A Framework and Workflow for Hyper Resolution Image Visualization in Mineshaft Inspection

Master Thesis
of
Michael Brandner

November 2017

supervised by

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary

Chair of Automation
University of Leoben
Austria

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Leoben, am ______________________ ______________________

Michael Brandner

Acknowledgements

First of all, I want to express my gratitude to Professor Paul O'Leary who has been guiding me over the past few years, constantly sharing his experience and knowledge with me. He encouraged me to continually learn new skills, approach new solutions and critically question existing ones. Without him, I would not have the mindset that I have today.

I would like to especially thank my colleague Jakob König for supporting me over the last year. He always helped me, even if he had to set aside his own work and interests.

I want to thank my colleagues Michael Habacher, Roland Schmidt, Roland Ritt, Werner Kollment and Petra Hirtenlehner for helping me with any arising matters, may they have been of technical or organizational nature.

I could not thank my family enough for their support and their confidence throughout my studies.

Abstract

This thesis presents a framework and workflow for the automatic preparation of data and images to enable the use of hyper resolution images for mineshaft inspections. The focus of the thesis is on the process from image registration to visualization of image data in the dimensions of 10^6 x 10^5 pixels.

Concepts regarding image registration are presented, including phase correlation, non-rigid registration and homography. The thesis addresses the suitability of different web mapping libraries for building a visualization tool for deep mine shafts. Attention is focused on handling large volumes of data, resulting from the demand for a high image quality with a resolution of 1 pixel per millimetre for $2.5 \cdot 10^4\ \mathrm{m}^2$. A mapping tool is developed, using the Leaflet JavaScript library. The generation of a tile-layer as input for the mapping application is achieved, using GDAL for generic tiling of hyper resolution images. JSON is used as data exchange format and the JSON files provide meta-data associated with tag representations in the hyper resolution image. In order to guarantee consistency in connecting meta-data to pixel coordinates an example for the use of a common coordinate reference system is proposed. Finally, the functionality of the mapping application is presented and the framework is tested.

Index Terms

automatic shaft monitoring; hyper resolution images; web mapping applications; Leaflet

Kurzfassung

Diese Arbeit stellt ein Framework und einen Arbeitsablauf zur automatischen Vorbereitung von Daten und Bildern vor, um die Verwendung von hochauflösenden Bildern für die Inspektion von Minenschächten zu ermöglichen. Der Fokus der Arbeit liegt dabei auf dem Prozess von der Bildregistrierung bis zur Visualisierung von Bilddaten mit den Dimensionen von 10^6 x 10^5 Pixeln.

Phasenkorrelation, Non-Rigid-Registration und Homographie werden als Konzepte der Bildregistrierung vorgestellt. Die Arbeit befasst sich mit der Eignung unterschiedlicher Web-Mapping Bibliotheken für die Entwicklung eines Visualisierungswerkzeuges für tiefe Schächte. Besondere Aufmerksamkeit liegt auf der Handhabung großer Datenmengen, die sich aus der Forderung nach einer hohen Bildqualität mit einer Auflösung von 1 Pixel pro Millimeter für $2.5 \cdot 10^4$ m^2 ergibt. Leaflet, eine JavaScript Bibliothek, wird für die Entwicklung eines Mapping-Tools verwendet. GDAL wird für die generische Erzeugung eines Kachel-Layers aus hochauflösenden Bildern verwendet. Dieser Layer dient als Eingabe für die Mapping-Anwendung. JSON wird als Datenaustauschformat verwendet und die JSON-Dateien beinhalten Metadaten, die mit den Tag-Darstellungen im hochauflösenden Bild verknüpft sind. Um die Konsistenz bei der Verknüpfung von Metadaten mit Pixel-Koordinaten zu gewährleisten wird ein Beispiel für die Verwendung eines gemeinsamen Koordinatenreferenzsystems vorgeschlagen. Abschließend wird die Funktionalität der Mapping-Anwendung vorgestellt und das Framework getestet.

Schlagwörter

automatische Schachtüberwachung; hochauflösende Bilder; Web-Mapping Anwendungen; Leaflet

Contents

Chapter 1

Introduction

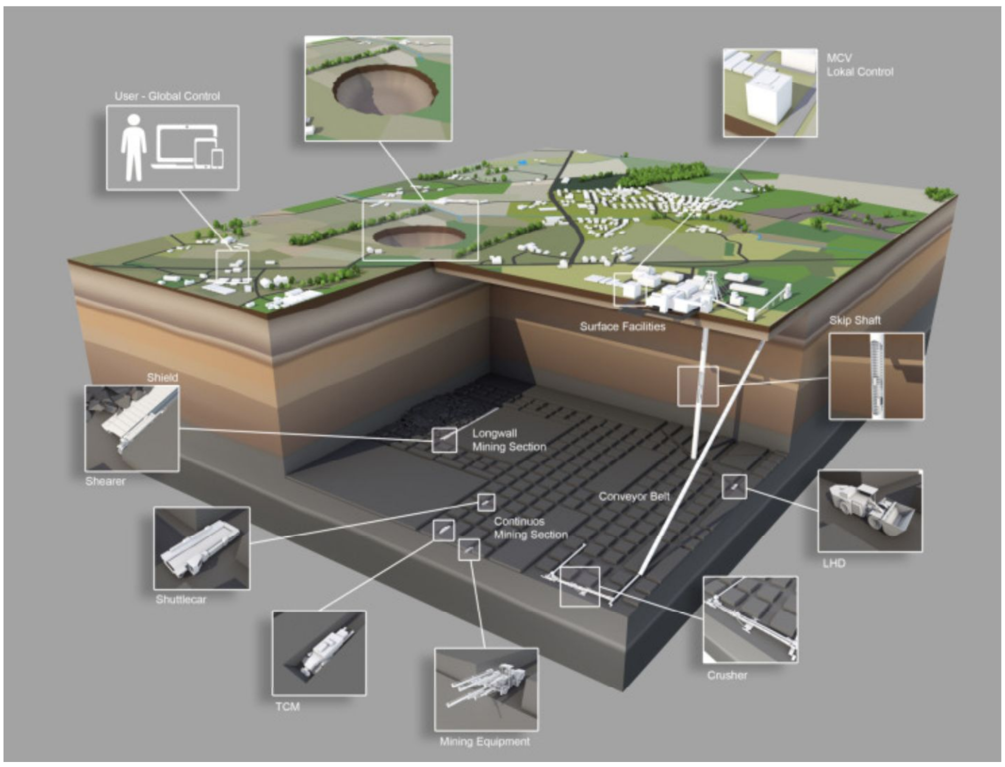


Figure 1.1: Model of a high-performance underground mining operation. Courtesy of DMT.

Mine shafts are exposed to various influences, such as motion of earth, vibration, abrasion through production and corrosion of steel components, which causes the shafts' building structure to degenerate, which can lead to instabilities or water inflow. This can also cause possible production interruptions and leads to very serious safety concerns for the mining operators. Mining companies distinguish between active production shafts and disused shafts. Interruptions in active shafts, operated in high-performance underground mining operations as shown in Figure 1.1, lead to production downtimes, often causing losses in the dimensions of several million Euro. The risk of collapse of a disused deep-mine shaft, as found amongst densely populated areas like the Ruhr region, is a great danger to the local infrastructure and can also lead to the pollution of ground water. The detection of damages, wear and change is therefore of great interest for the mining companies. Newer regulations from the mining authorities require more precise methods of shaft inspection. In particular, cracks in the dimension of a few millimetres have to be detected in deep mine shafts with a depth of $1000\,\mathrm{m}$ and a diameter of $8\,\mathrm{m}$. The intervals of these inspections range from monthly to daily inspection runs.

The current methods of inspection involve experts examining the shaft wall. These inspection runs are not only time and labour intensive, but also deliver poor quality results, due to fatigue and limited attentiveness and precision of the human inspectors. Most inspections are only documented by handwritten records. Additional photos taken by the inspectors support the documentation of their findings, but there exists no system that provides comparison of surveys over time, an automatic alert functionality, or a simply navigation through these surveys. Due to these disadvantages it is necessary to develop a system, supporting an objective and high quality inspection. As presented in [1] existing solutions use methods, such as kinematic laser scanning and profiling for 3D reconstruction. These solutions generate huge volumes of data, in particular point clouds, that are very difficult to handle by the means of data transfer and visualization. However, there are as of yet no approaches towards a replacement of the regular visual shaft inspection.

In 2016, the project *iDeepMon* (Intelligent Deep Mine Shaft Inspection and Monitoring), aiming at a fully automated process of shaft inspection, was awarded funding from the *European EIT RawMaterials Network*. As part of the project team, the Chair of Automation is responsible for the development of an automatic data processing system. This system receives amongst other data a sequence of overlapping images along a shaft, acquired from an eight-camera prototype from DMT[1]. This data is then processed to provide comparable and reliable datasets, linked to a virtual representation of the shaft, upon which the shaft inspector is able to perform data evaluation and investigation. Furthermore, the new system allows virtual marking of locations within the shaft that need further attention; monitoring of critical changes over time in history reports; reliable documentation of inspection results in case of damage or accidents; automated protocol generation; reduction of inspection-time; and the possibility to carry out the inspection in a safe environment.

In order to detect millimetre-sized cracks, an accuracy of 1 pixel per millimetre is demanded. The visualization of deep-mine shafts with a depth of $1000\,\mathrm{m}$ and a diameter of $8\,\mathrm{m}$ in the mentioned accuracy leads to hyper resolution images in the dimensions of $10^6 \times 10^5$ pixels. Assuming PNG as data format, 4 channels per pixel, 8 bits per channel, no compression and ignoring meta-data, the file size of the resulting image is approximately $370\,\mathrm{GB}$, just for the visualization of one inspection run. Additional data, such as the input stream of single images from the camera prototype, as well as the comparison of weekly inspection runs makes it obvious that the new system has to handle data volumes of several terabytes.

The implementation of such a data processing system builds upon the ideas of data ingestion, processing and storage, presented in [3] and also includes two research areas. The first area handles the generation of hyper resolution images out of an input stream of smaller, overlapping images, while the second area addresses the problem of visualizing large amounts of data in user-tolerated times at the client. The above mentioned system requirements: visualization of high resolution

[1]DMT is a global corporate group of 14 engineering and consulting firms, providing interdisciplinary services in the four markets Mining, Oil & Gas, Civil Engineering and Infrastructure & Plant Engineering[2]

imagery; navigation; marking and searching of locations; linking additional data to locations; showing past records; and providing different types of information in several layers are addressed by using web mapping applications. Their features are adapted to the specifications of automatic shaft inspection. [2]

Besides presenting methods of image registration, the main contributions of this thesis are:

1. showing the concept of using web mapping applications for hyper resolution image visualization;
2. implementation of a web mapping tool for mineshaft monitoring;
3. development of a framework and workflow for the automatic preparation of data and images to enable the use of hyper resolution images for mineshaft inspections.

[2]The nature of this project and its complexity is truly interdisciplinary. For this reason a number of different people with different skill sets and knowledge have been involved. This will be visible within certain chapters in this thesis.

Chapter 2

Image Registration

Receiving a sequence of images, acquired from the prototype shown in Figure 2.1, identifies the beginning of the workflow, presented in this thesis. Together the received images cover a horizontal field of view of $360°$ over the total depth of a mineshaft. In order to obtain a single hyper resolution image, used for the visual inspection of the shaft, these images are aligned through registration. This chapter gives a short introduction to the registration methods, used for this purpose.

