropriate for tracking brain and tumor changes caused by the tumor growth. To track the tumor changes, we have to segment tumor, register different examination from one patient and register changes. We proposed a method for tumor tracking. It consists from five steps.

First step is preprocessing of the data. This is important, because examination from one patient have different ranges and different brightness. We used histogram matching algorithm to balance the brightness of the examinations from one patient and the results are very promising.

Second and important step is tumor segmentation method. We have to segment tumor to be able to track tumor changes and changes of the health brain. In our approach we decided to re-implement one of the state of the art method. Selected method was based on cascaded convolutional neural networks. However, during the testing we did not reach satisfactorily results, so in the next steps we decide to use tumor masks from the datasets. Boundaries of the tumor are crucial for correctness of the rigid registration and can be very helpful for non-rigid registration.

Third step is rigid registration of the volumes from different examinations but from one patient. During the rigid registration we have to compute without the tumor, because as time flows, the tumor changes. We used basic flow of the registration which consists of several steps: creating mask for registration, finding key points, computing feature vectors, finding matches between the vectors, filtration of the matches, finding matrix for transformation and transforming data finally. Proposed method can process 2D slices. We tested the method on the dataset which contains registered and warped 2D slices created from Siemens dataset. We tested different configurations of the method, using different segmentation masks, different key point detectors and feature extractors. Results are promising. The best results we reached using registration mask of the health brain parts, ORB key point detector and descriptor.

Enlargement of the tumor causes also changes of the health structures of the brain, because they are threaded out by the tumor. This causes irregular changes of the brain so this is perfect task for fourth step – non-rigid registration. We proposed method based on the optical flow for this problem. We tested two different algorithms: Farneback and Lukas-Kanade, but Farneback algorithm worked better against the Lukas-Kanade, because Farneback returns dense flow. We tested the method on the BRATS 2015 dataset using only patients with more than one examination. During the valuation we had to transform one examination non-rigidly to other one to compute histogram correlations between the examinations. We used Delaunay triangulation and affine transformation. We tested the impact of different modalities and different input parameter of the Farneback algorithm. We reached the best results using Flair modality and comparatively T2. However, different configuration of the Farneback algorithm did not have a big impact on the non-rigid registration. Only usage of the smaller step, during the computation of the corresponding points from the flow, helped to get higher correlation after the transformation in more cases than other configurations.

Final step is visualization. For the visualization of the tumor changes we implemented three different methods: painting the tracks, HSV image of the changes and image morphing. We have also created a simple visualization tool where user can see volumetric data in axial, coronal and sagittal direction and it also cab create 3D model of the brain and tumor using ray casting algorithm.

References

[1] ACHANTA, R. et al. SLIC Superpixels. In *InfoScience EPFL* [online]. 2010. [cit. 2015-06-22]. . Dostupné na internete: <http://infoscience.epfl.ch/record/149300>.

[2] BAJCSY, R. - KOVAČIČ, S. Multiresolution elastic matching. In *Computer Vision, Graphics, and Image Processing* [online]. 1989. Vol. 46, no. 1, s. 1–21. [cit. 2017-05-10]. . Dostupné na internete: <http://linkinghub.elsevier.com/retrieve/pii/S0734189X89800143>.

[3] BAUER, S. et al. Fully Automatic Segmentation of Brain Tumor Images Using Support Vector Machine Classification in Combination with Hierarchical Conditional Random Field Regularization. In [online]. 2011. s. 354–361. Dostupné na internete: <http://link.springer.com/10.1007/978-3-642-23626-6_44>.

[4] BAY, H. et al. Surf: Speeded up robust features. In *Computer vision–ECCV* . 2006. s. 404–417. .

[5] BEUCHER, S. - LANTUEJOUL, C. [online]. .1979. Dostupné na internete: <http://www.citeulike.org/group/7252/article/4083187>.

[6] BOYKOV, Y.Y. - JOLLY, M.-P. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001* [online]. 2001. Vol. 1, no. July, s. 105–112. Dostupné na internete: <http://ieeexplore.ieee.org/document/937505/>.

[7] BRADSKI, G. - KAEHLER, A. *Learning OpenCV: Computer Vision with the OpenCV Library* [online]. . First. vyd. USA: O'Reilly Media, 2008. 555 s. ISBN 978-0-596-51613-0.

[8] BRAIN, M. - IMAGE, T. Multimodal Brain Tumor Image Segmentation Benchmark : " Change Detection ". In *Multimodal Brain Tumor Image Segmentation (BRATS) Challenge, MICCAI* . 2016. .

[9] CANDEMIR, S. et al. Graph Cut Based Automatic Lung Boundary Detection in Chest Radiographs. In . 2012. s. 7–9. .

[10] COHEN, L.D. Contour Models. In *CVGIP: Graphical Models and Image Processing* . 1991. Vol. 53, no. 2, s. 211–218. .

[11] COLLIGNON, A. et al. Automated multi-modality image registration based on information theory. In *14th International Conference on Information Processing in Medical Imaging* . 1995. s. 263–274. .

[12] CRIMI, A. et al. *Brainlesion : Glioma , Multiple Sclerosis ,. .* 2016. ISBN 9783319555232.

[13] CRIMI, A. et al. Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries. In *MICCAI 2015* . [s.l.]: Springer, 2015. s. 1–122. .

[14] CRUM, W.R. et al. Non-rigid image registration: theory and practice. In *The British Journal of Radiology* [online]. 2004. Vol. 77, no. suppl_2, s. S140–S153. Dostupné na internete: <http://www.birpublications.org/doi/10.1259/bjr/25329214>.

[15] DIAZ, I. - BOULANGER, P. Atlas to patient registration with brain tumor based on a mesh-free method. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS* . 2015. Vol. 2015–Novem, s. 2924–2927. .

[16] ELNAKIB, A. et al. *Multi Modality State-of-the-Art Medical Image Segmentation and*

Registration Methodologies [online]. . 2011. 1-39 s. ISBN 978-1-4419-8203-2.

[17] FERRANTE, E. - PARAGIOS, N. Slice-to-volume medical image registration: A survey. In *Medical Image Analysis* [online]. 2017. Vol. 39, s. 101–123. Dostupné na internete: <http://dx.doi.org/10.1016/j.media.2017.04.010>.

[18] FUERST, B. et al. Automatic ultrasound-MRI registration for neurosurgery using the 2D and 3D LC2 Metric. In *Medical Image Analysis* [online]. 2014. Vol. 18, no. 8, s. 1312–1319. Dostupné na internete: <http://dx.doi.org/10.1016/j.media.2014.04.008>.

[19] GLASBEY, C.A. AND HORGAN, G.W. Chapter 4 Segmentation. In *Image Analysis for the Biological Sciences* . 1995. s. 1–31. .

[20] GOODFELLOW, I. et al. *Deep Learning* [online]. . [s.l.]: MIT Press, 2016. .

[21] GOOYA, A. et al. An EM algorithm for brain tumor image registration: A tumor growth modeling based approach. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops* [online]. 2010. s. 39–46. Dostupné na internete: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5543440>.

[22] GRAU, V. et al. Improved Watershed Transform for Medical Image Segmentation Using Prior Information. In *IEEE Transactions on Medical Imaging* [online]. 2004. Vol. 23, no. 4, s. 447–458. Dostupné na internete: <http://ieeexplore.ieee.org/document/1281998/>.

[23] HAO, X.H.X. et al. A novel region growing method for segmenting ultrasound images. In *2000 IEEE Ultrasonics Symposium. Proceedings. An International Symposium (Cat. No.00CH37121)* . 2000. Vol. 2, no. November, s. 1–4. .

[24] HAVAEI, M. et al. Brain tumor segmentation with Deep Neural Networks. In *Medical Image Analysis* [online]. 2017. Vol. 35, s. 18–31. Dostupné na internete: <http://dx.doi.org/10.1016/j.media.2016.05.004>.

[25] HIEN, N.M. - BINH, N.T. 2016. .

[26] HUANG, Y.L. et al. Level set segmentation for breast tumor in 2-D sonography. In *International Journal of Computer Assisted Radiology and Surgery* . 2006. Vol. 1, no. SUPPL. 7, s. 63–65. .

[27] IVINS, J. - PORRILL, J. EVERYTHING YOU ALWAYS WANTED (BUT WERE AFRAID TO ASK). In *Vision Research* [online]. 2000. Vol. 86, no. July 1993, s. 35. Dostupné na internete: <http://www-ma1.upc.es/~susin/files/SnakesAivru86c.pdf>.

[28] JOHNSON, H.J. et al. The ITK Software Guide Book 1: Introduction and Development Guidelines Fourth Edition Updated for ITK version 4.7. In [online]. 2015. Dostupné na internete: <https://itk.org/>.

[29] KOCH, G. et al. Siamese Neural Networks for One-Shot Image Recognition. In *Proceedings of the 32 nd International Conference on Machine Learning* . 2015. Vol. 7, no. 11, s. 956–963. .

[30] KONONENKO, I. Machine learning for medical diagnosis: history, state of the art and perspective. In *Artificial intelligence in medicine* . 2001. Vol. 23, no. 1, s. 89–109. .

[31] KOTSIANTIS, S.B. et al. Machine learning: A review of classification and combining techniques. In *Artificial Intelligence Review* . 2006. Vol. 26, no. 3, s. 159–190. .

[32] KRIESEL, D. A Brief Introduction to Neural Networks. In *Retrieved August* [online]. 2005.

s. 244. Dostupné na internete: <http://www.dkriesel.com/en/science/neural_networks>.

[33] KWON, D. et al. PORTR: Pre-operative and post-recurrence brain tumor registration. In *IEEE Transactions on Medical Imaging* . 2014. Vol. 33, no. 3, s. 651–667. .

[34] KWONG, K.K. et al. Dynamic magnetic resonance imaging of human brain activity during primary sensory stimulation. In *Proceedings of the National Academy of Sciences* [online]. 1992. Vol. 89, no. 12, s. 5675–5679. [cit. 2014-12-23]. . Dostupné na internete: <http://www.pnas.org/content/89/12/5675.short>.

[35] KYBIC, J. Elastic image registration using parametric deformation models. In *Swiss Federal Institute of Technology Lausanne* [online]. 2001. Vol. 2439, s. 182. Dostupné na internete: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.4447&rep=rep1&type=pdf%5Cnpapers2://publication/uuid/9089267B-7045-48E6-85E2-336D677DA195>.

[36] KYRIACOU, S.K. et al. Nonlinear elastic registration of brain images with tumor pathology using a biomechanical model [MRI]. In *IEEE Transactions on Medical Imaging* [online]. 1999. Vol. 18, no. 7, s. 580–592. Dostupné na internete: <http://ieeexplore.ieee.org/document/790458/>.

[37] LECUN, Y. et al. Gradient-based learning applied to document recognition. In *IEE* . 1998. .

[38] LOWE, D.G. Distinctive image features from scale-invariant keypoints. In *International journal of computer vision* . 2004. Vol. 60, no. 2, s. 91–110. .

[39] MANG, A. et al. A model of tumour induced brain deformation as bio-physical prior for non-rigid image registration. In *Proceedings - International Symposium on Biomedical Imaging* . 2011. s. 578–581. .

[40] MASOOD, S. et al. A Survey on Medical Image Segmentation. In *Current Medical Imaging Reviews* [online]. 2015. Vol. 11, no. 1, s. 3–14. Dostupné na internete: <http://www.eurekaselect.com/openurl/content.php?genre=article&issn=1573-4056&volume=11&issue=1&spage=3>.

[41] MENZE, B. et al. Multimodal Brain Tumor Segmentation. In *Proc MICCAI-BRATS (Multimodal Brain Tumor Segmentation Challenge)* . 2013. .

[42] MENZE, B.H. et al. The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). In *IEEE Transactions on Medical Imaging* . 2015. Vol. 34, no. 10, s. 1993–2024. .

[43] MENZE, B.H. - JAKAB, A. Boston, Massachusetts, 2014. .

[44] NG, H.P. et al. Medical Image Segmentation Using K-Means Clustering and Improved Watershed Algorithm. In *2006 IEEE Southwest Symposium on Image Analysis and Interpretation* [online]. 2006. no. April 2014, s. 61–65. Dostupné na internete: <http://ieeexplore.ieee.org/document/1633722/>.