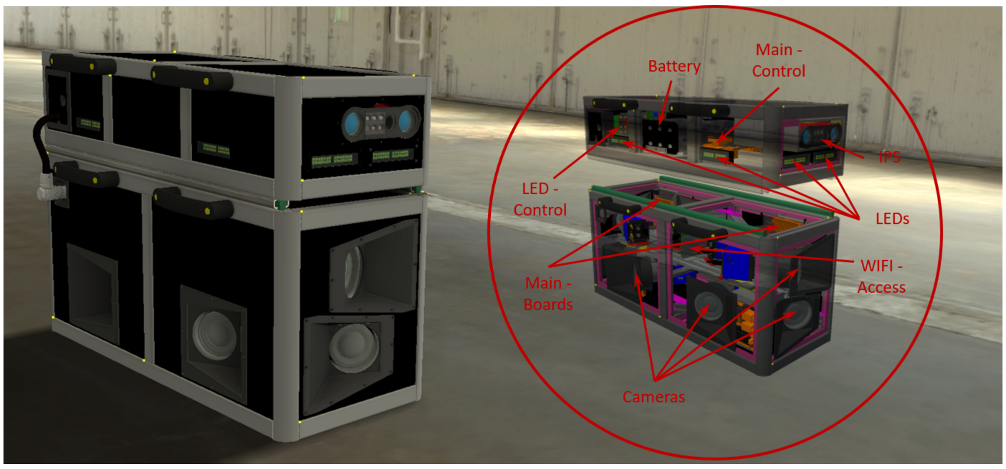


Figure 2.1: 8-camera prototype from DMT, which is supposed to acquire overlapping images of a shaft wall for later use in a visualization tool.

2.1 Homography

A point $\boldsymbol{p}$ in 3 dimensional space is mapped to an image coordinate $\tilde{\boldsymbol{x}}_0$, where 0 denotes the camera, through a combination of rotation and translation $\boldsymbol{E}_0$,

$$\tilde{\boldsymbol{x}}_0 = \begin{bmatrix} \boldsymbol{R}_0 & \boldsymbol{t}_0 \\ \boldsymbol{0}^T & 1 \end{bmatrix} \boldsymbol{p} = \boldsymbol{E}_0 \boldsymbol{p}, \tag{2.1}$$

using the formulation of the projection matrix $\boldsymbol{P}$,

$$\tilde{\boldsymbol{x}} \sim \left[\begin{array}{c|c} \boldsymbol{K} & \boldsymbol{0} \\ \hline \boldsymbol{0}^T & 1 \end{array}\right] \boldsymbol{p} = \boldsymbol{P}\boldsymbol{p} \tag{2.2}$$

where $\boldsymbol{K} = diag(f, f, 1)$ denotes the camera intrinsics with f being the focal length.

Combining the projection matrix $\boldsymbol{P}$ and the camera extrinsics $\boldsymbol{E}$ it is possible to formulate the projection of an image point $\tilde{\boldsymbol{x}}_0$ into an image point $\tilde{\boldsymbol{x}}_1$ in a different image,

$$\tilde{\boldsymbol{x}}_1 = \boldsymbol{P}_1\boldsymbol{E}_1\boldsymbol{p} = \boldsymbol{P}_1\boldsymbol{E}_1\boldsymbol{E}_0{}^{1}\boldsymbol{P}_0{}^{1}\tilde{\boldsymbol{x}}_0 = \boldsymbol{M}_{10}\tilde{\boldsymbol{x}}_0. \tag{2.3}$$

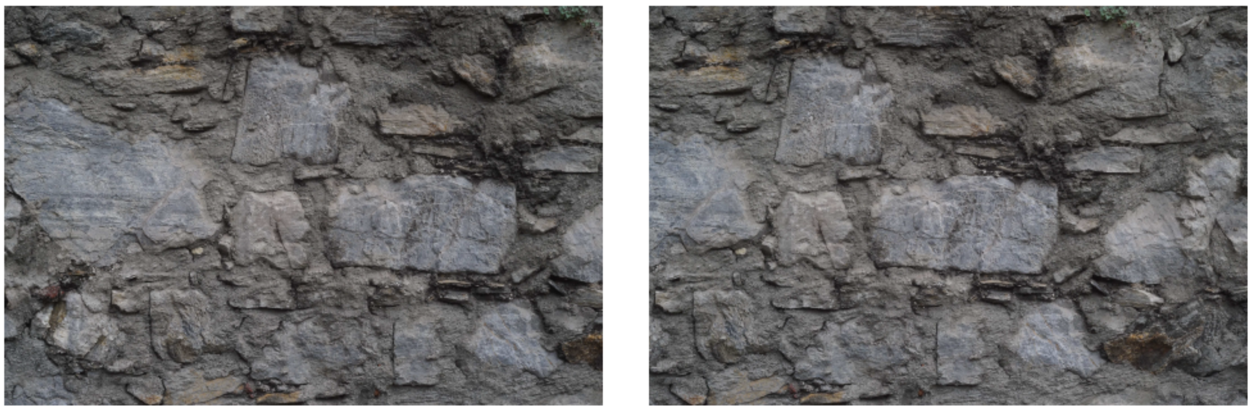
For a planar scene, the mapping is reduced to

$$\tilde{\boldsymbol{x}}_1 = \boldsymbol{H}_{10}\tilde{\boldsymbol{x}}_0 \tag{2.4}$$

where $\boldsymbol{H}_{10}$ is a 3x3 homography matrix and $\tilde{\boldsymbol{x}}_0$, $\tilde{\boldsymbol{x}}_1$ are 2D homogeneous coordinates[4]. In the case of images taken from a horizontal rotation which are then projected onto a closed surface, e.g., a cylinder, the registration problem is reduced to finding the translation between images. In order to determine the translation $\boldsymbol{t}$, different registration methods are available. In this thesis the so called *Phase Correlation* and *Non Rigid Registration* are presented.

2.2 Phase Only Correlation

Given two translated images, either through the correct choice of projection or by acquisition, it is intended to automatically find the offset between them.



(a) Reference image. f (b) Image g with an offset to the reference image.

Figure 2.2: Example for two horizontally translated images with overlapping area.

This can be achieved by analysing the phase correlation of the two overlapping images. For this the cross-power spectrum R of the two input images f and g is computed by

$$R = \frac{F \circ G^*}{|F \circ G^*|} \tag{2.5}$$

where $F = \mathcal{F}(f)$ is the Fourier transform of the image f and $G^* = \overline{\mathcal{F}(g)}$ is the complex conjugate of the Fourier transform of the image g. The correlation factor γ can be computed by

$$\gamma = \mathcal{F}^{-1}(R) \tag{2.6}$$

and the offset between the images is

$$(\Delta x, \Delta y) = \arg\max_{(x,y)}\{\gamma\}. \tag{2.7}$$

Figure 2.3 shows the correlation factor γ for the images shown in Figure 2.2.

Δx and Δy are then used in a *reference object* in MATLAB to perform the translation computation and the images are positioned relative to each other which is shown in Figure 2.4. A hyper resolution image of a shaft wall is produced by finding the position of the images, acquired by the prototype, relative to each other and computing a 2D reconstruction by correct alignment according to their position.

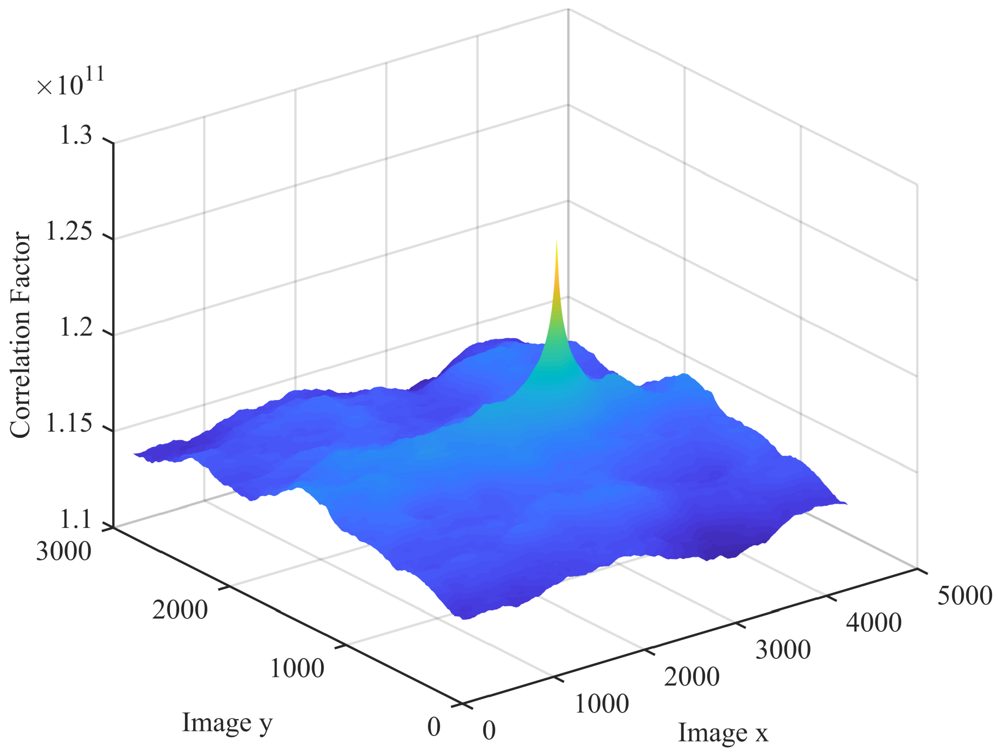


Figure 2.3: Correlation factor γ of the images from Figure 2.2.

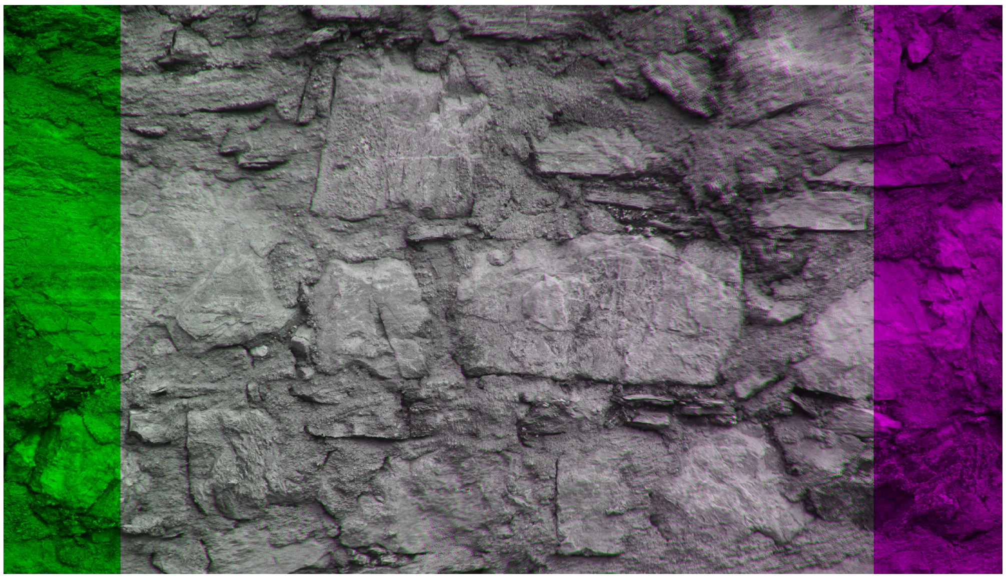
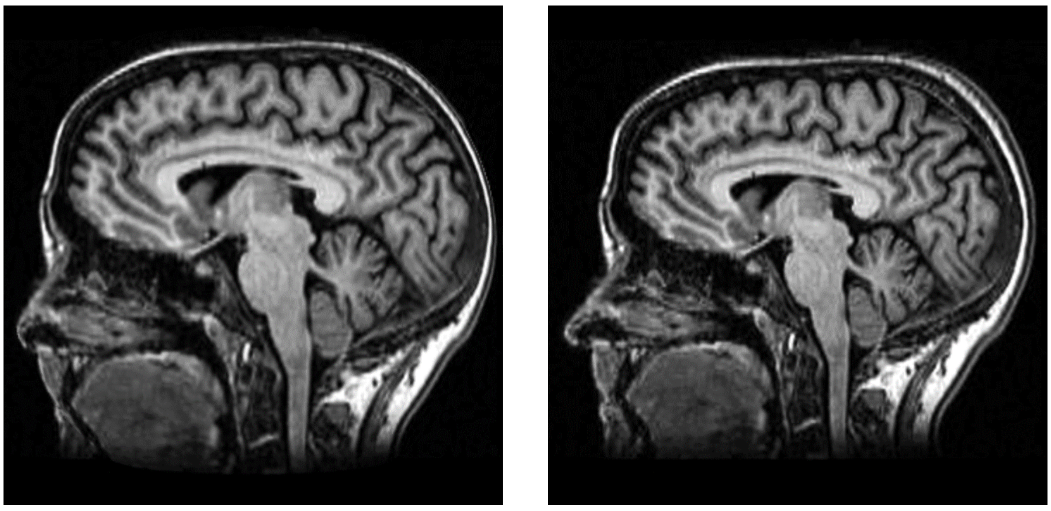


Figure 2.4: Overlay of the registered images according to the result of the phase correlation.