[45] NIELSEN, M. Neural networks. In [online]. 2017. Dostupné na internete: <http://neuralnetworksanddeeplearning.com/chap6.html>.

[46] OSOWSKI, S. et al. Support Vector Machine-Based Expert System for Reliable Heartbeat Recognition. In *IEEE Transactions on Biomedical Engineering* . 2004. Vol. 51, no. 4, s. 582–589. .

[47] POLAK, M. Motion-Robust MRI through Real-Time Motion Tracking and Retrospective Super-Resolution Volume Reconstruction. In . 2013. s. 1–12. .

[48] RAO, V. et al. Multimodal Brain Tumor Image Segmentation Benchmark. In *Multimodal Brain*

Tumor Image Segmentation (BRATS) Challenge, MICCAI [online]. 2015. Vol. 2015, s. 56. Dostupné na internete: <http://people.csail.mit.edu/menze/papers/proceedings_miccai_brats_2015.pdf>.

[49] RIVAZ, H. et al. Automatic deformable MR-ultrasound registration for image-guided neurosurgery. In *IEEE Transactions on Medical Imaging* . 2015. Vol. 34, no. 2, s. 366–380. .

[50] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. In *Psychological Review* [online]. 1958. Vol. 65, no. 6, s. 386–408. Dostupné na internete: <http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519>.

[51] ROSTEN, E. - DRUMMOND, T. Machine learning for high-speed corner detectionMachine learning for high-speed corner detection. In *Computer vision–ECCV* . 2016. s. 430–443. .

[52] RUBLEE, E. et al. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision* . 2011. s. 2564–2571. .

[53] SANAKAL, R. - JAYAKUMARI, S.T. Prognosis of Diabetes Using Data mining Approach-Fuzzy C Means Clustering and Support Vector Machines. In *International Journal of Computer Trends and Technology* . 2014. Vol. 11, no. 2, s. 94–98. .

[54] SINCLAIR, D.A. A 3D Sweep Hull Algorithm for computing Convex Hulls and Delaunay Triangulation . In . .

[55] SINHA, S. Graph Cut Algorithms in Vision, Graphics and Machine Learning An Integrative Paper. In *UNC Chapel Hill* [online]. 2004. Dostupné na internete: <http://cs.unc.edu/~ssinha/pubs/SinhaGraphCutsIP2004.pdf>.

[56] SOILLE, P. - VINCENT, L.M. Determining watersheds in digital pictures via flooding simulations. In KUNT, M.Ed. [online]. 1990. s. 240–250. [cit. 2017-05-09]. Dostupné na internete: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=951210>.

[57] STRICKLAND, R.N. *Image-Processing Techniques for Tumor Detection* [online]. . [s.l.]: CRC Press, 2002. 384 s. ISBN 0203909356.

[58] ŠIKUDOV, E. et al. *Počítačové videnie*. . 1. vyd. Praha: Wikina, 2011. 397 s. ISBN 978-80-87925-07-2.

[59] TOENNIES, K.D. *Guide to Medical Image Analysis: Methods and Algorithms*. . 2012. 98-101 s. ISBN 978-1-4471-2751-2.

[60] TOMAS-FERNANDEZ, X. - WARFIEL, S.K. Multimodal brain tumor segmentation. In *Proc MICCAI-BRATS (Multimodal Brain Tumor Segmentation Challenge)* [online]. 2012. s. 1–73. Dostupné na internete: <http://www.imm.dtu.dk/projects/BRATS2012>.

[61] WANG, G. et al. [online]. .2017. Dostupné na internete: <http://arxiv.org/abs/1709.00382>.

[62] WU, M.-N. et al. Brain Tumor Detection Using Color-Based K-Means Clustering Segmentation. In *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2007)* [online]. 2007. no. December 2007, s. 245–250. Dostupné na internete: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4457697>.

[63] XIAO, Y. et al. User-friendly freehand ultrasound calibration using Lego bricks and automatic registration. In *International Journal of Computer Assisted Radiology and Surgery* . 2016. Vol. 11, no. 9, s. 1703–1711. .

[64] YAVARIABDI, A. et al. Mapping and characterizing endometrial implants by registering 2D

transvaginal ultrasound to 3D pelvic magnetic resonance images. In *Computerized Medical Imaging and Graphics* [online]. 2015. Vol. 45, s. 11–25. Dostupné na internete: <http://dx.doi.org/10.1016/j.compmedimag.2015.07.007>.

[65] YUSHKEVICH, P.A. et al. User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. In *NeuroImage* . 2006. Vol. 31, no. 3, s. 1116–1128. .

[66] ZACHARAKI, E.I. et al. Registration of brain images with tumors: towards the construction of statistical atlases for therapy planning. In *Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on* . 2006. s. 197–200. .

[67] ZEILER, M.D. - FERGUS, R. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* . 2014. Vol. 8689 LNCS, no. PART 1, s. 818–833. .

[68] ZIKIC, D. et al. Decision forests for tissue-specific segmentation of high-grade gliomas in multi-channel MR. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* [online]. 2012. Vol. 15, no. Pt 3, s. 369–76. Dostupné na internete: <http://link.springer.com/10.1007/978-3-642-33454-2_46%5Cnhttp://www.ncbi.nlm.nih.gov/pubmed/23286152>.

[69] ZITOVÁ, B. - FLUSSER, J. Image registration methods: A survey. In *Image and Vision Computing* . 2003. Vol. 21, no. 11, s. 977–1000. .

[70] 5.3 Experiments. In [online]. [cit. 2017-12-07]. Dostupné na internete: <https://www.cs.sfu.ca/~stella/papers/blairthesis/main/node31.html>.

[71] ANTs N4ITK bias correction. In [online]. Dostupné na internete: <https://github.com/ANTsX/ANTs/blob/master/Examples/N4BiasFieldCorrection.cxx>.

[72] BRATS 2015 dataset repository. In [online]. Dostupné na internete: <https://www.smir.ch/BRATS/Start2015>.

[73] Documentation | VTK. In [online]. [cit. 2017-05-08]. Dostupné na internete: <http://www.vtk.org/documentation/>.

[74] FAST Algorithm for Corner Detection — OpenCV 3.0.0-dev documentation. In [online]. [cit. 2017-05-08]. Dostupné na internete: <http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html>.

[75] graph cut. In [online]. [cit. 2017-12-07]. Dostupné na internete: <http://cybertron.cg.tu-berlin.de/xiwang/imgs/visapp2014.png>.

[76] Introduction to SURF (Speeded-Up Robust Features) — OpenCV 3.0.0-dev documentation. In [online]. [cit. 2017-05-08]. Dostupné na internete: <http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html>.

[77] MICCAI 2016 - Day 2 - Keynote - Yoav Medan - YouTube. In [online]. [cit. 2017-05-08]. Dostupné na internete: <https://www.youtube.com/watch?v=zVQCKrMXrnQ&index=6&list=PLsrNuu93xWxUDbZ9h5NLlOQ7ax5HZYTen>.

[78] OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform). In [online]. [cit. 2017-05-08]. Dostupné na internete: <http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html>.

[79] ORB (Oriented FAST and Rotated BRIEF) — OpenCV 3.0.0-dev documentation. In [online]. [cit. 2018-04-28]. Dostupné na internete: <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html>.

[80] region growing. In [online]. [cit. 2017-12-07]. Dostupné na internete: <https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/35269/versions/2/screenshot.jpg>.

[81] repository vitalab / VITALabAI_public / source / VITALabAI / model / brats — Bitbucket. In [online]. 2017. [cit. 2017-12-11]. Dostupné na internete: <https://bitbucket.org/vitalab/vitalabai_public/src/19cd5eba023d8c75e2dcb75d77330fb00c24ce99/VITALabAI/model/brats/?at=master>.

[82] SVM. In [online]. [cit. 2017-12-07]. Dostupné na internete: <https://www.analyticsvidhya.com/wp-content/uploads/2015/10/SVM_1.png>.

[83] web page of Asimov institute. In [online]. [cit. 2017-12-08]. Dostupné na internete: <http://www.asimovinstitute.org/neural-network-zoo/>.

[84] Welcome to opencv documentation! — OpenCV 2.4.13.2 documentation. In [online]. [cit. 2017-05-08]. Dostupné na internete: <http://docs.opencv.org/2.4/index.html>.

Appendix A: Technical documentation

All source codes were developed in Python and also OpenCV, SimpleITK and other Python libraries were used. Visualization of the volumetric data was developed using VTK library, PyQt and PyQt plugin for VTK.

Preprocessing

Preprocessing can process 3D or 2D data, input for the preprocessing are reference and target data. Preprocessing is initialized with target data. Target is normalized and some percentile of the data can be removed – see Listing 1 (target = template).

Listing 1: *Initialization of preprocessing*

```
def __init__(self, template=None, remove_percentile=False, bottom_percentile=5,
top_percentile=99, ignore_zero_values=False):

        self.ignore_zero_values = ignore_zero_values
        self.bottom_percentile = bottom_percentile
        self.top_percentile = top_percentile
        self.rem_percentile = remove_percentile


        if self.ignore_zero_values:
            tmp = np.ma.masked_where(template == 0, template)
            tmp = tmp.compressed()
        else:
            tmp = None

        if template is not None:
            if self.rem_percentile:
                template = self.remove_percentile(data=template, data_info=tmp,
                                                  bottom_percentile=self.bottom_percentile,
                                                  top_percentile=self.top_percentile)
            template = self.normalize(data=template)

            self.template = template
```

Then the reference image can be processed. Reference image is normalized and the same percentile can be removed from the data, but reference data are also matched to the target using histogram matching algorithm – see Listing 2.

Listing 2: *Function for processing the reference data*

```
def process_data(self, data, path=None):

        if self.ignore_zero_values:
            # ignore black backround - zero values
            tmp = np.ma.masked_where(data == 0, data)
            tmp = tmp.compressed()
        else:
            tmp = None

        if self.rem_percentile:
            data = self.remove_percentile(data=data, data_info=tmp,
bottom_percentile=self.bottom_percentile, top_percentile=self.top_percentile)
        data = self.normalize(data=data)
        data = self.hist_match(template=self.template, data=data)

        return self.normalize(data)
```

In the preprocessing some help functions are used: *normalize()* function – Listing 3, *remove_percentile()* function – Listing 4 and *hist_match()* function – Listing 5.

Listing 3: *Normalization of the data*

```
def normalize(self, data):

        new_min = np.amin(data)
        new_max = np.amax(data)
        normalized = ((data - new_min) / (new_max - new_min)) * 255 #65535

        return np.uint8(normalized)
```

Listing 4: *Remove some percentile from the data*

```
def remove_percentile(self, data, data_info=None, bottom_percentile=10, top_percentile=99):

        if data_info is not None:
            new_min = np.percentile(data_info, bottom_percentile)
            new_max = np.percentile(data_info, top_percentile)
        else:
            new_min = np.percentile(data, bottom_percentile)
            new_max = np.percentile(data, top_percentile)

        data[np.where(data < new_min)] = new_min
        data[np.where(data > new_max)] = new_ma

        return data
```

Listing 5: *Histogram matching of the target and reference data*

```
def hist_match(self, template, data):
        oldshape = data.shape
        data = data.ravel()
        template = template.ravel()

        # get the set of unique pixel values and their corresponding indices and
        # counts
        s_values, bin_idx, s_counts = np.unique(data, return_inverse=True,
                                                return_counts=True)
        t_values, t_counts = np.unique(template, return_counts=True)

        # take the cumsum of the counts and normalize by the number of pixels to
        # get the empirical cumulative distribution functions for the data and
        # template images (maps pixel value --> quantile)
        s_quantiles = np.cumsum(s_counts).astype(np.float64)
        s_quantiles /= s_quantiles[-1]
        t_quantiles = np.cumsum(t_counts).astype(np.float64)
        t_quantiles /= t_quantiles[-1]

        # interpolate linearly to find the pixel values in the template image
        # that correspond most closely to the quantiles in the data image
        interp_t_values = np.interp(s_quantiles, t_quantiles, t_values)

        return interp_t_values[bin_idx].reshape(oldshape)
```

Rigid registration

Input for the algorithm are two 2D images generated from volumes of magnetic resonance test in axial direction. One of the images is reference and next image is input which will be registered. Next listings show important steps of the algorithm.

In the first step, mask for registration is created. There are two different types of masks. One is a subtraction of the mask of the tumor segmentation from the mask of the brain. Mask of the brain is created using threshold method. Second one is mask of the cranium created from mask of the brain using find contours algorithm. See Listing 6.

Listing 6: *Create mask for registration*

```
# creation of mask for area which we will work with in registration process
def __create_mask_for_registration(self, image, segmentation_mask, idx, path):

        # threshold input image first to get mask of brain
        _, thresholded_image = cv2.threshold(image, 10, 255, cv2.THRESH_BINARY)

        if self.registration_mask_type is 'health_brain':
            # reduction of mask to remove margins of brain
            kernel = np.ones((5, 5), np.uint8)
            thresholded_image = cv2.dilate(thresholded_image, kernel=kernel, iterations=1)
            kernel = np.ones((15, 15), np.uint8)
            thresholded_image = cv2.erode(thresholded_image, kernel=kernel, iterations=1)
            # substract tumor from mask
            result = thresholded_image - segmentation_mask
            _, result = cv2.threshold(result, 250, 255, cv2.THRESH_BINARY)

        elif self.registration_mask_type is 'cranium':
            kernel = np.ones((10, 10), np.uint8)
            thresholded_image = cv2.dilate(thresholded_image, kernel=kernel, iterations=1)
            kernel = np.ones((7, 7), np.uint8)
            thresholded_image = cv2.erode(thresholded_image, kernel=kernel, iterations=1)
            _, contours, _ = cv2.findContours(thresholded_image, cv2.RETR_EXTERNAL,
                                              cv2.CHAIN_APPROX_NONE)
            result = np.zeros_like(image)
            cv2.drawContours(result, contours, -1, 255, 20)

        if self.show_results:

        return result
```

In the next step we can detect the key points using SIFT, SURF, ORB or FAST key point detector (Listing 7) and compute feature vectors using SIFT, SURF or ORB descriptors (Listing 8).

Listing 7: *Detect the key points using SIFT, SURF or FAST detector*

```
# detect keypoint using SIFT detector
def __detect_keypoints_SIFT(self, image, registration_mask, idx, path):
        sift = cv2.xfeatures2d.SIFT_create()
        keypoints = sift.detect(image, registration_mask)
        if self.show_results or self.save_results:
            self.__panit_keypoints(image, keypoints, idx, path)
        return keypoints

# detect keypoint using SURF detector
def __detect_keypoints_SURF(self, image, registration_mask, idx, path):
        surf = cv2.xfeatures2d.SURF_create(400)
        keypoints = surf.detect(image, registration_mask)
        if self.show_results or self.save_results:
            self.__panit_keypoints(image, keypoints, idx, path)
        return keypoints

    # detect keypoint using ORB detector
```

```
def __detect_keypoints_ORB(self, image, registration_mask, idx, path):
        orb = cv2.ORB_create()
        keypoints = orb.detect(image, registration_mask)
        if self.show_results or self.save_results:
            self.__panit_keypoints(image, keypoints, idx, path)
        return keypoints

    # detect keypoint using FAST detector
def __detect_keypoints_FAST(self, image, registration_mask, idx, path):
        fast = cv2.FastFeatureDetector_create()
        keypoints = fast.detect(image, registration_mask)
        if self.show_results or self.save_results:
            self.__panit_keypoints(image, keypoints, idx, path)
        return keypoints
```

Listing 3: *Calculate descriptors (feature vectors) using SIFT or SURF*

```
# compute descriptors using SIFT
def __calculate_descriptors_SIFT(self, image, keypoints):
        sift = cv2.xfeatures2d.SIFT_create()
        _, descriptors = sift.compute(image, keypoints)
        return descriptors

# compute descriptors using SURF
def __calculate_descriptors_SURF(self, image, keypoints):
        surf = cv2.xfeatures2d.SURF_create(400)
        _, descriptors = surf.compute(image, keypoints)
        return descriptors

# compute descriptors using ORB
def __calculate_descriptors_ORB(self, image, keypoints):
        orb = cv2.ORB_create()
        _, descriptors = orb.compute(image, keypoints)
        return descriptors
```

Then feature vectors can be matched using brute force matcher. See Listing 8.

Listing 8: *Match descriptor vectors*

```
# get all matches (depending on computed descriptors) between the keypoints of both images
def __get_matches(self, desc_1, desc_2, img_1, img_2, keypoints_1, keypoints_2, path):
        # create BFMatcher object
        if self.key_point_descriptor == 'SIFT' or self.key_point_descriptor == 'SURF':
            bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
        else:
            bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

        # match descriptors
        matches = bf.match(desc_1, desc_2)

        return matches
```

Good matches were filtered depending on the prejudice that brain volume is not a lot rotated to each other so lines between matches should be almost evenly. See Listing 9.

Listing 9: *Filter matches*

```
# filter good matches from all matches - they should not be very far away
def __get_good_matches(self, matches, keypoints_1, keypoints_2, img_1, img_2, path,
tolerancy_lr=20, tolerancy_tb=20):
        good_matches = []
        for i in range(0, len(matches)):

            x1 = keypoints_1[matches[i].queryIdx].pt[0]
            y1 = keypoints_1[matches[i].queryIdx].pt[1]
```

```
            x2 = keypoints_2[matches[i].trainIdx].pt[0]
            y2 = keypoints_2[matches[i].trainIdx].pt[1]

            # good matches have to satisfy the criteria
            if x1 + tolerancy_lr >= x2 and x1 - tolerancy_lr <= x2 and
               y1 + tolerancy_tb >= y2 and y1 - tolerancy_tb <= y2:
                good_matches.append(matches[i])

        return good_matches
```

Finally, transformation matrix is computed from the filtered matches and input is warped to the reference image – registration is completed. See Listing 10.

Listing 10: *Rigid transformation*

```
# find afine transform
afine = cv2.estimateRigidTransform(dst_pts, src_pts, False)

if afine is not None:
            # apply afine transform
            img_result = cv2.warpAffine(img_2, afine, (img_2.shape[1], img_2.shape[0]))
```

Non rigid registration

Non rigid registration can process 3D data, but it still works slice by slice. It is based on the optical flow. We used two different algorithms Lukas-Kanade optical flow (Listing 11) and Farneback dense optical flow (Listing 12)

Listing 11: *Lukas-Kanade optical flow*

```
# function process all corresponding frames of examination volumes using lucas-canade optical
flow
def process_frames_lk(self, frames, paths=None, path_extension=None, modality='flair'):
        old_gray = None
        p0 = None

        for index, frame in enumerate(frames):
            result = None
            warped result = []
            old_points = []
            new_points = []
            save_path = ""

            if old_gray is None and p0 is None:
                # take first frame
                old_gray = frame.copy()
                # find corners on first frame
                p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **self.feature_params)
            else:
                # actual image
                actual_gray = frame.copy()
                # calculate optical flow - Lucas Kanade
                p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, actual_gray, p0, None,
                                                       **self.lk_params)
                # select good points
                good_old_points = p0[st == 1]
                good_actual_points = p1[st == 1]
                old_points = list(tuple(map(tuple, good_old_points)))
                new_points = list(tuple(map(tuple, good_actual_points)))

                # warp old image to actual image for evaluation using Delaunay
                warped_result = self.warp_image(img_old=old_gray.copy(),
                                    img_new=actual_gray.copy(), old_points=old_points,
                                    new_points=new_points, path=save_path)
                # morph images for visualization
                self.morph_images(img_old=old_gray.copy(), img_new=actual_gray.copy(),
```

```
                            old_points=old_points, new_points=new_points, path=save_path)

                # update the previous frame and previous points
                old gray = actual gray.copy()
                p0 = good_actual_points.reshape(-1, 1, 2)

        return warped_frames, old_points_list, new_points_list
```

Listing 12: *Farneback optical flow*

```
# function process all corresponding frames of examination volumes using farneback algorithm
def process_frames_farneback(self, frames, paths=None, path_extension=None, modality='flair'):
        old_gray = None

        for index, frame in enumerate(frames):
            result = None
            old_points = None
            new_points = None
            save_path = ""

            if old_gray is None:
                # take first frame
                old_gray = frame.copy()
            else:
                # actual image
                actual gray = frame.copy()
                # calc optical flow using Farneback
                flow = cv2.calcOpticalFlowFarneback(old_gray, actual_gray, None,
                                                    **self.farneback_params)
                # get old (for old_gray) and new points (for actual grey) from flow
                old_points, new_points = self.__get_points_from_flow(img=actual_gray,
                                                                     flow=flow, step=16)
                # warp old image to actual image for evaluation using Delaunay
                result = self.warp_image(img_old=old_gray.copy(), img_new=actual_gray.copy(),
                                         old_points=old_points, new_points=new_points,
                                         animate=False, path=save_path)
                # morph images for visualization
                self.morph_images(img_old=old_gray.copy(), img_new=actual_gray.copy(),
                                  old_points=old_points, new_points=new_points, animate=False,
                                  path=save_path)

                old_gray = actual_gray.copy()
        return warped_frames, old_points_list, new_points_list
```

If we are using Farneback algorithm, we have to compute points from the flow. See Listing 13.

Listing 13: *Compute points from the Farneback optical flow*

```
# function returns points of old frame and actual frame computed from flow
def __get_points_from_flow(self, img, flow, step=8):
        h, w = flow.shape[:2]
        y, x = np.mgrid[step / 2:h:step, step / 2:w:step].reshape(2, -1).astype(int)
        fx, fy = flow[y, x].T
        lines = np.vstack([x, y, x + fx, y + fy]).T.reshape(-1, 2, 2)
        lines = np.int32(lines + 0.5)

        old points = []
        new points = []
        for (x1, y1), (_x2, _y2) in lines:
            if img[y1][x1] > 0 and img[_y2][_x2] > 0: # points inside the brain are accepted
                old_points.append((x1, y1))
                new_points.append((_x2, _y2))

        return old_points, new_points
```

Finally, we can visualize the changes using different visualization techniques. We can draw track after Lukas-Kanade algorithm (Listing 14), draw tracks after Farneback algorithm (Listing 15) or crate HSV image from Farnebask dense flow (Listing 16).

Listing 14: *Draw tracks after Lukas-Kanade algorithm*

```
# function draw motions between examination frames after lucas-canade optical flow
def __draw_tracks_after_lk(self, actual_img, good_old, good_new, mask):
        # draw the tracks
        for i, (new, old) in enumerate(zip(good_new, good_old)):
            a, b = new.ravel()
            c, d = old.ravel()
            mask = cv2.line(mask, (a, b), (c, d), self.color[i].tolist(), 2)

        img = cv2.add(actual_img, mask)

        return mask, img
```

Listing 15: *Draw tracks after Farneback algorithm*

```
# function draw flow - motions between examination frames after farneback
def __draw_flow_after_farneback(self, img, old_points, new_points):

        vis = cv2.cvtColor(img, cv2.COLOR GRAY2BGR)

        for (x1, y1), (_x2, _y2) in zip(old_points, new_points):
            cv2.arrowedLine(vis, (x1, y1), ( x2,  y2), (0, 255, 0), 1)
            cv2.circle(vis, (x1, y1), 1, (0, 255, 0), -1)

        return vis
```

Listing 16: *Create HSV image form Farneback dense flow*

```
# function draw HSV after farneback - it shows motion between two examinations
def __draw_hsv_after_farneback(self, flow):

        h, w = flow.shape[:2]
        fx, fy = flow[:, :, 0], flow[:, :, 1]
        ang = np.arctan2(fy, fx) + np.pi
        v = np.sqrt(fx*fx+fy*fy)
        hsv = np.zeros((h, w, 3), np.uint8)
        hsv[..., 0] = ang*(180/np.pi/2)
        hsv[..., 1] = 255
        hsv[..., 2] = np.minimum(v*4, 255)
        bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

        return bgr
```

We can also visualize the changes using animation. For the animation we can create additional images to provide more smooth animation. Thus, we implemented image morphing algorithm based on corresponding point, Delaunay triangulation and alpha blending (see Listing 17).