2.3 Non-rigid-registration

In the case of areas in the image chain, which can not be registered, methods such as the *non-rigid registration* are necessary to perform the registration. This is especially important for monitoring shafts over time. An image, capturing deformation of the shaft's wall, is registered by taking a previously acquired image of the same frame as reference image for the registration. This allows comparison and visualization of time series images.

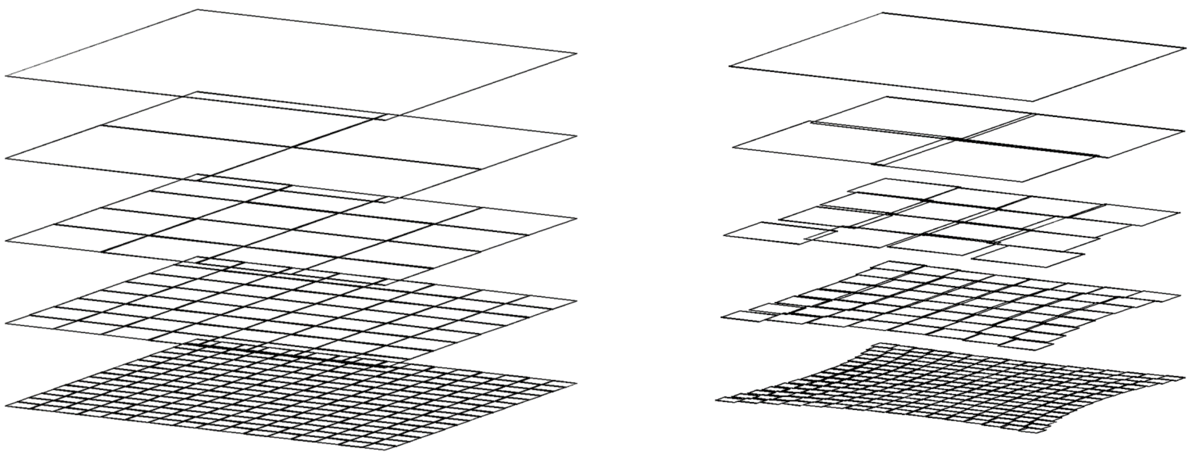


(a) Reference image.

(b) Image which is to be registered.

Figure 2.5: Example for change in x-rays over time.

Figure 2.5 shows x-rays of a brain, acquired at different times. The second image shows deformation in the brain area relative to the first image. The images are subdivided using a quad-tree structure. A coarse to fine registration is performed from the top to the bottom of the tree. The quad-tree structure of the decomposition of the reference image, as well as the registered and modified patches in the quad-tree structure of the second image are shown in Figure 2.6.



(a) Reference decomposition tree.

(b) Registered decomposition tree with modified position of each patch.

Figure 2.6: Pyramid structures of the deconstruction tree, each with 5 layers.

A modified normalized phase correlation is used for registration. The images are compressed and decimated from the bottom to the top of the tree, using Savitzky-Golay smoothing. This ensures that only global features are used for global registration and local features (details) at the bottom of the tree. The results of the smoothing and decimation for layer level l, 2 to 5 across the multi-resolution pyramid are shown in Figure 2.7. The decimation rate shown corresponds to 2^{l-1}.

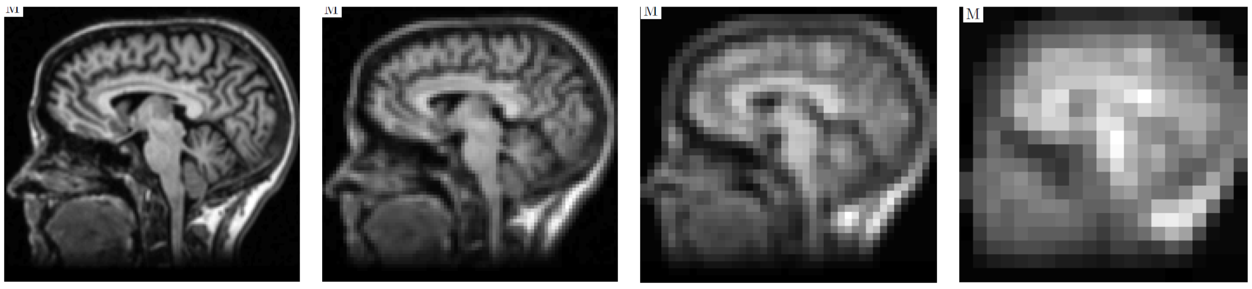


Figure 2.7: Results from the Savitzky-Golay smoothing and decimation with decimation rates, from left to right, 2, 4, 8 and 16.

A intermediate bivariate tensor polynomial approximation is used between each subdivision. This eliminates the tendency of individual patches to shift away from the grid.

There may be regions in an image which contain no or very little information. The principle is to weight the patch during the tensor approximation proportional to the information content. This is achieved by vectorizing the tensor approximation, and using a weighted bivariate polynomial regression. The reformed grid is then computed for all patches.

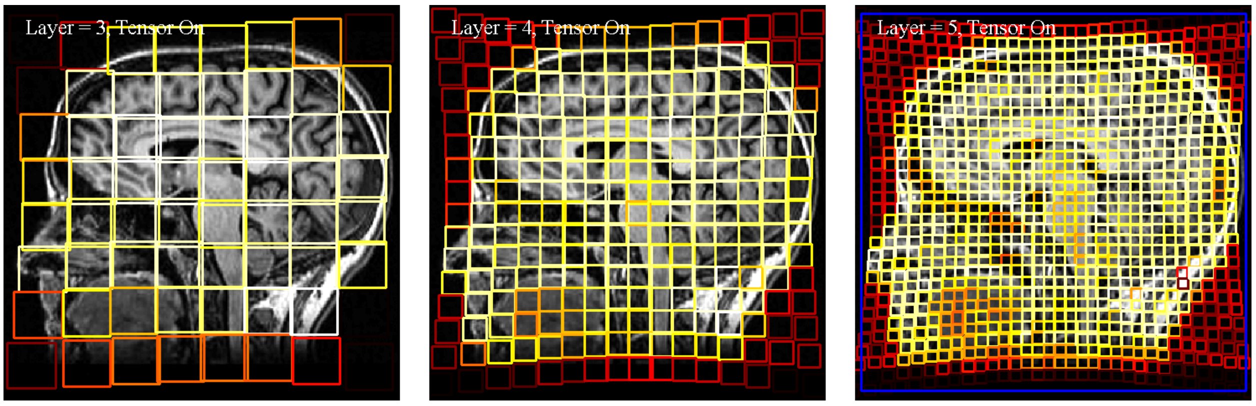


Figure 2.8: The above figures show the registration at sublayer 3 to 5 in the multi-resolution pyramid. The color of the patch is proportional to the entropy of data contained within the patch.

Figure 2.8 shows the weighting of the patches dependent on the entropy of the data in a patch. Other weighting measures such as standard deviation, total gradient, or moment invariants could also be used. Adding weighted tensor polynomial approximation improves the quality of the registration by reducing the effect of patches with low information content. The parameter which is suitable to indicate information content is dependent on the application. [5][6]

Chapter 3

Web Mapping Applications

Web mapping applications, such as Google Maps, Bing Maps or OpenStreetMaps are used by geographic information systems (GIS) to display maps over the internet. This chapter presents the method of using a *raster tile map* to display high resolution imagery. Raster tile maps are also referred to as *tiled web maps* or *slippy maps*. Furthermore, the suitability of this method for displaying hyper resolution images with a resolution of $10^6 \times 10^5$ pixels is investigated. In addition, concepts of web mapping applications, including projections and coordinate systems are introduced and relevant differences between different providers of web mapping libraries are specified.

3.1 Tiling

At first, the general structure of a *slippy map* is explained. Then, the specific structure considering the requirements from Chapter 1 is investigated with respect to the resulting file sizes. In addition, different ways of tile indexing are presented.

3.1.1 General structure

Slippy Map is a term, referring to the zooming and panning functionality of modern web maps (the map slips around when you drag the mouse)[7]. This map does not consist of a single high resolution image, but is composed of many small images, called *tiles*. Each tile is a square with a resolution of 256×256 pixels. These tiles are arranged in a raster and build up a pyramid like hierarchy according to their zoom level. Figure 3.1 shows the top 5 levels of this hierarchy. On zoom level 0, the whole world is displayed in a single tile. With each increasing zoom level, the number of tiles doubles in horizontal and vertical direction. This means on zoom level z, the whole map consists of 4^z tiles with a side length of $256 \cdot 2^z$ pixels. In order to avoid reloading the whole web page when the user pans around, the map is an *AJAX* component and the browser runs JavaScript. AJAX (Asynchronous JavaScript and XML) is a concept of asynchronous data transfer between browser and server. This allows to dynamically request new tiles from the server and keep still relevant tiles displayed. The tiles are rendered in advance and only actually needed tiles

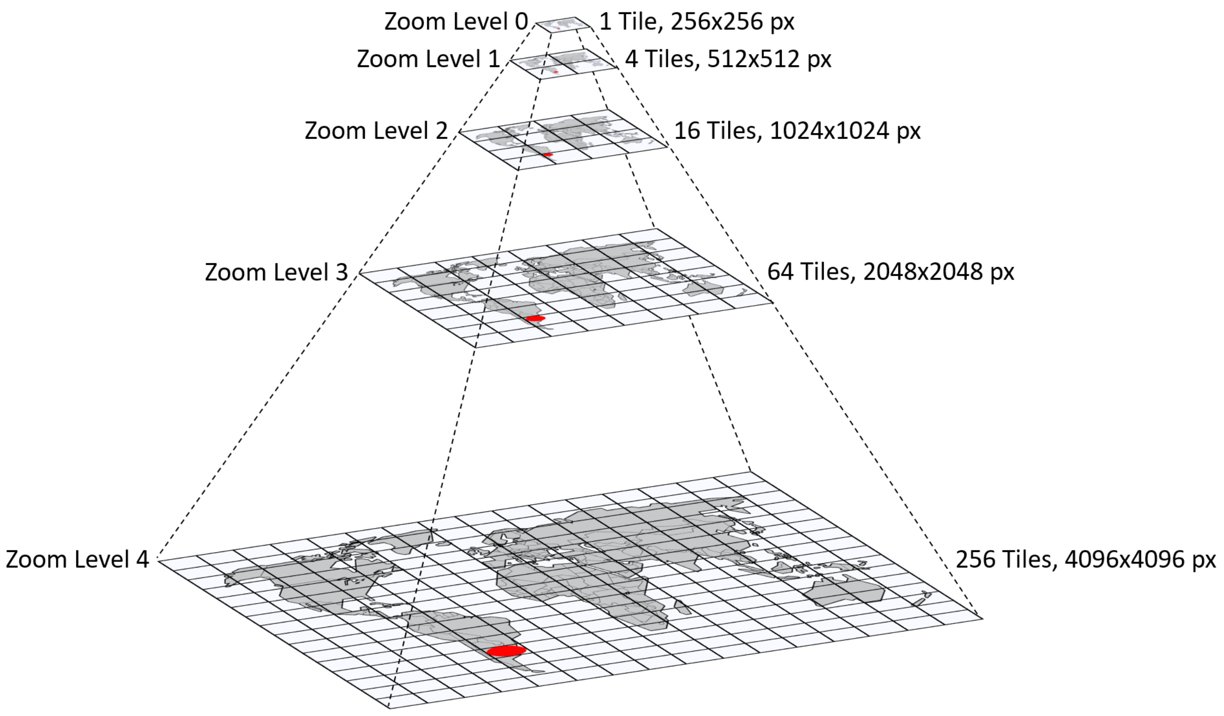


Figure 3.1: Representation of the pyramid tiling scheme from zoom level 0 to 4.

are loaded, which makes this method significantly faster than loading the whole map or render the current viewport. Rendering, storing and providing tiles are the main functionalities of a so called tile server. The maximum zoom level most tile servers support is 18. However, one can go beyond that level, by setting up a custom tile server.