Listing 17: *Image morphing algorithm*

```
# compute delaunay triangulation
def delaunay_triangulation(self, points, img, animate=True, path=None):

        # keep a copy of input image
        img_orig = img.copy()
```

```
        # create rectangle for Subdiv2D
        size = img.shape
        rect = (0, 0, size[1], size[0])

        # create an instance of Subdiv2D
        subdiv = cv2.Subdiv2D(rect)

        for idx, p in enumerate(points):
            # insert point to the SubDiv2D
            subdiv.insert(p)


        # get triangles from subdiv
        triangleList = subdiv.getTriangleList()
        size = img.shape
        r = (0, 0, size[1], size[0])

        # select only good triangles and save as lst of points
        triangles = []
        for t in triangleList:
            pt1 = (int(t[0]), int(t[1]))
            pt2 = (int(t[2]), int(t[3]))
            pt3 = (int(t[4]), int(t[5]))

            if self.__rect_contains__(r, pt1) and self.__rect_contains__(r, pt2) and
               self.__rect_contains__(r, pt3):
                triangles.append([pt1, pt2, pt3])

        return triangles

# transform one triangle to another one
def apply_transform(self, src, srcTri, dstTri, size):
        # pair of triangles is used to find the affine transform
        warpMat = cv2.getAffineTransform(np.float32(srcTri), np.float32(dstTri))
        # apply affine transform
        dst = cv2.warpAffine(src, warpMat, (size[0], size[1]), None, flags=cv2.INTER_LINEAR,
                             borderMode=cv2.BORDER REFLECT 101)
        return dst

# morph old and actual image for visualization
def morph_images(self, img_old, img_new, old_points, new_points, animate=False, path=None):
        alpha = 0.5

        # morph images without warping
        img_morphed = np.uint8((1.0 - alpha) * img_old + alpha * img_new)

        r = cv2.boundingRect(img_old)
        b1 = (r[0], r[1])
        b2 = (r[0] + r[2], r[1])
        b3 = (r[0] + r[2], r[1] + r[3])
        b4 = (r[0], r[1] + r[3])

        # compute weighted average point coordinates
        points = []

        old_points.extend((b1, b2, b3, b4))
        new_points.extend((b1, b2, b3, b4))

        for p1, p2 in zip(old_points, new_points):
            x = (1 - alpha) * p1[0] + alpha * p2[0]
            y = (1 - alpha) * p1[1] + alpha * p2[1]
            points.append((int(x), int(y)))

        # compute delaunay trinagulation on weighted points
        triangles = self.delaunay_triangulation(points=points, img=img_old, animate=animate,
                                                path=path)

        # get triangles of old point set and actual points set
        triangles old = []
        triangles new = []
        for t in triangles:
            p1 i = points.index(t[0])
            p2_i = points.index(t[1])
            p3_i = points.index(t[2])
            triangles old.append([old points[p1 i], old points[p2 i], old points[p3 i]])
            triangles_new.append([new_points[p1_i], new_points[p2_i], new_points[p3_i]])
```

```
        # morph images
        img_morph = img_old.copy()
        img_paint = np.zeros(img_old.shape, dtype=img_old.dtype)

        idx = 0
        for t, t1, t2 in zip(triangles, triangles_old, triangles_new):
            # find bounding rectangle for triangles
            r = cv2.boundingRect(np.float32([t]))
            r1 = cv2.boundingRect(np.float32([t1]))
            r2 = cv2.boundingRect(np.float32([t2]))

            tRect = []
            t1Rect = []
            t2Rect = []

            for i in range(0, 3):
                tRect.append(((t[i][0] - r[0]), (t[i][1] - r[1])))
                t1Rect.append(((t1[i][0] - r1[0]), (t1[i][1] - r1[1])))
                t2Rect.append(((t2[i][0] - r2[0]), (t2[i][1] - r2[1])))

            # get mask of processed triangle by filling triangle
            mask = np.zeros((r[3], r[2]), dtype=np.float32)
            cv2.fillConvexPoly(mask, np.int32(t2Rect), (1.0, 1.0, 1.0), 16, 0)

            img1Rect = img_old[r1[1]:r1[1] + r1[3], r1[0]:r1[0] + r1[2]]
            img2Rect = img_new[r2[1]:r2[1] + r2[3], r2[0]:r2[0] + r2[2]]

            size = (r[2], r[3])
            warpImage1 = self.apply_transform(img1Rect, t1Rect, tRect, size)
            warpImage2 = self.apply_transform(img2Rect, t2Rect, tRect, size)

            # alpha blending for rectangular patches
            imgRect = (1.0 - alpha) * warpImage1 + alpha * warpImage2

            # copy triangular region of the rectangular patch to the output image
            img_morph[r[1]:r[1] + r[3], r[0]:r[0] + r[2]] = img_morph[r[1]:r[1] +
                                r[3],r[0]:r[0] + r[2]] * (1 - mask) + imgRect * mask
            img_paint[r[1]:r[1] + r[3], r[0]:r[0] + r[2]] = img_paint[r[1]:r[1] +
                                r[3],r[0]:r[0] + r[2]] * (1 - mask) + imgRect * mask
```

Visualization tool

Function *volumeRender()* is used to render brain and tumor volume in 3D model visualization. At first transfer functions for opacity and color are defined and all voxels are appended to the defined functions. Then ray casting algorithm is used to create the volume and defined properties of the input data are mapped to the volume. See Listing 18.

Listing 18: *Creating 3D model and preparation for the visualization*

```
def volumeRender(self, img, tf=[], spacing=[1.0, 1.0, 1.0]):

        importer = self.numpy2VTK(img, spacing)

        # Transfer Functions
        opacity_tf = vtk.vtkPiecewiseFunction()
        color_tf = vtk.vtkColorTransferFunction()

        if len(tf) == 0:
            tf.append([img.min(), 0, 0, 0, 0])
            tf.append([img.max(), 1, 1, 1, 1])

        for p in tf:
            color_tf.AddRGBPoint(p[0], p[1], p[2], p[3])
            opacity_tf.AddPoint(p[0], p[4])
```

```
        # working on the CPU
        volMapper = vtk.vtkVolumeRayCastMapper()
        compositeFunction = vtk.vtkVolumeRayCastCompositeFunction()
        compositeFunction.SetCompositeMethodToInterpolateFirst()
        volMapper.SetVolumeRayCastFunction(compositeFunction)
        volMapper.SetInputConnection(importer.GetOutputPort())

        # The property describes how the data will look
        self.volProperty = vtk.vtkVolumeProperty()
        self.volProperty.SetColor(color_tf)
        self.volProperty.SetScalarOpacity(opacity_tf)
        self.volProperty.ShadeOn()
        self.volProperty.SetInterpolationTypeToLinear()

        # the lines below speed things up
        # pix_diag = 5.0
        # volMapper.SetSampleDistance(pix_diag / 5.0)
        # volProperty.SetScalarOpacityUnitDistance(pix_diag)


        volume = vtk.vtkVolume()
        volume.SetMapper(volMapper)
        volume.SetProperty(self.volProperty)

        return volume
```

Slices in axial, coronal and sagittal direction are created from the volume to visualize data in 2D. At first data are loaded and additional information, such as spacing, minimal and maximal position of the data are reached from the input. Spacing information is then used to compute center of the volume. Positions of the center are parameters for the matrixes. They are used for slice creation. See Listing 19.

Listing 19: *Creating slices and preparation for the visualization*

```
def getBrainSlices(self, sliceType):
        VTK_DATA_ROOT = vtkGetDataRoot()

        # Start by loading some data.
        reader = vtk.vtkMetaImageReader()
        reader.SetFileName("../../data/brain/flair.mha")
        reader.Update()

        # Calculate the center of the volume
        reader.Update()
        (xMin, xMax, yMin, yMax, zMin, zMax) =
                        reader.GetExecutive().GetWholeExtent(reader.GetOutputInformation(0))
        (xSpacing, ySpacing, zSpacing) = reader.GetOutput().GetSpacing()
        (x0, y0, z0) = reader.GetOutput().GetOrigin()

        center = [x0 + xSpacing * 0.5 * (xMin + xMax),
                  y0 + ySpacing * 0.5 * (yMin + yMax),
                  z0 + zSpacing * 0.5 * (zMin + zMax)]

        metrix = vtk.vtkMatrix4x4()
        # Matrices for axial, coronal, sagittal, oblique view orientations
        if sliceType == "coronal":
            metrix.DeepCopy((1, 0, 0, center[0],
                             0, 0, 1, center[1],
                             0, 1, 0, center[2],
                             0, 0, 0, 1))

            self.axes.paintAxeZ([center[0], - center[1], 0], [center[0], - center[1], 250],
                                self.modelRenderer)
            self.axes.setCenterZ(center[2])

        elif sliceType == "sagittal":
            metrix.DeepCopy((0, 0,-1, center[0],
                             1, 0, 0, center[1],
                             0, 1, 0, center[2],
                             0, 0, 0, 1))
```

```
            self.axes.paintAxeY([center[0], 0, center[2]], [center[0], 250, center[2]],
                                self.modelRenderer)
            self.axes.setCenterY(-center[1])

        elif sliceType == "oblique":
            metrix.DeepCopy((1, 0, 0, center[0],
                             0, 0.866025, -0.5, center[1],
                             0, 0.5, 0.866025, center[2],
                             0, 0, 0, 1))
            self.line = self.axeObject.paintLine([0, 0, 0], [0, 0, 0])

        else:  # axial
            metrix.DeepCopy((1, 0, 0, center[0],
                            0, 1, 0, center[1],
                            0, 0, 1, center[2],
                            0, 0, 0, 1))
            self.axes.paintAxeX([0, - center[1], center[2]], [250, - center[1], center[2]],
                                self.modelRenderer)
            self.axes.setCenterX(center[0])

        self.modelWindow.Render()

        # Extract a slice in the desired orientation
        #reslice = vtk.vtkImageReslice()
        self.resliceBrain.SetInputConnection(reader.GetOutputPort())
        self.resliceBrain.SetOutputDimensionality(2)
        self.resliceBrain.SetResliceAxes(metrix)
        self.resliceBrain.SetInterpolationModeToLinear()

        # Create a greyscale lookup table
        self.table.SetRange(0, 2000) # image intensity range
        self.table.SetValueRange(0.0, 1.0) # from black to white
        self.table.SetSaturationRange(0.0, 0.0) # no color saturation
        self.table.SetRampToLinear()
        self.table.Build()

        # Map the image through the lookup table
        color = vtk.vtkImageMapToColors()
        color.SetLookupTable(self.table)
        color.SetInputConnection(self.resliceBrain.GetOutputPort())

        threshold = vtk.vtkThreshold()
        threshold.SetInputConnection(color.GetOutputPort())
        threshold.ThresholdByUpper(100)

        # Display the image
        actor = vtk.vtkImageActor()
        actor.GetMapper().SetInputConnection(color.GetOutputPort())

        return actor
```

Histogram is counted separately for the health brain and for the tumor structures. Then the values are used to paint the histogram on the canvases using simple rectangles. See Listing 20.

Listing 20: *Histogram calculation and visualization*

```
def makeHistogram(self, data):

        min = 100000
        max = 0

        self.points = vtk.vtkPoints()
        polygons = vtk.vtkCellArray()

        for idx, d in enumerate(data):
            if idx > 0:
                if d < min:
                    min = d
                if d > max:
                    max = d
```

```
        for idx, d in enumerate(data):
            if(idx > 0):
                print(d)
                d = (d - min)/(max-min)
                print(d)
                polygons.InsertNextCell(self.createRectangle((idx-1)*0.1, (idx-1)*0.1+0.1, 0, d, (idx-
1)*4))

        # polygons.InsertNextCell(self.createRectangle(10, 20, 0, 40, 4))

        polygonPolyData = vtk.vtkPolyData()
        polygonPolyData.SetPoints(self.points)
        polygonPolyData.SetPolys(polygons)

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputData(polygonPolyData)

        actor = vtk.vtkActor()
        actor.SetMapper(mapper)

        axeObject = axe.Axe()
        actorX = axeObject.paintLine([0, 0, 0], [len(data)*0.1, 0, 0])
        actorY = axeObject.paintLine([0, 0, 0], [0, 1, 0])

        actorList = []
        actorList.append(actor)
        actorList.append(actorX)
        actorList.append(actorY)

        return actorList

    def createRectangle(self, x1, x2, y1, y2, i):
        self.points.InsertNextPoint(x1, y1, 0)
        self.points.InsertNextPoint(x1, y2, 0)
        self.points.InsertNextPoint(x2, y2, 0)
        self.points.InsertNextPoint(x2, y1, 0)

        polygon = vtk.vtkPolygon()
        polygon.GetPointIds().SetNumberOfIds(4)
        polygon.GetPointIds().SetId(0, i)
        polygon.GetPointIds().SetId(1, i+1)
        polygon.GetPointIds().SetId(2, i+2)
        polygon.GetPointIds().SetId(3, i+3)

        return polygon
```

Axes in the 3D model of the brain are created for the better navigation. Axes correspondent with slices displayed in the widgets. Thus, orders of the displayed slices are used to create lines in the 3D model of the brain. See Listing 21.