3.1.2 File size consideration

The maximum possible zoom level z_{max} can be determined by,

$$t_s = 256, \tag{3.1}$$

$$s = \frac{max(p_x, p_y)}{t_s}, \tag{3.2}$$

$$z_{max} = \lceil \log_2 (s) \rceil, \tag{3.3}$$

where t_s is the side length of a tile in pixels, s is a scaling factor and p_x and p_y specify the number of pixels in horizontal and vertical direction of the hyper resolution image.

The number of tiles n in the respective direction at each zoom level can then be calculated by,

$$n = \left\lceil \frac{p/2^{(z_{max}-z)}}{t_s} \right\rceil, \tag{3.4}$$

Table 3.1: Data sizes of each zoom level for the visualization of a shaft with a depth of $1000\,\mathrm{m}$ and a diameter of $8\,\mathrm{m}$ and an accuracy of 1 pixel per millimetre. Assuming PNG as data format, 4 channels per pixel, 8 bits per channel, no compression, no meta-data.

Zoom level	Horizontal tiles	Vertical tiles	Data size	Cumulated data size
0	1	1	$256\,\mathrm{kB}$	$256\,\mathrm{kB}$
1	1	2	$512\,\mathrm{kB}$	$768\,\mathrm{kB}$
2	1	4	$1\,\mathrm{MB}$	$1.75\,\mathrm{MB}$
3	1	8	$2\,\mathrm{MB}$	$3.75\,\mathrm{MB}$
4	1	16	$4\,\mathrm{MB}$	$7.75\,\mathrm{MB}$
5	1	31	$7.75\,\mathrm{MB}$	$15.5\,\mathrm{MB}$
6	2	62	$31\,\mathrm{MB}$	$46.5\,\mathrm{MB}$
7	4	123	$123\,\mathrm{MB}$	$169.5\,\mathrm{MB}$
8	7	245	$428.75\,\mathrm{MB}$	$598.25\,\mathrm{MB}$
9	13	489	$1.55\,\mathrm{GB}$	$2.14\,\mathrm{GB}$
10	25	977	$5.96\,\mathrm{GB}$	$8.10\,\mathrm{GB}$
11	50	1954	$23.85\,\mathrm{GB}$	$31.95\,\mathrm{GB}$
12	99	3907	$94.43\,\mathrm{GB}$	$126.38\,\mathrm{GB}$

where z is the current zoom level and p the number of horizontal or vertical pixels.

Table 3.1 shows the number of tiles in horizontal and vertical direction, the data size and the cumulated data size of each zoom level. The visualization of a shaft specified in Chapter 1 requires a pyramid structure with 12 zoom levels and leads to an overall data size of approximately $125\,\mathrm{GB}$. About $8\,\mathrm{MB}$ need to be transferred in order to display a viewport filling the whole screen of a monitor with a resolution of 1920×1080 pixels. Data transfer rates above $64\,\mathrm{Mbit/s}$ guarantee loading times of under 1 second for this viewport and are sufficient to provide tiles without noticeable latency as the user navigates through the map. The currently supported 18 zoom levels are sufficient to visualize images with $6.7 \cdot 10^7$ horizontal and vertical pixels. The method of using a raster tile map is therefore suited for displaying hyper resolution images in the field of mineshaft inspections.

3.1.3 Indexing

According to [8], the three main systems of tile indexing are: *Google XYZ*, *Microsoft QuadTree* and *TMS* (Tile Map Service). Figure 3.2 shows the differences between tile indexing in these systems. Dealing with GIS-applications, the terminology for up-down is north-south, referring to the vertical axis and the terminology for left-right is west-east referring to the horizontal axis. The first row shows *Google's* tile coordinates in the format (x,y). The origin tile is in the north-west corner of the map. [9] The x-value (first coordinate) increases from west to east and the y-value (second coordinate) increases from north to south. The second row shows TMS indexing in the format (x,y), however compared to Google the origin tile starts in the south-west corner with increasing x-value towards east and increasing y-value towards north. Both systems, Google and

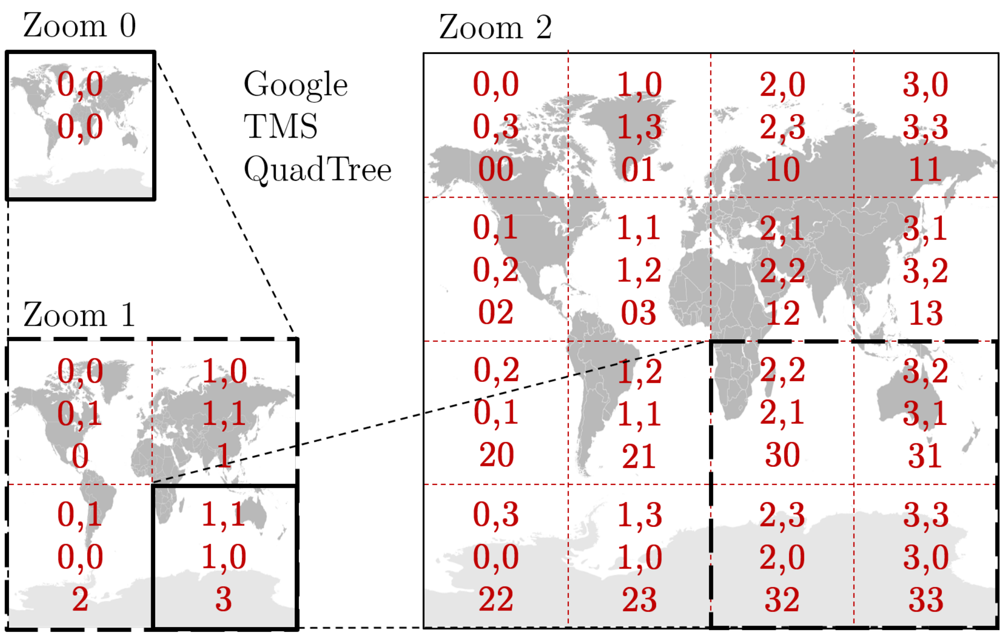


Figure 3.2: Different tile indexing schemes for zoom level 0 to 2. The first row shows Google's tile coordinates in the format (x,y). The second TMS indexing (x,y) and the third quad-keys. Attribution: Felipe Menegaz

TMS, use the zoom level of the respective layer as additional coordinate z, which is not shown in Figure 3.2. The third row shows *Microsoft's Bing Maps Tile System* indexing. This system uses so called quad-keys. On zoom level 1 the map is divided into 4 tiles, indexed clockwise with (0), (1), (3) and (2), starting in the upper-left corner. With each increasing zoom level the quad-keys of the children are indexed in the same way, starting with the quad-key of the parent tile. As shown in Figure 3.2, the tile with quad-key (3) is the parent of the tiles with quad-keys (30) through (33).

3.2 Map Projections

A map projection is a mathematical transformation of a 3-dimensional spherical model, i.e. the planet earth, into a flat 2-dimensional surface. All map projections create distortions and are divided into categories according to which properties they preserve: area; shape; direction; distance or scale. The standard for web mapping applications is the *Web Mercator Projection*, which is a variant of the Mercator projection. The Mercator projection is a cylindrical map projection. Figure 3.3 shows the construction of a cylindrical projection. The earth is approximated with a sphere and a cylinder is placed tangential to it associated with the equatorial line. A line is extended from the centre point of the sphere and a point on the surface of the sphere is mapped to a point on the cylinder as the line intersects the cylinder. The cylinder is then rolled out to obtain the map, shown in Figure 3.4. Distortion rates grow as the distance to the equatorial line increases and the projection goes to infinity at the poles. However, the scale of the east-west stretching is equal to the scale of the north-south stretching at every point, making the Mercator projection *conformal*. Angles are preserved locally, which means that the shape of relatively small objects is preserved as well.

In addition north and south equal straight up and down, and east and west equal straight right and left at any point. In order to avoid partially filled tiles, a square aspect ratio is used and because the projection goes to infinity at the poles, polar regions are excluded, by truncating latitudes above 85.05 degrees. [10][9][11][12]

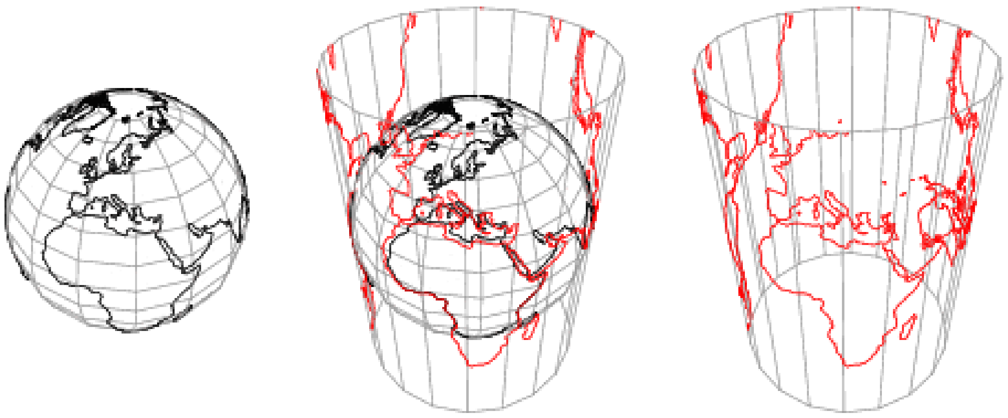


Figure 3.3: Construction of a cylindrical map projection.[13]

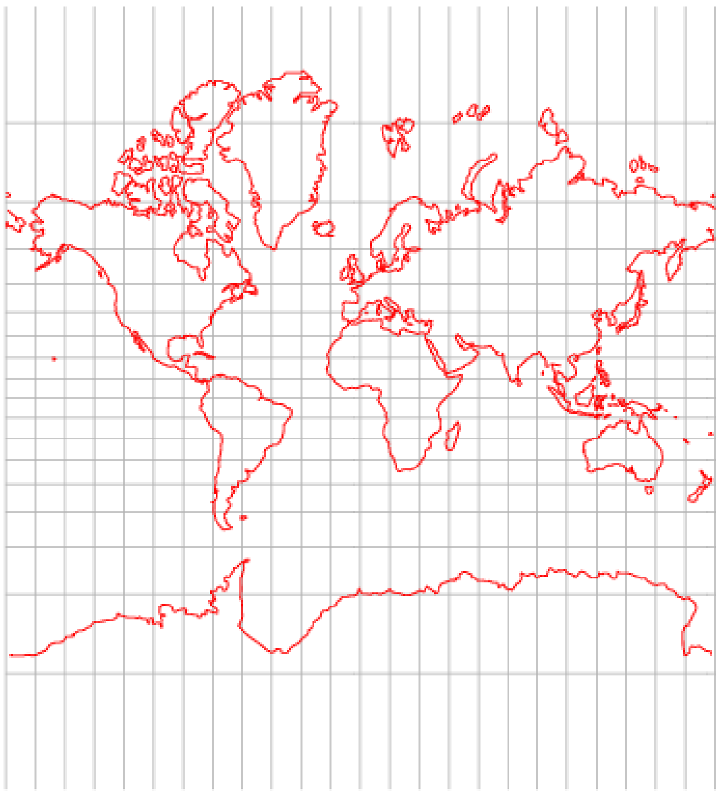


Figure 3.4: Result of the Mercator projection.[14]

3.3 Coordinate Systems

In general, coordinate systems are used to address specific locations on the earth's surface. The most common is the *geographic coordinate system*, where locations are specified by latitude, longitude and elevation. Lines of latitude and longitude form a grid over the earth's surface, as shown in Figure 3.5. Lines of latitude are parallel to the equator and circle the globe. The latitude of a point is specified by the angle between the axis, passing through this point on the surface of the earth and the equatorial plane. The equator has a latitude of 0 degrees and the poles 90 degrees north, respectively south. These lines are the same distance apart. Lines of longitude, also called meridians are vertical to the equator and circle the globe. They intersect at the poles, making them not equidistant. The prime meridian at 0 degree longitude runs through the town Greenwich. The longitude of a point is specified by the angle between this prime meridian and the meridian running through this point. The earth is divided into 180 degrees of longitude west, and 180 degrees of longitude east. Elevation is specified as the normal distance between a point on the surface and the geoide of the mathematical model of the Earth's sea level.