Listing 21: *Display the axes in the model*

```
def paintLine(self, x1, x2):

        # Create a vtkPoints object and store the points in it
        pts = vtk.vtkPoints()
        pts.InsertNextPoint(x1)
        pts.InsertNextPoint(x2)

        # Setup two colors - one for each line
        green = [0, 255, 0]

        # Setup the colors array
        colors = vtk.vtkUnsignedCharArray()
        colors.SetNumberOfComponents(3)
        colors.SetName("Colors")

        # Add the colors we created to the colors array
```

```
colors.InsertNextTuple(green)

# Create the x line
linex = vtk.vtkLine()
linex.GetPointIds().SetId(0, 0)
linex.GetPointIds().SetId(1, 1)

# Create a cell array to store the lines in and add the lines to it
lines = vtk.vtkCellArray()
lines.InsertNextCell(linex)

# Create a polydata to store everything in
linesPolyData = vtk.vtkPolyData()

# Add the points to the dataset
linesPolyData.SetPoints(pts)

# Add the lines to the dataset
linesPolyData.SetLines(lines)

# Color the lines - associate the first component (red) of the
# colors array with the first component of the cell array (line 0)
# and the second component (green) of the colors array with the
# second component of the cell array (line 1)
linesPolyData.GetCellData().SetScalars(colors)

# Visualize
mapper = vtk.vtkPolyDataMapper()
mapper.SetInputData(linesPolyData)

line_actor = vtk.vtkActor()
line_actor.SetMapper(mapper)

return line_actor
```

Appendix B: Installation guide

To run the project, follow these steps:

1. Operating system

-developed on: Ubuntu 16.04

Project was developed on Ubuntu 16.04 and also **this guide provides steps to prepare environment for the project on the Ubuntu.**

2. Python installation

-version: Python 3.5.2

Project is developed in Python programming language. During the development Python 3.5.2 was the newest version of the python programming language.

To install python in terminal use command:

```
$ sudo apt-get install python3.5-dev
```

To control python installation use command:

```
$ python --version
```

I suggest to use Pycharm IDE for writing code. If you don't have installed Pycharm yet, then dowload the sources and install. It is JetBrains software which is free for the students - download student licence.

3. Virtual environment preparation

I suggest to use virtual environment.

-requirement: pip

To find out if you have pip in terminal:

```
$ pip --version
```

If you do not have pip, use command for installation in terminal:

```
apt-get -y install python-pip
```

Finally, to install and prepare virtual environment via console:

```
$ pip install virtualenv
$ cd /my_project_folder      //go to the folder directory !!!
$ python3 -m venv env        //create virtual environment called env
$ source env/bin/activate    //activate virtual environment
```

When virtual environment is activated, in terminal you can see:

```
(env) $
```

Important: You should control, if Pycharm IDE automatically set interpreter for the project. If it didn't, it won't know libraries which will be installed in next steps. So you must set it manually (for Pycharm):

- In Pycharm open File --> Settings --> enroll Project: project_name --> click to Project Interpreter. In the field Project Interpreter, you should see path to your environment folder.

- If it is not set, add it manually. Push button next to the Project Interpreter field and choose Add Local. Then browse to the virtual environment folder.

4. Install requirements

You can install all requirements which support pip using pip command (DON'T FORGET: Virtual environment must be activated):

```
$ source env/bin/activate      //use this command only if your virtual environment
is not activated
(env) $ pip3 – r install requirements.txt          //you have to be in folder where
requirements are located
```

5. OpenCV

-version: OpenCV 3.2.0

In the project is used OpenCV library. Durig the development OpenCV 3.2.0 was the newest version.

OpenCV represents images as NumPy arrays, so we need to install NumPy into our env virtual environment (DON'T FORGET: Virtual environment must be activated):

```
$ source env/bin/activate      //use this command only if your virtual environment
is not activated
(env) $ pip3 install numpy
```

If you end up getting a Permission denied error related to pip's. cache directory, try to use following commands:

```
(env) $ sudo rm -rf ~/.cache/pip/
(env) $ pip install numpy
```

To install next prerequisites for the library use following commands (now the virtual environment don't have to be activated):

```
(env) $ deactivate
//deactivate the virtual environment
$ sudo apt-get install build-essential cmake git pkg-config
//install developer tools used to compile OpenCV
$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev
//install libraries and packages used to read various image formats
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
//install libraries used to read video formats
$ sudo apt-get install libgtk2.0-dev
//install GTK so we can use OpenCV's GUI features
```

```
$ sudo apt-get install libatlas-base-dev gfortran
//install packages that are used to optimize various functions inside OpenCV, such
as matrix operations
```

Now we can pull down OpenCV from GitHub:

```
$ cd ~
$ git clone https://github.com/Itseez/opencv.git
```

We'll also need to grab the opencv contrib repo. Without this repository, we won't have access to standard keypoint detectors and local invariant descriptors (such as SIFT, SURF, etc.) that were available in the OpenCV 2.4.X version. We'll also be missing out on some of the newer OpenCV 3.0 features like text detection in natural images.

To pull down opencv_contrib repo use commands:

```
$ cd ~
$ git clone https://github.com/Itseez/opencv_contrib.git
```

If you have also Python 2.x in your computer, you should set default python command for python 3.x. If you won't do this step, OpenCV library will be probably installed in the Python 2.x site-packages folder (/usr/local/lib/python3.4/site-packages/cv2.cpython-34m.so). To set Python 3.x as default use command:

```
$ alias python=python3
```

There is time to setup the build (cmake MUST BE installed!!!):

```
$ cd ~/opencv
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_C_EXAMPLES=ON \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D BUILD_EXAMPLES=ON ..
```

Now we can compile OpenCV (where the 4 can be replaced with the number of available cores on your processor to speedup the compilation time):

```
$ make -j4
```

Assuming OpenCV 3.0 compiled without error, you can now install it on your system:

```
$ sudo make install
$ sudo ldconfig
```

Now OpenCV should be installed in

```
/usr/local/lib/python3.5/site-packages/
```

However, in order to use OpenCV within our env virtual environment, we first need to sym-link OpenCV into the site-packages directory of the env environment, like this:

```
$ cd /my_project_folder                  //go to the folder directory !!!
$ cd env/lib/python3.5/site-packages/
```

```
$ ln -s /usr/local/lib/python3.5/site-packages/cv2.cpython-34m.so cv2.so
```

Notice changing the name from cv2.cpython-34m.so to cv2.so — now Python can import our OpenCV bindings using the name cv2.

This one OpenCV step follows some step from online OpenCV downloading and environment preparation tutorial.

Test OpenCV installation

In terminal:

```
$ cd /my_project_folder                    //go to the folder directory !!!
$ source env/bin/activate                  //use this command only if your virtual
environment is not activated
(env) $ python
>>> import cv2
>>> cv2.__version__
```

OpenCV help: Some OpenCV tutorials for python are available for easier start with OpenCV in Python programming language.

6. VTK

-version: vtk 7.1.1

Download corresponding version from VTK wensite

Build VTK...in terminal:

```
$ mkdir vtk
$ cd vtk
$ ccmake path_to_downloaded_folder
```

Now press c to configure (YOU CAN HAVE SOME IMPORT ERRORS...THEN YOU MUST INSTALL). After configuring several options should appear. Press t to toggle to Advanced mode and change the following:

```
BUILD_TESTING on
VTK_WRAP_PYTHON on
VTK_WRAP_TCL on
Change CMAKE_INSTALL_PREFIX to /usr
Change VTK_PYTHON_VERSION to 3.5 (your python version)
Change PYTHON_EXECUTABLE to your python executable. For me it was
/usr/local/bin/python . You can check it by typing which python in your another
terminal.
Change PYTHON_INCLUDE_DIR to the directory where the python libraries are installed.
In my case it is /usr/lib/python3.5/
```

Press c to configure. Now press g to generate Makefile. (You may have to press c and g again as it sometimes does not work properly)

Then install VTK and the Wrappers in terminal:

```
$ make
$ sudo make install
$ cd Wrapping/Python
```

```
$ make
$ sudo make install
```

Set PYTHONPATH environment variable for your computer (not for your virtual environment), use terminal:

```
$ export
PYTHONPATH=/home/zuzana/vtk/lib:/home/zuzana/vtk/Wrapping/Python:$PYTHONPATH
```

Test vtk installation

```
(env) $ python
>>> import vtk
>>> print(vtk.vtkVersion.GetVTKSourceVersion())
```

If you want set PYTHONPATH permanently in your computer, use terminal and type:

```
$ cd
```

```
$ gedit .bashrc
```

and then add row: export
PYTHONPATH=/home/zuzana/vtk/lib:/home/zuzana/vtk/Wrapping/Python:$PYTHONPATH

Probably it will be necessary to restart your computer. You can also add the same row into your virtual environment into file activate (located in your_project_folder/env/bin/), but when you run code from IDE not from console, you must do next step.

SET PATHS FOR VIRTUAL ENVIRONEMNT IN YOUR INTERPRETER (for Pycharm):

- select Settings > Project Interpreter
- to the right of interpreter selector there is an icon button, click it and select "More..."
- pop up a new "Project Interpreters" window select right bottom button (named "show paths for the selected interpreter")
- pop up a "Interpreter Paths" window
- click the "+" buttom > select your desired PYTHONPATH directory (the folder which contains python modules) and click OK Done! Enjoy it!

7. SimpleITK

Activate virtual environament and run command:

```
$ cd your_project_path
$ source env/bin/activate
(env) $  pip install --trusted-host itk.org -f
https://itk.org/SimpleITKDoxygen/html/PyDownloadPage.html SimpleITK
```

T

Appendix C: User guide for visualization tool

Visualization tool is a simple desktop application. It has only one main window as shown in the Figure 109. It contains:

- three 2D views which visualize data in axial, coronal and sagittal direction,
- one canvas with 3D model of the brain and tumor,
- histogram of the health brain and tumor and
- some check boxes for setting different properties of the visualizations depending on the user preferences.

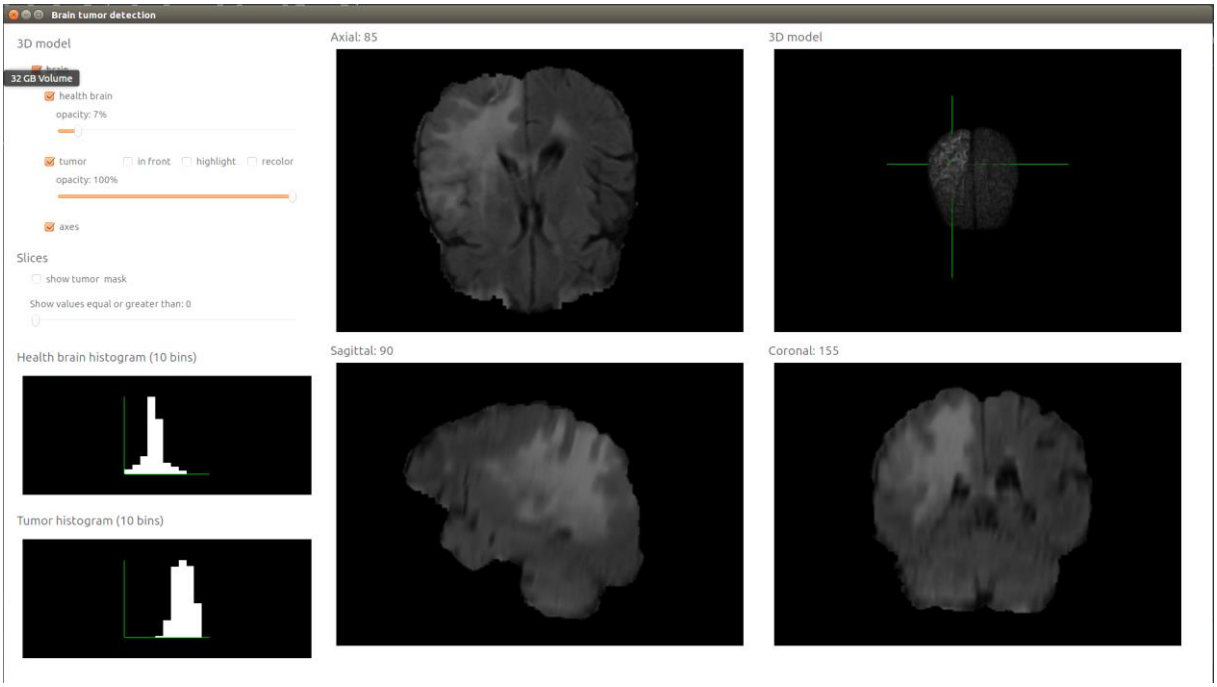


Figure 109: Initial window of visualization tool

Application can visualize different volumes from one patient in different phase of processing, but this can be still set only programmatically in the *main* function. One of the following data type can be chosen for the visualization:

- different examination or different modality and
- original volumes, preprocessed volumes or transformed volumes.

Interaction allows you to:

- zoom, translate and scroll through slices,
- zoom and rotate 3D model,

- zoom histograms.

In the application can also be set different configurations of the visualizations using check boxes on the left side of the window:

- in canvases which contains 2D slice in different directions:
    - show or hide tumor masks,
    - filter values to be displayed in the slices.
- in 3D model:
    - show or hide tumor or whole brain,
    - change opacity of the tumor or whole brain,
    - change visualization setting of the tumor:
        - move tumor in frond – paint tumor 3D model above the brain model
        - highlight tumor – it changes the brightness of the tumor
        - recolor tumor – it changes the color of the tumor
    - show or hide axes

Appendix D: Work plan

project	week	task	State
DP1	1	analyze computer vision and preprocessing	Done
	2	analyze segmentation	Done
	3		Done
	4	analyze registration	Done
	5		Done
	6	implement segmentation	Done
	7		Done
	8	test segmentation	Done
	9	rigid registration	Done
	10		Done
	11		Done
	12	evaluate rigid registration	Done
DP2	1	preprocessing	Done
	2	non-rigid registration	Done
	3		Done
	4		Done
	5	evaluate non-rigid registration	Done
	6	tumor changes visualization	Done
	7	image morphing	Done
	8	visualization tool	Done
	9		Done
	10		Done
	11	documentation	Done
	12		Done

Appendix E: Disc content

Basic structure of the folders and files on the disc:

- data folder
    - non-rigid_dataset_example-BRATS2015 folder – several examples of the data used for non-rigid registration (from BRATS 2015 dataset)
    - rigid_dataset_example-Siemens folder – several examples of the data used for rigid registration (from Siemens dataset)
- diagrams folder
    - diagrams_activity.vsdx – activity diagrams used in the work
    - diagrams_calss.vsdx – class diagrams used in the work
    - diagrams_sequence.vsdx – sequence diagrams used in the work
- evaluation folder
    - result_nonrigid_configurations.xlsx – evaluation, graphs, charts and statistical evaluation of non-rigid registration for different configuration
    - result_nonrigid_modalities.xlsx – evaluation, graphs, charts and statistical evaluation of rigid registration for different modalities
    - results_rigid_registration.xlsx – evaluation, graphs, charts and statistical evaluation of rigid registration for different configurations
- sources folder – all source codes developed during the work
- visualization_tool_screenshots – more screenshots of the visualization tool
- dp_xbobotova.pdf - documantation

z

Appendix E: Summary - Resumé

1. Úvod

Medicína je oblasť, ktorá stále vyvíja. Je dôležitá pre ľudí a ich zdravie. Na diagnostikovanie ochorení sa využíva mnoho rôznych techník. V našom výskume pracujeme na metódach vhodných na monitorovanie vývoja nádoru na mozgu. Najčastejšie používaným testom na diagnostiku nádorov je zobrazovanie pomocou magnetickej rezonancie. Výsledkom testu magnetickej rezonancie mozgu je séria obrazov nazývaných rezy. V súčasnosti sa počítače používajú stále viac počas diagnostikovania pacientov. Poskytujú lekárom viac príležitostí a môžu im pomôcť pracovať efektívnejšie.

Spracovanie trojrozmerných lekárskych údajov (v našom prípade MRI) môže byť časovo náročná a nepríjemná úloha. Počítačové videnie je silný nástroj na riešenie mnohých zaujímavých problémov a môže pomôcť pri automatizácii monotónnych a rutinných úloh v mnohých oblastiach. Lekárstvo je jednou z oblastí, v ktorých je počítačové videnie veľmi užitočné a môže byť veľmi silným nástrojom.

V skutočnosti existuje veľa úloh v medicíne a diagnostike, kde niektoré algoritmy môžu uľahčiť prácu špecialistom. Na vyriešenie jednej konkrétnej úlohy sa vyvíjajú vždy špecializované metódy počítačového videnia. Preto sme sa rozhodli zamerať sa na mozog a rakovinu v mozgu - nádory. V našej práci prezentujeme metódy, ktoré sú schopné pomôcť s diagnózou rakoviny v mozgu, a to hlavne s cieľom sledovať zmeny nádoru.

Pri spracovaní dát z MRI bez metód počítačového videnia, napríklad pri segmentácia nádoru z 3D dát, špecialista musí segmentovať nádor na každom reze. To je časovo náročná a monotónna úloha, ale je to naozaj nevyhnutné a veľmi užitočné - hlavne počas procesu liečby ožiarením, keď by sa zdravé časti mozgu mali pred žiarením čo najviac chrániť.

Hlavnou úlohou práce bolo nájsť automatické metódy na sledovanie zmien ochorenia pacienta. Aby to bolo možné, je potrebné segmentovať nádor z MRI, zaregistrovať MRI z rôznych vyšetrení a nájsť korešpondujúce časti mozgu a nádoru v rôznych vyšetreniach zachytených počas liečby v pravidelných časových intervaloch. Dnes existuje veľa prístupov vhodných na segmentáciu nádoru. Rozhodli sme sa niektoré analyzovať a otestovať. Na registráciu údajov z rôznych vyšetrení sme navrhli metódu založenú na rigidnej registrácii. Nakoniec sme navrhli metódu založenú na nerigidnej registrácii, ktorá vie nájsť korešpondencie rovnakých častí mozgu a tumoru medzi dvoma vyšetreniami pomocou algoritmov založených na optickom toku.

V nasledujúcich kapitolách uvádzame krátky úvod do problematiky predspracovania dát, segmentácie a registrácie v časti 2 Analýza, návrh našej metódy a popis jednotlivých krokov v časti 3 Návrh metódy a výsledky. V tejto kapitole opisujeme aj dosiahnuté výsledky. A v poslednej časti 4 Záver sa nachádza krátke zhrnutie.

2. Analýza

Počítačové videnie je oblasť, ktorá dokáže porozumieť digitálnym obrazom, videám a iným digitálnym údajom. Je dôležité uvedomiť si, že počítač vidí údaje inak než my. Obraz je len súborom hodnôt a ak chceme získať nejaké informácie, je nevyhnutné vyvinúť nejaké metódy a algoritmy.

Počítačové videnie je používané v rôznych vedeckých disciplínach, ale aj v každodennom živote. V súčasnosti ho môžeme nájsť aj v mnohých funkciách automobilov, v priemysle sa používajú na kontrolu kvality, v mnohých monitorovacích systémoch na kontrolu situácie, zabránenie nejasnostiam alebo odhalenie niektorých udalostí. Fyzika, biológia, chémia, vesmírny výskum alebo robotika sú oblasti, v ktorých má počítačové videnie veľa aplikácií.

Medicína je ďalšia oblasť, v ktorej počítačové videnie môže uľahčiť ľudskú prácu a pomôcť rôznymi spôsobmi. V medicíne je veľa testov, kde sa používajú rôzne skenery a snímače. Známe testy sú napríklad magnetická rezonancia (MRI), počítačová tomografia (CT), sonografia, röntgenové vyšetrenie atď. Metódy počítačového videnia môžu pomôcť lekárom a diagnostikom spracovávať údaje a uľahčiť tak ich prácu.

Predspracovanie dát

Predspracovanie je prvým krokom po získaní obrazových dát. V medicíne sa stretávame s údajmi, ktoré nie sú dokonalé, takže je dôležité aplikovať niektoré techniky, aby sme ich vylepšili. Dáta po predspracovaní môžu obsahovať jasnejšie a lepšie okraje objektov. Hlavným cieľom predspracovania je zlepšiť kvalitu údajov.

Obrázky je možné spracovať v priestorovej alebo frekvenčnej oblasti. Najbežnejšie techniky vylepšenia kvality obrázkov sú:

- zvýšenie kontrastu,
- vylepšenie rozlíšenia,
- zvýraznenie hrán a
- redukcia šumu.

Segmentácia

Segmentačné metódy rozdeľujú obrazové dáta na niekoľko častí v závislosti od nejakých vlastností, ako napríklad tvar, farba, intenzita alebo textúra. Segmentácia sa zvyčajne používa na detekciu nejakého objektu na obrázku. V súčasnosti existuje veľa segmentačných metód vhodných na rôzne typy problémov.

Dostupná literatúra rozdeľuje segmentačné prístupy na manuálne, poloautomatické a automatické. Manuálna segmentácia nevyžaduje žiadne metódy počítačového videnia. Objekty sú segmentované odborníkom ručne pomocou nejakého špeciálneho softvéru a nástrojov. Je to časovo náročná a monotónna úloha. Poloautomatické metódy vyžadujú určitú interakciu s používateľom, ale sú rýchlejšie než manuálna segmentácia. Automatické metódy nevyžadujú žiadne vstupy od používateľa, všetko sa vypočíta podľa segmentačného algoritmu.

V medicíne sa segmentačné algoritmy používajú na detekciu niektorých špecifických orgánov a častí tela alebo na detekciu niektorých anomálií na tele, ako sú nádory. Existuje niekoľko prístupov, ako získať údaje o štruktúre tela.

Metódy segmentácie je možné rozdeliť na niekoľko skupín na základe ich prístupu k segmentácii. Prieskumy sumarizujúce odborné články ich rozdeľujú na:

- Metódy založené na hranách - jeden z najobľúbenejších prístupov. Často sa používajú na usmernenie segmentácie, umožňujú extrahovať rôzne typy hrán: parametrické, intenzívne, textúrové a okrajové. Medzi najznámejšie algoritmy tohto typu patria algoritmus rez grafom a algoritmus aktívnych kontúr.

- Metódy založené na regiónoch – segmentujú vstupné dáta na základe podobnosti hodnôt v jednotlivých regiónoch obrazov. Najjednoduchším algoritmom je prahovanie, ktoré na základe 1 vstupnej konštanty vie rozdeliť obraz na regióny s nižšou hodnotou a vyššou hodnotou oproti tomuto prahu. Medzi ďalšie známe algoritmy patrí metóda rastúcich regiónov a záplavový algoritmus.

- Strojové učenie – jednoducho povedané robí počítače schopné učiť sa bez explicitného programovania. Počítač sa môže naučiť niektoré modely a vykonávať segmentácie v závislosti od modelov na základe podobných podmienok na vstupnom obrázku alebo rozdeliť obraz na časti - segmenty v závislosti od niektorých charakteristík. Základné rozdelenie techník strojového učenia je na:

    - učenie bez učiteľa (klastrovanie) – metóda, ktorá rozdeľuje pixely obrazu na niekoľko klastrov pričom nie je potrebný žiadny tréning.