In order to address locations on a map, web mapping applications translate spherical coordinates, longitude and latitude, to Cartesian coordinates x and y. In the terminology of web mapping applications, these coordinates are referred to as *world coordinates*. World coordinates are independent of the current zoom level and refer to the tile at zoom level 0 with x and y ranges between 0 and 256. World coordinates are specified by floating point values, measured from the upper-left corner to the specific location[9]. The x coordinate increases towards east and the y coordinate towards south. Pixel coordinates are used to identify a specific pixel on the map at a specific zoom level. Given latitude ϕ, longitude λ, the current zoom level z and the side length of a tile t_s, the pixel coordinates p_x and p_y can be calculated by,

$$r = 2^z t_s, \tag{3.5}$$

$$s_\phi = \sin\left(\frac{\phi\pi}{180}\right), \tag{3.6}$$

$$p_x = r\frac{\lambda + 180}{360}, \tag{3.7}$$

$$p_y = r\left(0.5 - \frac{1}{4\pi}\log\left(\frac{1+s_\phi}{1-s_\phi}\right)\right), \tag{3.8}$$

where r is a scaling factor and s_ϕ is the sine of ϕ (in radians), as described in [10]. Using Equation 3.4 with p as pixel coordinates and taking the integer part of the solution instead of rounding up, the tile index of the tile, where the pixel is located can be determined.

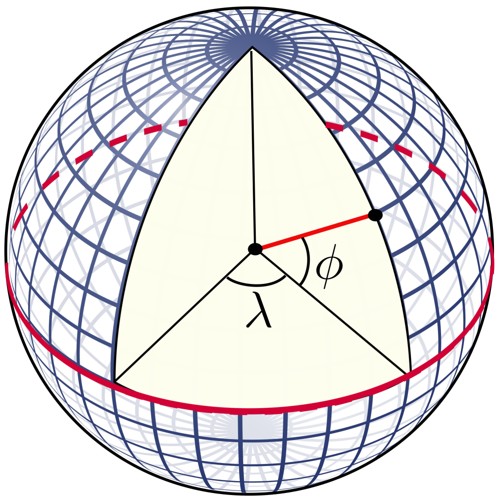


Figure 3.5: Representation of lines of latitude ϕ and longitude λ.

Chapter 4

Implementation

The implementation of a web mapping application requires the use of a web mapping library. Several APIs (Application Programming Interface) exist from providers like Google, Microsoft, OpenLayers, Leaflet, Mapbox etc., however not every library is suited for the implementation of a non-GIS application. As web mapping applications are intended for the use of GIS maps and geodata, they expect the base map to be in Web Mercator projection, as well as the use of a geographic coordinate system. An image of a shaft wall, or any other non-GIS imagery, contains no geographic informations with respect to longitude and latitude. Although hypothetical geographic coordinates can be assigned to the corner points of the image as proposed by [15], this leads to inaccurate measurements or presentations of distances due to the effects of distortion. Leaflet is the only library that natively supports methods for defining a coordinate reference system (CRS). Defining a CRS regulates the assignment of coordinates, which makes the use of projections unnecessary. Besides that, an external library called *Proj4Leaflet* exists, that supports the use of projections and CRSs not built into Leaflet. Furthermore Leaflet is the most light weight library with a data size of approximately $130\,\mathrm{kB}$, supports all major browsers on desktop and mobile platforms, has a well documented API and is free to use. Therefore, Leaflet's JavaScript library is used for the implementation of a visualization tool for hyper resolution images in the fields of mineshaft inspection. [1]

4.1 Layers

Web mapping applications display different information on different layers. Leaflet accepts three basic types of input data for these layers: image data; raster tiles; and vector data. Every type can be used as base map, or as an overlay for certain sectors, or as a full layer on top of the base map. Large images can not be handled efficiently due to limited data transfer rates. Therefore large images are divided into tiles as described in Chapter 3. SVG (Scalable Vector Graphics) images can also be used in Leaflet. The data size of such an image does not depend on its dimensions, but on the information it contains. As long as the data size is small enough, a SVG image can

[1]This chapters's descriptions of Leaflet's methods, classes and functionalities are taken from the tutorials and documentation, provided by http://leafletjs.com

be handled efficiently by assigning its corner points to coordinates on the map. Opacity can be adjusted for all kinds of image and raster data and is used to display different layers at once, e.g. a heat, pressure or error map on top of the base map. The use of vector data will be discussed in Section 4.2. Figure 4.1 shows different layers with different data types on top of each other.

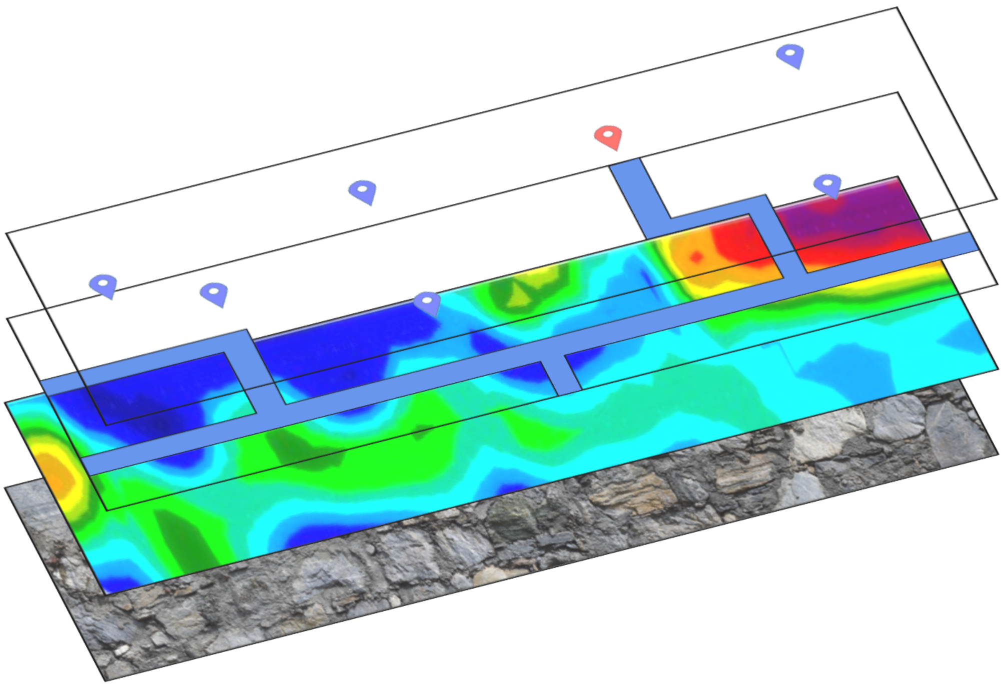


Figure 4.1: Representation of different information in different layers. From bottom to top: a raster tile layer as base map; an image layer using a PNG image; an image layer using a scalable vector graphic (SVG); and a vector layer represented by markers on the map.

In order to receive continuous location referencing across all layers, a CRS is defined. The Chair of Automation uses different software for image processing and data analysis. Especially in the data exploratory phase, *MATLAB* and *Python* are mainly used. Both represent images as a 3 dimensional matrix. This matrix has 3 fields for every pixel, holding values from 0 to 255 for the RGB color space. The CRS is defined to use the row and column indices of this matrix as y and x pixel coordinates of the image. This ensures correct referencing between different software. Furthermore a coordinate system in pixel coordinates provides simple integer coordinates that can be used to identify certain tiles with the formulas from Chapter 3. In addition it is easily possible to convert between pixels and meters, if the camera intrinsics and extrinsics are known. Leaflet uses the CRS internally for all distance and location calculations. Leaflet's built in *CRS.Simple* is used for flat maps and transforms longitude and latitude into x and y world coordinates directly. The new CRS is generated through extending this *CRS.Simple* by defining an affine coordinate transformation. With

$$\boldsymbol{S} \triangleq \begin{bmatrix} a & 0 \\ 0 & c \end{bmatrix}, \tag{4.1}$$

$$\boldsymbol{t} \triangleq \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}, \tag{4.2}$$

$$\boldsymbol{p} \triangleq \begin{bmatrix} x \\ y \end{bmatrix}, \tag{4.3}$$

$$\tilde{\boldsymbol{p}} \triangleq \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix}, \tag{4.4}$$

the matrix equation is formulated as

$$\tilde{\boldsymbol{p}} = \boldsymbol{S}\boldsymbol{p} + \boldsymbol{t}, \tag{4.5}$$

with $\hat{x}$ and $\hat{y}$, being the transformed coordinates of a point $\boldsymbol{p}$ with coordinates x and y, the scaling matrix $\boldsymbol{S}$ with

$$a = c = \frac{1}{s}, \tag{4.6}$$

using s from Equation 3.2 and the translation vector $\boldsymbol{t}$ with horizontal translation Δx and vertical translation Δy.

The corner points of the image can then be used as pixel coordinates to define the map bounds, which ensures correct loading of tiles. The tile layer, serving as base map is loaded by committing the directory structure of the raster tile pyramid as URL (Uniform Resource Locator). Leaflet uses the *OpenStreetMap* standard scheme for this URL, referring to tiles that follow Google's indexing system. Figure 4.2 shows a representation of the directory structure, following this scheme. Every zoom level has its own sub-directory in which again every x-index makes up an own sub-directory. The y-index is used in the file name of the tiles. The generation of this tile-directory will be described in Chapter 5. Each layer can be added to the so called *layer control*, which enables the user to switch between base maps and show or hide additional layers. Figure 4.3 shows the user interface of the layer control among other features.

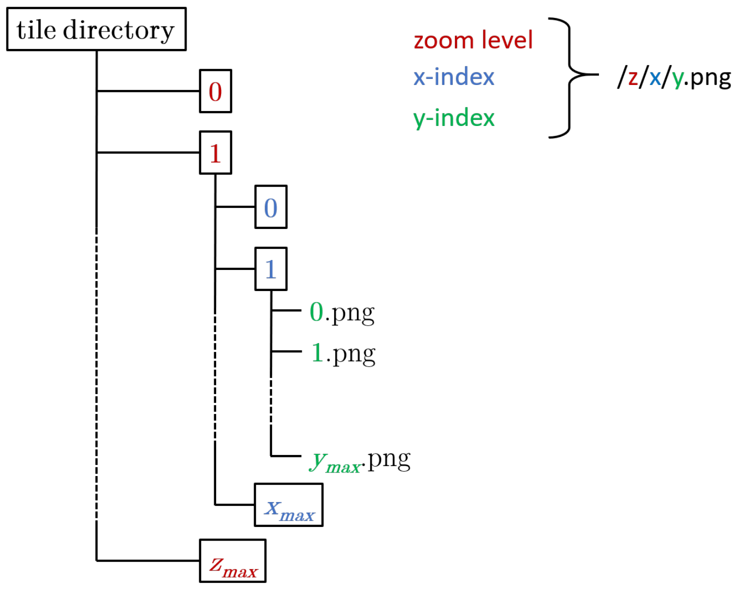


Figure 4.2: Representation of the directory structure resulting from using the *OpenStreetMap* standard URL (tile name) scheme: '/zoom/x/y.png'.

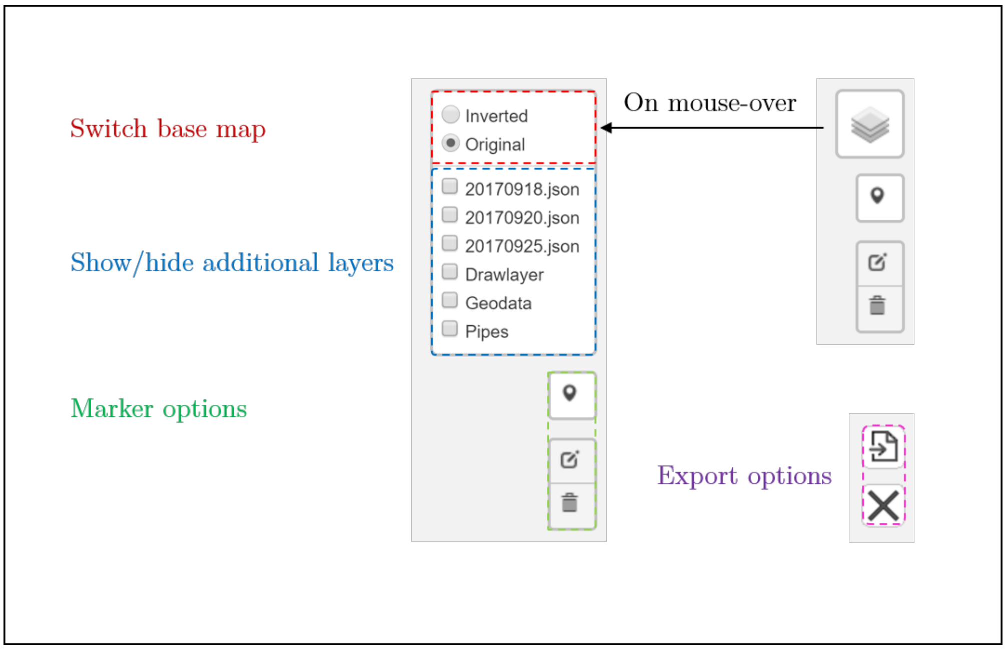


Figure 4.3: Representation of the user interface, that allows the user to switch between different base maps; show or hide additional layers; place, edit or delete markers; and export tagged locations as a JSON file. Icons made by Leaflet, *www.flaticon.com/authors/anton-saputro* and *www.flaticon.com/authors/google*.

```
{ "type": "FeatureCollection",
  "features": [{"type": "Feature",
            "geometry": {"type": "Point",
                   "coordinates": [9393,802]},
            "properties": {"name": "Location1",
                     "file": "report1.pdf"}},
           {"type": "Feature",
            "geometry": {"type": "Point",
                   "coordinates": [178,2057]},
            "properties": {"name": "Location2",
                     "file": "report2.pdf"}}]}
```

Source Code 4.1: Example for the content of a JSON file, containing two locations with associated coordinates and linked pdf-files.

4.2 Location Tagging

It is possible to add information to locations in the visualization. In web mapping applications, tagged locations are represented by markers. *JSON* (JavaScript Object Notation) objects are used by Leaflet to store the information content. These objects are also called *features* and can be stored in JSON files. JSON is used as data exchange format between different software[2]. Besides vector data, these JSON files provide meta data associated with the tag representation in the hyper resolution image. Thus it is possible to link additional files and documents to marked locations, e.g. downloadable error reports or inspection results. The content of such an JSON file can be defined as follows:

Besides using markers to represent tagged locations, web mapping applications use different geometries to display vector data: points; lines; circles; rectangles; and polygons. A marker is Leaflet's native representation of a point. Different features can be displayed at different zoom levels, which avoids overcrowded maps with too much information. Google Maps uses this functionality to display street names of small streets only below a certain zoom level.

In order to navigate through all tagged locations, a navigation list is implemented, as shown in Figure 4.4. By clicking on a location in this list, the map is centred on its coordinates and zoomed closer. The same can be achieved by clicking on a marker on the map. The navigation list also contains click-able links forwarding to the associated PDF-files.

Another feature is implemented using the *Leaflet.draw* plugin, that allows the user to place and edit markers on the map. This enables the shaft inspector to tag locations, that need further attention. To start placing markers, the user has to add a so called *drawlayer* to the map in the layer control. If markers are already on the map when activating the drawlayer, they will be added to this layer automatically. The user can then edit, delete or remove these markers, or add new ones. In order to save new locations, the feature collection can be exported as a JSON file. Figure 4.4 shows the graphical user interface (GUI) of the application with a short description of its functionalities. The source code of the HTML file of the application is shown in the list of source codes in the appendix of this thesis.

[2]MATLAB supports JSON encoding and decoding since version R2017a.

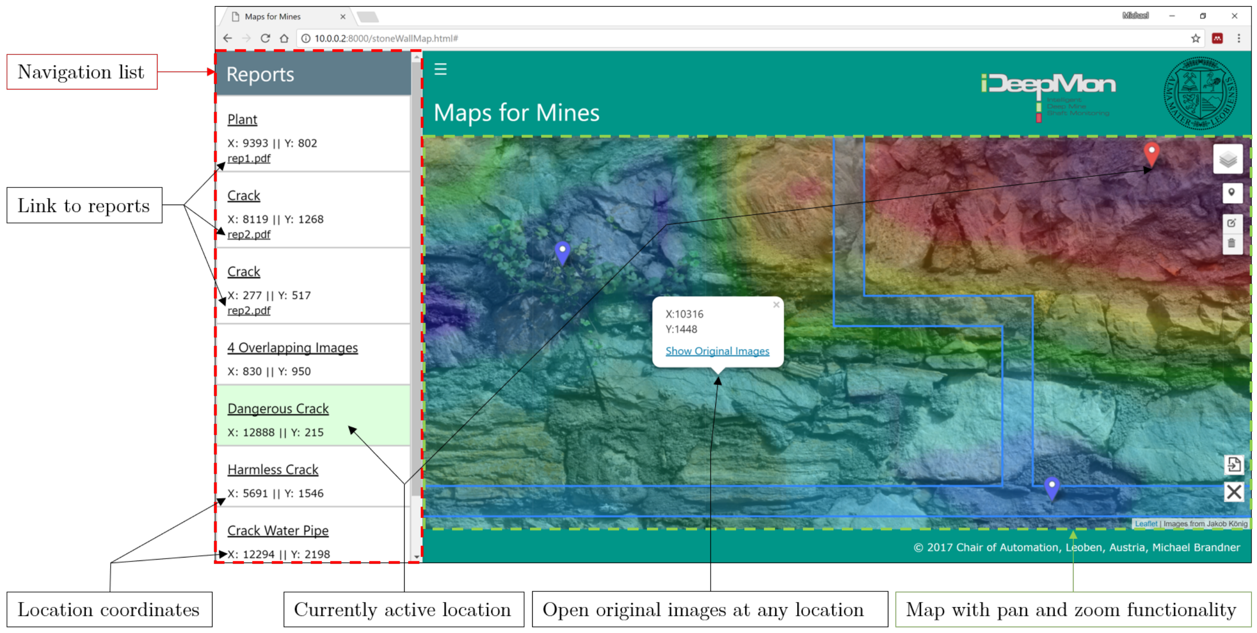


Figure 4.4: Graphical user interface with description of features. Note that all layers from Figure 4.1 are displayed.

4.3 Image Identification

The hyper resolution image production process includes the composition of the original images according to the results of the registration, and a blending step. This blending produces regions in the image where important features might be unrecognisable. To account this problem, a method is implemented to inspect all original images, which contribute to the blended region. This feature uses MATLAB's *reference object*, that was generated during registration, for referencing pixel coordinates of the map with the original images. As the usability of the application depends on the amount of data being transferred, considerations concerning data structure demand explicit reference. The attempt to use a matrix with the dimensions of the composite image and store the IDs of the belonging original image in the corresponding field of the pixel fails due to the resulting huge amount of data. A solution is found by storing only information about the image area and its location in the hyper resolution image in an array, which is shown in Table 4.1. The row index of the array is used as image ID and a second array contains the belonging file names of the original image. Table 4.2 compares the two approaches, considering filesize and dimensions. The resulting file size of the reference array is only proportional to the number of original images.

By clicking the right button of the mouse at any location on the map, a popup opens, which displays the pixel coordinates of this location. In addition, the popup contains a link that opens a slideshow tracing back to the original captured images. Figure 4.5 shows this slideshow. This functionality can be expanded to show images of different inspection runs within the same frame. This enables the inspector to compare changes in the shaft wall over time, e.g. length of cracks, deformation or wear. In order to monitor these changes, the shaft inspector can then use the tagging functionality to mark locations that need further attention.

Table 4.1: Extract of the reference array of approach 2, storing the corner points of the original images in pixel coordinates, referring to an area in the composite image. The row index is used as ID, linked to the actual file name of the image.

x_{min}	x_{max}	y_{min}	y_{max}
1	722	1	451
622	1476	1	451
1376	2002	1	451
⋮	⋮	⋮	⋮
14187	14260	904	2525
14885	14260	904	2525

Table 4.2: Comparison of different data structures for image backtracking. Approach 1 uses an array in the dimensions of the composite image to store the IDs of the belonging original images for each pixel. Approach 2 stores only the corner points of each original image. The test was performed for a high resolution image with 14260×2525 pixels, composed of 75 images.

	Number of rows	Number of columns	Resulting file size
Approach 1	2525	14260	300 MB
Approach 2	75	4	3 kB

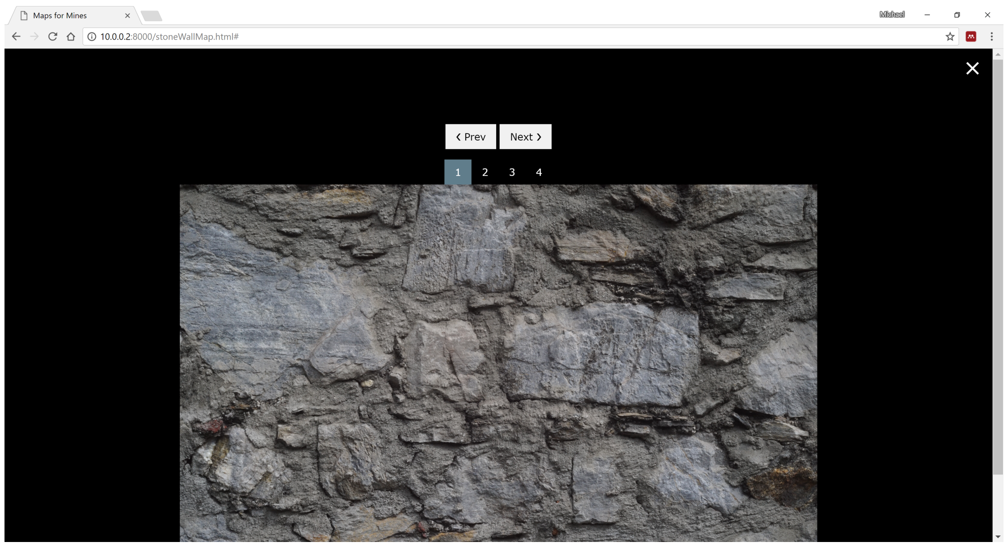


Figure 4.5: Slideshow presenting overlapping images from the construction of the hyper resolution image.

Chapter 5

Dataflow Design

This chapter completes the framework for hyper resolution image visualization in mineshaft inspection by presenting the last missing component: the implementation of the tiling process. The complete workflow from image acquisition to data visualization is considered in detail and a solution towards a generic tiling and registration process is proposed. Finally the workflow is tested.