    - učenie s učiteľom (klasifikácia) – umožňuje natrénovať model na nejakých vstupných dátach, ktorý vie potom následne klasifikovať podobné dáta na základe tohto naučeného modelu.

    - neurónové siete – dnes najčastejšie používaný prístup, ide v podstate o učenie s učiteľom, kedy sa taktiež na základe vstupných dát trénuje nejaký model.

Registrácia

Proces, ktorý môže zarovnať dva alebo viac obrázkov (referenčný a snímaný) tej istej scény, sa nazýva registrácia. Pri registrácii predpokladáme, že pre nejaké pixely z prvých vstupných dát sa nachádza pozícia zodpovedajúcich pixelov v nasledujúcich vstupných dátach. Obrázky môžu byť snímané rôznymi snímačmi z rôznych uhlov pohľadu alebo môžu byť naskenované v rôznych časoch alebo v rôznych podmienkach. Registrácia je nevyhnutným krokom na analýzu a získavanie informácií z kombinácie rôznych zdrojov údajov.

Z pohľadu, aké obrazy sú registrované, môže byť registrácia rozdelená do štyroch skupín:

- registrácia rôznych výrezov - cieľom je získať väčšie 2D zobrazenie alebo 3D zobrazenie scény

- registrácia údajov získaných v rôznych časových obdobiach vždy aj s rôznymi podmienkami - cieľom je zistiť zmeny medzi akvizíciami

- registrácia údajov získaných rôznymi senzormi, nazývaná tiež multimodálna analýza - cieľom je pracovať s viacerými informáciami získanými z rôznych senzorov

- registrácia modelu a scény, kde model predstavuje počítačové zobrazenie scén ako sú atlasy - cieľom je nájsť lokalizáciu scény v modeli a porovnať ju

Zvyčajne sa proces registrácie údajov skladá zo štyroch základných krokov:

- Detekcia príznakov - na obrázku sa hľadajú nejaké výrazné objekty. Obyčajne sú reprezentované ako pole bodov nazývané kľúčové body. Existuje veľa možností detekcie funkcií, ale hlavným predpokladom je, že by mali byť ľahko detekovateľné. Z pohľadu kritéria, na základe ktorého zhody hľadáme ich môžeme rozdeliť na geometrické, senzorové, založené na intenzite a hybridné. Z pohľadu ako pracujú môžu byť rozdelené na metódy založené na regiónoch alebo metódy založené na príznakoch.

- Hľadanie zhôd medzi príznakmi - hľadanie zhody medzi príznakmi detekovanými na snímanom obrázku a príznakmi získanými z referenčného obrázku. Problémom je, že ak sme v predchádzajúcom kroku zistili nejaké nesprávne príznaky, rozšíri sa do tohto kroku. Ďalším problémom je, že v skutočnosti korešpondujúce príznaky môžu byť odlišné, pretože obrázky môžu byť skenované s rôznymi podmienkami. Tento problém by sa mal vyriešiť s dobrou voľbou deskriptora príznakov. Na jednej strane musí byť deskriptor schopný rozlišovať medzi rôznymi znakmi, ale na druhej strane nemôže byť ovplyvnený šumom a neočakávanými zmenami príznakov.

- Odhad transformačného modelu je ďalším krokom registrácie. Korešpondujúce príznaky sa v tomto kroku používajú na výpočet transformačného modelu. Autori článkov rozdeľujú metódy z deformačného hľadiska na rigidnú a nerigidnú transformáciu nazývanú aj elastická transformácia. Rigidná registrácie nedeformuje dáta, umožňuje len škálovanie, rotáciu, reflexiu a transláciu. Na druhej strane nerigidná registrácia deformuje štruktúru dát a objektov, tak aby sa výstup čo najviac podobal na referenčný obraz.

- Transformácia – na záver sa transformačné matice vytvorené v predošlom kroku použijú na transformáciu dát, čím vlastne dáta zaregistrujeme.

3. Návrh metódy a výsledky

Hlavným cieľom práce bolo navrhnúť a implementovať automatické metódy vhodné na sledovanie zmien nádoru. Keďže MRI je najbežnejším testom pre diagnostiku rakoviny mozgu, navrhované metódy pracujú so vstupnými MRI dátami - vyšetrenia od jedného pacienta zachyteného v pravidelných časových intervaloch.

MRI snímky mozgu sú trojrozmerné dáta. To znamená, že pozostávajú z niekoľkých rezov. Počas liečby rakoviny mozgu musí pacient podstúpiť niekoľko vyšetrení v pravidelných časových intervaloch. Lekári musia vyhodnotiť a kontrolovať všetky výsledky vyšetrení pacientov, aby sledovali priebeh liečby. Môže to byť naozaj veľmi časovo náročná a nepríjemná úloha, takže počítačové videnie môže byť v tejto oblasti veľmi užitočné a automatizovať niektoré monotónne úlohy.

Existujú určité problémy, ktoré je potrebné vziať do úvahy:

- Keďže MRI sa vykonávajú v inom čase, môžu byť inak otočené. Je potrebné otočiť dáta do rovnakej pozície, aby sme mohli na sledovať zmeny. To je úloha pre rigidnú registráciu.

- V priebehu času sa mení nádor a tiež mozog. Nádor môže rásť alebo zmenšovať sa, čo tiež ovplyvňuje štruktúru okolitých zdravých častí mozgu. Na sledovanie zmien ochorenia je dôležité vedieť, ako bol mozog deformovaný nádorom. To je úloha pre nerigidnú registráciu.

- Dáta sa zaznamenávajú s rôznymi zariadeniami alebo len s rôznymi nastaveniami jedného zariadenia a je bežné, že nie sú konzistentné. Hodnoty tých istých častí mozgu môžu byť odlišné. Tým sa musíme zaoberať ešte pred registráciou a tiež nám to pomôže vytvoriť lepšiu vizualizáciu. Preto musíme dáta predspracovať.

Navrhli sme vlastnú metódu na sledovanie zmien tumoru a mozgu. Vstupom pre metódu sú dáta z MRI testu, ako bolo spomenuté. Ide o 3D dáta, takže ak chceme pracovať len s rezmi, musíme ich vytvoriť zo vstupných dát. V práci používame údaje z datasetu BRATS 2015 a údaje z praxe od spoločnosti Siemens.

Prvým krokom navrhnutej metódy je prepracovanie. Musíme vyriešiť odlišný jas a kontrast dát zachytených v inom čase. Je to dôležité najmä pre vizualizáciu, ale môže to byť užitočné pre celý proces.

Druhým krokom je segmentácia nádoru. Musíme poznať hranice nádoru, aby sme mohli sledovať jeho zmeny. Je to tiež dôležité pre ďalší krok – pre rigidnú registrácia. Zmenené časti nádoru a okolitého mozgu môžu spôsobiť chyby v rigidnej registrácii, ak by neboli segmentované.

Ďalším krokom je rigidná registrácia. Potrebujeme otočiť dáta do rovnakej pozície s využitím translácie, rotácie a prípadne aj škálovania.

V ďalšom kroku musíme urobiť nerigidnú registráciu (nazývanú aj elastická), aby sme mohli sledovať zmeny zdravých častí mozgu a tiež nádoru. Zmeny spôsobené nádorom sú nepravidelné, takže je to perfektná úloha práve pre nerigidnú registráciu.

Nakoniec, aby sme mohli vizualizovať dáta vytvorili sme jednoduchú desktopovú aplikáciu.

Predspracovanie

Dáta z rôznych vyšetrení môžu mať rôzne hodnoty tých istých častí mozgu. Je to preto, že boli zachytené v rôznych časoch a s rôznymi nastaveniami zariadenia. Môžu mať tiež odlišný kontrast a jas. Preto sme navrhli metódu na predspracovanie týchto dát. Pozostáva z troch krokov: odstránenie percentilu, škálovanie údajov a algoritmus mapovania histogramov.

Odstránenie nejakého percentilu dát znamená, že nájdeme hodnotu určeného horného a dolného percentilu (my sme používali horný percentil 99 a dolný percentil 5) a hodnoty vyššie ako horný percentil nahradíme hodnotou tohto percentilu a naopak hodnoty nižšie ako dolný percentil nahradíme hodnotou dolného precentilu.

Pre škálovanie dát sme vyžili vzorec min max normalizácie a dáta sme preškálovali na rozsah 0-255:

$$data = \frac{\text{data} - \min(\text{data})}{\max(\text{data}) - \min(\text{data})} * 255$$, kde data sú n-dimenzionálne pole (2D alebo 3D)

Algoritmus mapovania histogramov je jednoduchý algoritmus, ktorý je založený na tom, že histogram jedných vstupných dát nazývaných referenčné dáta sa mapuje na histogram iných vstupných dát nazývaných cieľové dáta.

Výsledky testovania ukazujú, že metódy vhodne predspracujú dáta, pretože jas a kontrast dát po spracovaní je viac vyvážený ako pred spracovaním.

Segmentácia

Segmentácia nádorov pomocou počítačového videnia je problémom, ktorý je čato skúmaný odbornou komunitou. Riešenie tohto problému je založené na veľkej motivácii – pomôcť lekárom a rádiológom s manuálnou a monotónnou prácou, ktorú musia robiť pri diagnostike.

V skutočnosti je segmentácia nádorov z MRI vyšetrení mozgu ťažká úloha, pretože hranice nie sú vždy jasné a niekedy nádory nie sú výrazné. Veľa ľudí už riešilo daný problém a preto existuje mnoho prístupov v oblasti segmentácie mozgových nádorov s veľmi dobrými výsledkami. Preto sme sa rozhodli implementovať jednu z najnovších metód, ktorá dosiahla najlepšie výsledky na datasete BRATS 2013. Ide o metódu od Havaei a kol. [24] - Segmentácia nádorov mozgu s použitím hlbokých neurónových sietí.

Jedným z dôvodov, prečo sme sa rozhodli vybrať tento prístup, bolo, že autor dosiahol veľmi dobré výsledky ako je uvedené v článkoch a druhý z nich bol, že zdrojový kód je k dispozícii online [81]. Počas práce sme však narazili na problém a to, že zdrojový kód využíva staré verzie niektorých knižníc a nikde sa neuvádza, že aké. Preto bolo potrebné niektoré časti kódu re-implementovať a prispôsobiť na nové verzie knižníc. Danú metódu sme následne testovali, avšak nepodarilo sa nám dosiahnuť uspokojivé výsledky a preto sme sa v ďalších krokoch rozhodli radšej použiť segmentačné masky z datasetov.

Rigidná registrácia

Ďalším krokom nami navrhnutej metódy je rigidná registrácia. Keďže dáta z MRI pochádzajúce od jedného pacient z rôznych vyšetrení sa robia v úplne inom čase, tak môžu byť inak otočené, preložené alebo zmenšené. Ak chceme sledovať zmeny nádoru a mozgu, musíme zarovnať dáta do rovnakej pozície. To je dokonalá úloha pre rigidnú registráciu.

Nami navrhnutá rigidná registrácia pozostáva zo šiestich krokov. Vstupom metódy sú dva 2D rezy z 3D MRI dát. V prvom kroku vytvoríme registračnú masku pre oba vstupné obrázky. Navrhli sme algoritmy na vytvorenie dvoch rôznych typov másk a to masky zdravého mozgu a masky lebky. Na vytvorenie oboch másk využívame prahovanie nasledované morfologickými operáciami, čím dostaneme masku celého mozgu. Aby sme vytvorili masku len zdravej časti mozgu, tak ju erodujeme a následne odčítame ešte masku tumoru. Aby sme vytvorili masku lebky, tak nájdeme kontúru pôvodnej masky, kontúru zväčšíme a tým dostaneme približnú masky lebky.

Masky sa v ďalšom kroku používajú na nájdenie kľúčových bodov v oboch vstupných obrázkoch. V tomto kroku využívame 4 rôzne algoritmy: SIFT, SURF, FAST a ORB. Následne pre všetky nájdené kľúčové body vypočítame vektory príznakov pomocou deskriptorov SIFT, SURF alebo ORB.

Teraz máme vektory pre oba vstupné obrazy a v ďalšom kroku nájdeme medzi nimi zhody. V tomto kroku, okrem správnych zhôd, tiež získame aj nesprávne. Preto ich odfiltrujeme pomocou

jednoduchých podmienok založených na predpoklade, že dáta nie sú príliš rôzne otočené, zmenšené alebo posunuté.