5.1 Map Tiling

In order to use a hyper resolution image as a map in web mapping applications, the image has to be divided and rendered into base tiles and overview tiles. Base tiles are tiles at the maximum zoom level and overview tiles are tiles below that zoom level. The tiles are produced and stored by a tile server. For test purposes a local tile server is set up, using *GDAL* (Geospatial Data Abstraction Library) as tool for image tiling. Single tiles on this server can be accessed and investigated with respect to filesize, indexing and correct referencing, which is very valuable for debugging during the development of the mapping application. Furthermore the maximum number of zoom levels is only limited by disk size. GDAL is also used internally by many tile server providers to perform image tiling. The generation of base tiles is a simple cutting process. Creating overview tiles require image re-sampling (scaling) methods. GDAL uses the "average" scaling algorithm per default for rendering overview tiles, but also supports other algorithms[1]. The Python-script *gdal2tiles.py* is normally used for tiling, but only supports the TMS indexing scheme. As mentioned above, Leaflet follows the *OpenStreetMap* URL scheme for addressing tiles, following Google's indexing scheme. Therefore, a modified version of this script is used, provided by [16]. Figure 5.1 shows a sub-folder of the directory structure of a tiled test image on the example of *Windows Explorer*. Following the *OpenStreetMap* URL scheme for addressing tiles leads to duplicate file names in different directories, as there are many tiles with the same y-index. Although Leaflet follows this scheme, it is possible to define an own addressing scheme to avoid duplicate file names. Meta-data can be added by extending the directory structure or the file name itself. However, Leaflet transfers the requested tile coordinates only as plain integers. It is therefore not possible to use leading zeros in order to construct file names with the same number of characters. An example for a thoughtful

[1]http://www.gdal.org/gdal_translate.html

way of tile naming is Microsoft's Bing Maps' use of quad-key indexing, which allows unique file names. In addition tile retrieval performance is enhanced, which is described in [10] as follows: *"Quadkeys provide a one-dimensional index key that usually preserves the proximity of tiles in XY space. In other words, two tiles that have nearby XY coordinates usually have quadkeys that are relatively close together. This is important for optimizing database performance, because neighboring tiles are usually requested in groups, and it's desirable to keep those tiles on the same disk blocks, in order to minimize the number of disk reads."*

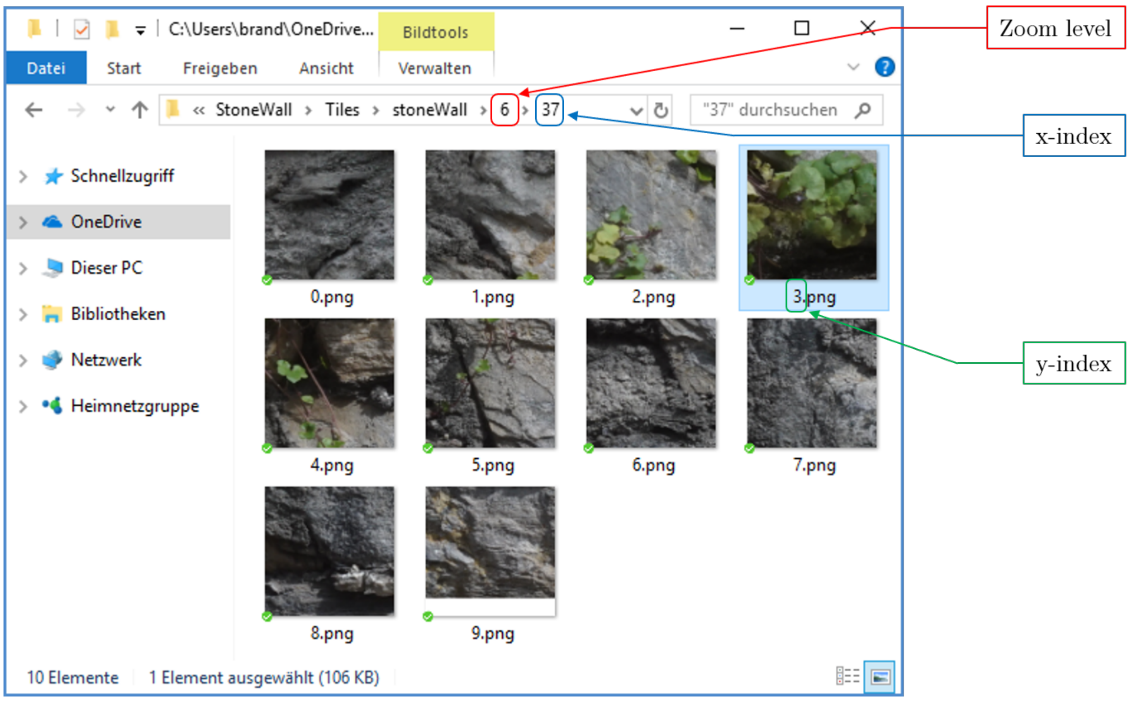


Figure 5.1: Representation of the directory structure of the image tiles in windows explorer. Note that the image *9.png* has a white area at the bottom of the image. This results from tiling an image whose dimensions does not match a power of 2.

5.2 Workflow

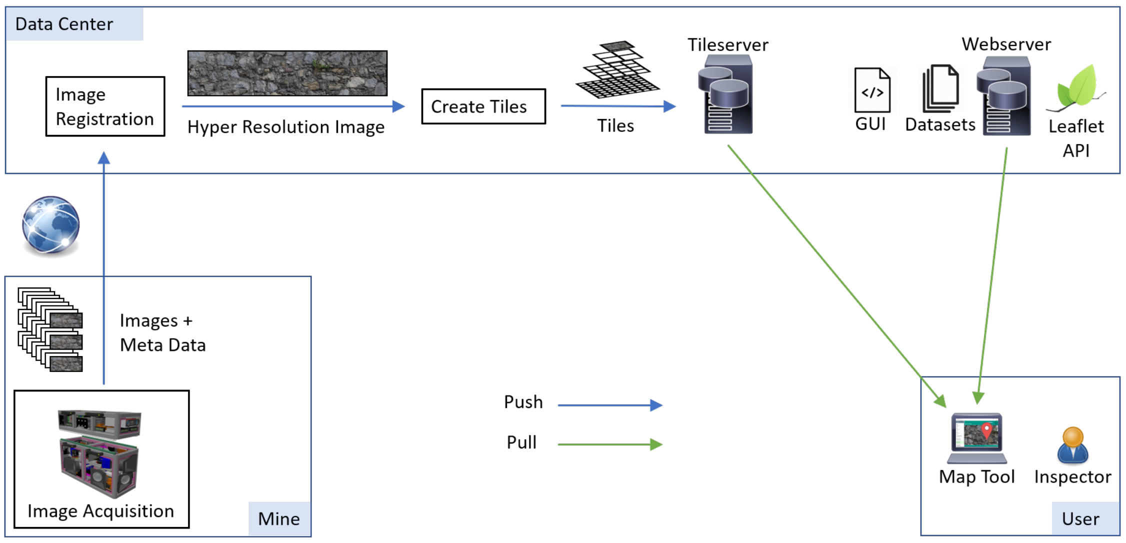


Figure 5.2: Workflow from image acquisition to data visualization.

The final workflow is shown in figure 5.2. The original images of the shaft wall are acquired by an 8-camera prototype of DMT. Together with meta-data these images are sent to the data centre, where they are registered and stitched to obtain a continuous visualization of the shaft wall. The image model is then tiled and the tiles are stored on a tile server. A generic solution for both, image registration and tile creation, is found by using *Apache NiFi* for automated data processing, as described in [3]. A NiFi *GetFile* processor monitors the incoming directory of the data centre's storage and once a complete sequence of images arrives, an *ExecuteStreamCommand* processor is triggered, that uses a Python script for image registration and stitching. The composed hyper resolution image is then sent to another processor, that uses the modified version of *gdal2tiles.py* to create the tiles. A *PutFile* processor places the tiles in the correct directory structure on the tile server. The creation and storage of tiles is executed by the data centre, but can also be outsourced to an external tile server provider. A web server hosts a web page with the implementation of the web mapping application and provides additional datasets and necessary meta-data. This web server permits viewing the data on any device, stationary or mobile. Finally the shaft inspector is able to access the web page and the data can be viewed.

5.3 System Testing

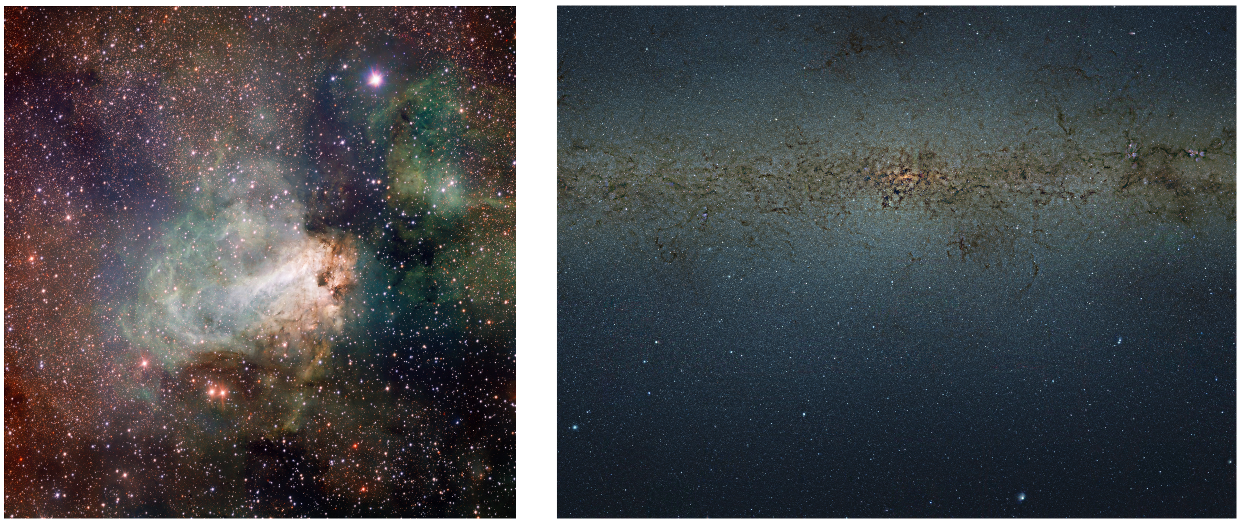
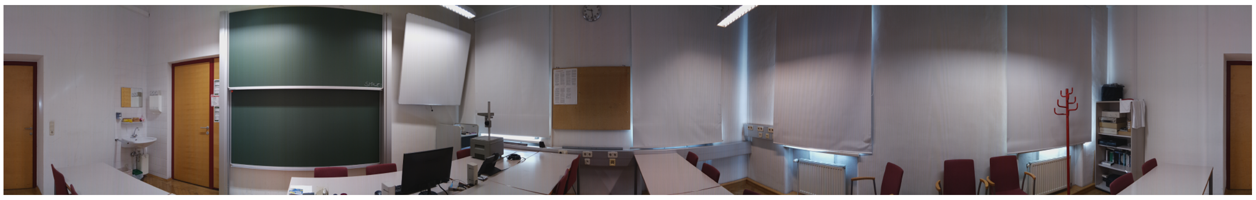

(a) VST image of the star-forming region Messier 17 (b) VISTA gigapixel mosaic of the central parts of the Milky Way

Figure 5.3: Two examples for hyper resolution images: (a) $664\,\mathrm{MB}$ TIFF-image in the dimensions 16017×16017 px and (b) $24.6\,\mathrm{GB}$ PSB-image in the dimensions 108199×81503 px. Credits: (a) ESO/INAF-VST/OmegaCAM. Acknowledgement: OmegaCen/Astro-WISE/Kapteyn Institute; (b) ESO/VVV Survey/D. Minniti Acknowledgement: Ignacio Toledo, Martin Kornmesser

While the camera prototype from DMT is still in development, test images are needed for the implementation of the mapping tool and for testing its performance. Tests are performed on hyper resolution images provided by the *European Southern Observatory (ESO)*, as well as on self-acquired images. Figure 5.3 shows the two images that are used for the performance tests. Image (a) contains approximately $3 \cdot 10^8$ pixels and image (b) $9 \cdot 10^9$ pixels. JPEG can not be used as data format for image (b), because *JPEG* images have a limited side length of approximately 65000 pixels[17]. Special data formats, such as *BigTIFF* or Photoshop's *PSB* are used to handle large images. Unfortunately the ESO is only able to provide image (b) in the PSB file format, which is not supported by GDAL. Therefore a smaller version of this image ($3.92\,\mathrm{GB}$) with a resolution of 40000×30131 pixels in the TIFF file format is used for testing. Both images are inconvenient to use by the means of loading times and resource consumption on a laptop with an *Intel Core i7* processor, $32\,\mathrm{GB}$ of RAM, and a SSD storage, even when stored and accessed locally. However tiling the images enables a zooming and panning functionality without latency, even when serving the tiles over the internet.