Následne z odfiltrovaných zhôd vypočítame transformačnú maticu pre transformáciu a použijeme ju na transformáciu a registráciu jedného zo vstupných obrázkov k inému.

Počas testovania sme skúšali rôzne konfigurácie rigidnej registrácie – skúšali sme využiť rôzne masky pre registráciu (buď zdravej časti mozgu alebo lebky), rôzne algoritmy na získanie kľúčových bodov a tiež výpočet vektorov príznakov.

Aby sme overili správnosť registrácie porovnali sme navzájom dva vstupné obrázky a porovnali sme ich s výstupnými zaregistrovanými a taktiež sme porovnali zaregistrované obrázky s originálmi (pôvodne zaregistrovanými obrázkami z datasetu). Neporovnávali sme celé rezy, ale vytvorili sme metódu, ktorá postupne z výsekov tých častí rezov, kde sa nachádza aspoň nejaká časť mozgu, vypočíta histogramy a určí korelácie, prieniky a Bhattacharryouvu vzdialenosť medzi týmito histogramami.

Výsledky ukázali, že registračná maska vytvorená z lebky bola pre našu metódu problematická. Naša metóda mala pri využití masky lebky problém s nájdením afinnej matice pre registráciu. Mali sme s využitím takejto masky aj niektoré veľmi úspešné registrácie, ale vo všeobecnosti metóda nefungovala dobre.

Navrhnutá metóda funguje lepšie, keď registrujeme rezy s využitím masky zdravými mozgovými častí a väčšina rezov bola úspešne zaregistrovaná. Najlepšie výsledky sme dosiahli s využitím algoritmu ORB pre hľadanie kľúčových bodov a tiež deskripciu príznakov. Porovnateľné výsledky sme dosiahli aj s využitím kombinácia FAST detektora kľúčových bodov a SURF deskriptora. Existujú však aj prípady, kedy metóda úspešne nezaregistrovala rezy. Zlyhanie metódy bolo spôsobené nesprávne nájdenými zhodami medzi vektormi príznakov a teda aj nesprávnou filtráciou zlých zhôd.

Nerigidná registrácia

Nerigidná registrácia je rozhodujúcim krokom pre pozorovanie zmien mozgu spôsobených nádorom. Nádor rastie a mení sa nepravidelne v závislosti od okolitých zdravých mozgových štruktúr čím tlačí na tieto štruktúry. Vďaka nerigidnej registrácii vieme sledovať zmeny zdravých častí mozgu, ale aj nádoru. Existuje niekoľko prístupov, ktoré sa pokúšajú riešiť podobný problém. Mnohé z nich používajú atlasy mozgu a sledujú rôzne ciele, ako je segmentácia nádoru alebo registrácia snímok pred operáciou a po rekurencii. V našej práci sme sa rozhodli implementovať a otestovať iné prístupy, pretože náš cieľ je trochu iný.

Vstupom metódy sú 3D dáta. Používame však algoritmy z knižnice, ktoré dokážu spracovať len 2D obrázky, takže naša metóda aj napriek tomu funguje po rezoch. V prvom kroku metóda nájde optický tok pomocou algoritmu Farneback alebo Lukas-Kanade. Základným predpokladom týchto algoritmov je, že pohyb susedných pixelov je rovnaký a intenzity objektov sa medzi dvomi nasledujúcimi snímkami veľmi nezmenia (aj preto sme museli dáta správne predspracovať). Tieto algoritmy sa často používajú na sledovanie pohybu v jednotlivých snímkach videa. Sú vhodné pre náš problém, pretože neočakávajú veľké zmeny medzi snímkami a to je presne to, čo potrebujeme.

V ďalšom kroku vypočítame zodpovedajúce body z toku. To je dôležité najmä pre algoritmus Farneback, pretože jeho výsledkom je hustý tok. Lukas-Kanade vráti súbor bodov a mi v tomto kroku

len zmeníme ich štruktúru pre finálny krok, ktorým je vizualizácia. Testovali sme rôzne prístupy k vizualizácii:

- vykreslenie zmien,
- vytvorenie HSV obrázka z hustého toku a
- morfovanie rezov.

Vykreslenie zmien je jednoduchý algoritmus, ktorý len nakreslí šípky v smere pohybu jednotlivých korešpondujúcich bodov. Vytvorenie HSV (hue - odtieň, saturation – sýtosť, value – hodnota) obrázka z hustého toku funguje tak, že z toku vypočítame uhol pohybu korešpondujúcich bodov a vzdialenosť medzi týmito bodmi a vytvoríme HSV obrázok tak, že uhol bude hodnota odtieňa a vzdialenosť hodnota hodnoty. Saturáciu sme nastavili konštantne na 255.

Posledným prístupov je morfovanie rezov. Je to založené na vytvorení animácie z rezov a metóda nám umožňuje vytvoriť nové rezy, ktoré sú priemerom medzi existujúcimi a teda ich v animácii umiestnime medzi existujúce. Tým vznikne krajšia a vyhladenejšia animácia. Pri morfovaní využívame algoritmu Delaunayovej triangulácie, rigidnú transformáciu vzniknutých trojuholníkov a alfa blending.

Aby sme mohli nerigidnú registráciu testovať, museli sme transformovať jeden z rezov. Využili sme na to metódu podobnú morfovaniu až na to, že v poslednom kroku sme nevytvárali nový rez, ale transfromovali sme jeden z rezov na ten druhý cez trojuholníky vypočítané pomocou Delaunayovej triangulácie. Následne sme podobne ako pri testovaní rigidnej registrácie vyhodnotili histogramovaú koreláciu, prienik a Bhattacharryouvu vzdialenosť.

Testovali sme, že aký vplyv má na nerigidnú registráciu využitie rôznych modalít a tiež využitie rôznych algoritmov a parametre týchto algoritmov. Zistili sme, že rôzne modality majú vplyv na úspešnosť a najlepšie výsledky sme dosiahli s použitím Flair modality a porovnateľne aj s použitím T2. Pri porovnávaní algoritmov sme zistili, že pre algoritmus Lukas-Kanade je toto ťažká úloha, pretože algoritmus sa len snaží nájsť zmeny v rezoch a tie nie sú veľmi výrazne. Úspešní sme boli s využitím Farnebackovho algoritmu, práve vďaka tomu, že vracia hustý tok avšak zistili sme, že parametre metódy nemajú výrazný vplyv na úspešnosť registrácie. Jedine, keď sme použili menší krok pri výpočte korešpondujúcich bodov z toku, tak sme zistili, že registrácia je úspešná vo viacerých prípadoch ako pri použití väčšieho kroku.

Nástroj na vizualizáciu 3D MRI dát

Aby bolo možné vizualizovať dáta, ktoré používame a spracúvame, vytvorili sme jednoduchý nástroj na vizualizáciu týchto 3D dát. Tento nástroj dokáže spracovať 3D dátový formát a vizualizovať snímky MRI v 2D a 3D pohľade. 2D pohľady sú tri a umožňujú pohľad na jednotlivé rezy v axiálnom, sagitálnom a koronálnom smere. 3D pohľad na dáta sprostredkúvame na základe vytvoreného 3D modelu pomocou algoritmu ray casting – umožňuje vidieť mozog alebo tumor. Aplikácia poskytuje tiež jednoduchú interakciu. Umožňuje prechádzanie cez rezy vo všetkých smeroch, ich približovanie a posúvanie, približovanie, posúvanie a otáčanie 3D modelu, zapnutie / vypnutie vizualizácie segmentácie nádoru v 2D aj 3D pohľade a tiež zobrazuje osi v 3D pohľade, ktoré ukazujú, kde sa práve zobrazené rezy v mozgu nachádzajú. V 3D modeli je tiež možné nastaviť priehľadnosť mozgu alebo aj tumoru, aby sa dala lepšie sledovať jeho lokalizácia.

4. Záver

Hlavným cieľom práce bolo vyvinúť metódy vhodné na sledovanie zmien mozgu a nádorov spôsobených rastom nádoru. Na sledovanie zmeny nádoru musíme segmentovať nádor, registrovať rôzne vyšetrenia od jedného pacienta a zaznamenať zmeny medzi nimi. V práci sme navrhli metódu vhodnú na sledovanie nádoru, ktorá pozostáva z piatich krokov.

Prvým krokom je predspracovanie údajov. To je dôležité, pretože vyšetrenia od jedného pacienta majú rozdielne rozsahy dát a rozdielny jas. Na vyváženie jasu vyšetrení od jedného pacienta sme použili algoritmus mapovania histogramov. Výsledky sú veľmi sľubné.

Druhým a dôležitým krokom je metóda segmentácie nádoru. Musíme segmentovať nádor, aby sme mohli sledovať zmeny tumoru a zmeny zdravého mozgu. V našom prístupe sme sa rozhodli opätovne implementovať jednu z najlepších metód. Vybraná metóda bola založená na kaskádovitých konvolučných neurónových sieťach. Počas testovania sme však nedosiahli uspokojivé výsledky, takže v nasledujúcich krokoch sme sa rozhodli používať nádorové masky z datasetov. Hranice nádoru sú rozhodujúce pre správnosť rigidnej registrácie a môžu byť veľmi užitočné pre nerigidnú registráciu.

Tretím krokom je rigidná registrácia dát z rôznych vyšetrení, ale od jedného pacienta. Počas rigidnej registrácie musíme počítať iba z oblasťami mozgu bez nádoru, pretože nádor sa časom mení. Použili sme základný tok registrácie, ktorý pozostáva z niekoľkých krokov: vytváranie masky pre registráciu, nájdenie kľúčových bodov, výpočet vektorov príznakov, nájdenie zhôd medzi vektormi, filtrácia zhôd, nájdenie matice pre transformáciu a nakoniec transformácia dát. Navrhovaná metóda môže spracovať 2D rezy. Metódu sme testovali na datasete, ktorý obsahuje registrované a neregistrované 2D rezy vytvorené zo Siemens datasetu. Testovali sme rôzne konfigurácie metódy s využitím rôznych segmentačných masiek, rôznych detektorov kľúčových bodov a deskriptorov príznakov. Výsledky sú sľubné. Najlepšie výsledky sme dosiahli s využitím registračnej masky zdravých častí mozgu, ORB kľúčového detektora a deskriptora.

Zväčšenie nádoru spôsobuje aj zmeny v štruktúre mozgu, pretože nádor sa časom mení. To spôsobuje nepravidelné zmeny mozgu, takže je to perfektná úloha pre štvrtý krok – nerigidnú registráciu. Pre tento problém sme navrhli metódu založenú na optickom toku. Testovali sme dva rôzne algoritmy: Farneback a Lukas-Kanade, ale algoritmus Farneback pracoval lepšie oproti Lukas-Kanade, vďaka tomu, že Farneback počíta hustý tok. Metódu sme testovali na datasete BRATS 2015 s použitím iba pacientov, ktorí mali viac ako jedno vyšetrenie. Počas vyhodnotenia sme museli jedno vyšetrenie nerigidne transformovať vzhľadom na druhé, aby sme vedeli medzi nimi vypočítať histogramové korelácie. Na to sme použili Delaunayovu trianguláciu a afinnú transformáciu. Testovali sme vplyv rôznych modalít a rôznych vstupných parametrov algoritmu Farneback. Najlepšie výsledky sme dosiahli pomocou modality Flair a porovnateľne aj T2. Avšak odlišná konfigurácia algoritmu Farneback nemala veľký vplyv na nerigidnú registráciu. Len použitie menšieho kroku pri výpočte zodpovedajúcich bodov z toku pomohlo dosiahnuť vyššiu koreláciu po transformácii vo viacerých prípadoch ako v prípade iných konfigurácií.

Posledným krokom je vizualizácia. Na vizualizáciu zmien nádoru sme zaviedli tri rôzne metódy: vykreslenie zmien čiarami, HSV obrázok zmien a morfovanie obrazu. Tiež sme vytvorili jednoduchý vizualizačný nástroj, kde používateľ môže vidieť MRI dáta v axiálnom, koronálnom a sagitálnom smere a tiež je dostupný aj 3D model mozgu a nádoru vytvorený pomocou algoritmu ray cating.