In order to test the whole workflow from image acquisition to data visualization, sequences of test images are acquired. Figure 5.4 shows two different approaches to image acquisition. The $360°$ panorama image (a) is composed of 3200 images from a slit-camera, as described in [18]. For obtaining the image sequence of image (b), a camera is positioned normal to a wall and moves along the wall in the same distance as the images are acquired. Image (b) is composed of 75 images. Image registration methods, presented in Chapter 2, are used to register the sequences of images and stitch them together to obtain hyper resolution images. The image reference object in MATLAB, created during registration, enables backtracking to the original images in the mapping tool. GDAL handles tiling as expected and it turns out that the actual file size of a single tile is only about one half of the originally assumed $256\,\mathrm{kB}$, due to compression. This means that the complete structure of the tile directory for the visualization of a deep mine shaft will only need around $60\,\mathrm{GB}$ of storage, if a similar compression rate applies. Finally, the data visualization in the mapping tool as well as the tool's functionality works as described.


(a) $360°$ panorama image from a slit camera approach

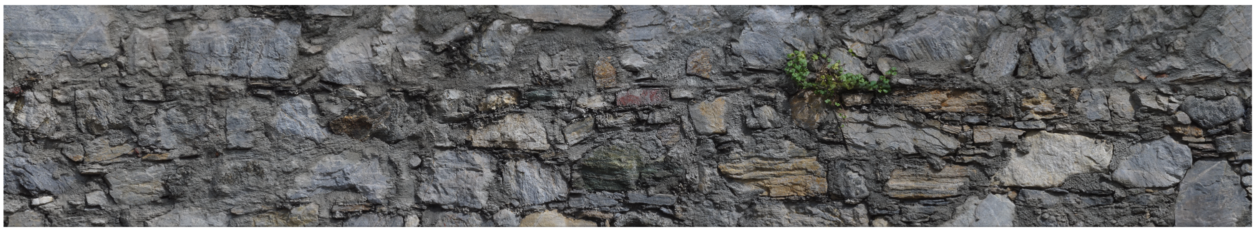

(b) Composite image of a stonewall

Figure 5.4: Two examples for stitched images: (a) $6.5\,\mathrm{MB}$ png-image in the dimensions 6400×972 px, composed of 3200 images and (b) $57.1\,\mathrm{MB}$ png-image in the dimensions 14260×2525 px, composed of 75 images.

Chapter 6

Conclusion and Outlook

It is concluded that the framework and workflow for hyper resolution image visualization established in this thesis meet the requirements of mineshaft inspection defined in Chapter 1. Furthermore the ability to use web mapping applications as framework for hyper resolution image visualization independent of their nature is demonstrated. Tiling an image with respect to a pyramid scheme is not only suited for web mapping applications, but for displaying hyper resolution images in general. This approach can be extended to be used with different software in order to handle visualization of very large datasets, both for vectorized and raster data.

Tests on tile servers from different providers have to be performed. This investigation allows decisions on whether to stay with the local solution, or to outsource image tiling, storage and tile serving to an external provider.

The integration of all components, from image acquisition in the mineshaft to providing data to the shaft inspector, into a fully automated system will be a future issue of the project *iDeepMon*. As the data volumes which the system needs to handle are truly massive, data transfer rates are a limiting factor and the transport of image data to a local storage and further to the data centre could be problematic. However, everything was prepared to guarantee a functioning process, once the data is at the data centre.

Appendix

During the work for this thesis, the author used own code as well as code that was available at the Chair of Automation.

MATLAB® is a registered trademark of The MathWorks, Inc.

Python® is a registered trademark of the Python Software Foundation.

NiFi™ is a registered trademark of the Apache Software Foundation in the United States and other countries.

JavaScript® is a trademark or registered trademark of Oracle in the U.S. and other countries.

Leaflet is developed by Vladimir Agafonkin, previously with CloudMade but is now employed by Mapbox.

List of Figures

List of Tables

List of Source Codes

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Maps for Mines</title>
    <meta charset="UTF-8">
    <!-- IMPORT -->
    <!--<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
    -->
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/Sources/w3.css">

    <!-- LEAFLET -->
    <link rel="stylesheet" href="/Sources/leaflet/leaflet.css">
    <script src="/Sources/leaflet/leaflet.js"></script>

    <!-- SIDEBAR -->
    <link rel="stylesheet" href="/Sources/leaflet-sidebar-master/src/L.Control
    .Sidebar.css">
    <script src="/Sources/leaflet-sidebar-master/src/L.Control.Sidebar.js"></
    script>

    <!-- JQUERY -->
    <script type="text/javascript" src="/Sources/jQuery.js"></script>

    <!-- LEAFLET DRAW -->
    <link rel='stylesheet' href='/Sources/leaflet/leaflet.draw.css' />
    <script src='/Sources/leaflet/leaflet.draw.js'></script>

    <!-- Page style -->
    <style>
        html, body {
            height: 100%;
        }
        #footer {
            position: fixed;
            bottom: 0;
            height: 50px;
            right: 0;
            width: 100%;
        }
        #mapContainer {
            position: fixed;
            top: 126px;
            bottom: 50px;
            width: 100%;
            right: 0;
        }
        #map {
            background: #f2f2f2;
        }
    </style>

    <script>
        function init() {

/**
 * -------------------------------------------------------------
 * GLOBAL VARIABLES
 * -------------------------------------------------------------
 */
            var tileSize = 256;
            var xImageLength = 14260;
            var yImageLength = 2525;
            var xLength = nextPowerOf2(xImageLength);
            var yLength = nextPowerOf2(yImageLength);
            var maxLength = Math.max(xLength, yLength);
            var scaleFactor = maxLength / tileSize;
            var overzoom = 0;
            var mapMinZoom = 0;
            var mapMaxZoom = Math.ceil(Math.log(scaleFactor) / Math.log(2)) +
    overzoom; //5
            var activeMark = "w3-pale-green";

/**
 * -------------------------------------------------------------
 * MAP, TILELAYERS
 * -------------------------------------------------------------
 */
            // Define coordinate reference system
            L.CRS.MySimple = L.extend({}, L.CRS.Simple, {
                transformation: new L.Transformation(1 / scaleFactor, 0, 1 /
    scaleFactor, 0)
            });

            //Specify map bounds (dimensions of the image)
            var mapBounds = L.latLngBounds([
                [yImageLength, 1], // southwest corner
                [1, xImageLength] // northeast corner
            ]);

            // Load tileLayers to variables
            var stoneWall = L.tileLayer('/Tiles/stoneWall/{z}/{x}/{y}.png', {
                minZoom: mapMinZoom,
                maxZoom: mapMaxZoom,
                bounds: mapBounds,
                attribution: "Images from Jakob Koenig",
                noWrap: true
            }),
                stoneWallInverted = L.tileLayer('/Tiles/stoneWallInverted/{z
    }/{x}/{y}.png', {
                    minZoom: mapMinZoom,
                    maxZoom: mapMaxZoom,
                    bounds: mapBounds,
                    noWrap: true
                }),
                geoData = L.tileLayer('/Tiles/geo/{z}/{x}/{y}.png', {
                    minZoom: mapMinZoom,
                    maxZoom: mapMaxZoom,
                    bounds: mapBounds,
                    noWrap: true,
                    opacity: 0.3
                });

            // Define Map
            var map = L.map('map', {
                maxZoom: mapMaxZoom,
                minZoom: mapMinZoom,
                crs: L.CRS.MySimple,
                zoomControl: false,
                layers: [stoneWallInverted, stoneWall]
            }).setView([0, 0], mapMinZoom + 2);

            map.fitBounds(mapBounds);

            var baseMaps = {
                "Inverted": stoneWallInverted,
                "Original": stoneWall
            };

/**
 * ---------------------------------------------------------------
 * SIDEBAR
 * ---------------------------------------------------------------
 */
            var sidebarContent = document.getElementById('listings');
            /**
             * Report List
             */
            function buildReportList(data) {
                // Iterate through the list of reports
                Object.keys(data).forEach(function(key){
                    var currentFeature = data[key].feature;
                    var prop = currentFeature.properties;
                    var listing = sidebarContent.appendChild(document.
    createElement('li'));
                    listing.className = 'w3-bar-item';
                    listing.classList.add("w3-border");
                    listing.classList.add("w3-hover-light-gray");
                    listing.id = key;
                    var link = listing.appendChild(document.createElement('a')
    );
                    link.href = '#';
                    link.className = 'title';
                    link.dataPosition = key;
                    link.innerHTML = "<h4>"+prop.name+"</h4>";
                    // Add an event listener for the links in the sidebar
    listing
                    link.addEventListener('click', function () {
                        var currentFeature = data[this.dataPosition].feature;
                        flyToPoint(currentFeature);
                        var activeItem = document.getElementsByClassName(
    activeMark);
                        if (activeItem[0]) {
                            activeItem[0].classList.remove(activeMark);
                        }
                        this.parentNode.classList.add(activeMark);
                        // Highlight marker on map
                        var jsonLayers = jsonLayerGroup.getLayers();
                        for(var j=0; j<jsonLayers.length;j++){
                            Object.keys(jsonLayers[j]._layers).forEach(
    function(key){
                                jsonLayers[j]._layers[key].setIcon(blueMarker)
    ;
                            });
                        }

                        data[this.dataPosition].setIcon(redMarker);
                    });

                    // Create a new div with the class 'details' for each
    report
                    // and fill it with the associated file
                    var coords = listing.appendChild(document.createElement('
    div'));
                    coords.className = "coords";
                    coords.innerHTML = "X: " + currentFeature.geometry.
    coordinates[0] + " || Y: " + currentFeature.geometry.coordinates[1];

                    var associatedFile = listing.appendChild(document.
    createElement('a'));
                    associatedFile.href = '#';
                    associatedFile.className = "associatedFile";
                    associatedFile.innerHTML = prop.file;
                    associatedFile.addEventListener("click", function () {
                        window.open("/Reports/PDFs/" + datum + "/" + this.
    innerHTML);
                    })
                });
            }

/**
```

```
 *  ------------------------------------------------------------
 *  OVERLAYS, REPORTS
 *  ------------------------------------------------------------
 */          var activeLayerID = null;
             var jsonLayerGroup = L.layerGroup();
             var datum;
             var reportFilesDirectory = '/Reports/Reports/';
             var fNames = [];

             // Create the control and add it to the map;
             var control = L.control.layers(baseMaps, null, {
                 position: 'topright',
                 hideSingleBase: true,
                 sortLayers: true
             }); // Grab the handle of the Layer Control, it will be easier to
   find.
             control.addTo(map);
             control.addOverlay(geoData, "Geodata");

             $.ajax({url: reportFilesDirectory}).then(function(html) {
                 // create temporary DOM element
                 var document = $(html);
                 // find all links ending with .pdf
                 document.find('a[href$=".json"]').each(function() {
                     var jsonName = $(this).text();
                     fNames.push(jsonName);
                 });
                 for(var f=0; f<fNames.length; f++){
                     (function (f) {
                         $.ajax({
                             dataType: "json",
                             url: reportFilesDirectory+fNames[f],
                             global: false,
                             success: function (data) {
                                 createOverlay(data, fNames[f]);
                             }
                         });
                     })(f);
                 }
             });

             // Pipes
             var pipesLayer = new L.geoJson();
             //pipesLayer.addTo(map);
             $.ajax({
                 dataType: "json",
                 url: "/Reports/pipes.json",
                 success: function(data) {
                     pipesLayer.addData(data);
                 }
             });
             control.addOverlay(pipesLayer, 'Pipes');

             map.on('overlayadd', onOverlayAdd);
```

```
        map.on('overlayremove', onOverlayRemove);

        var firstLoad = true;

        function onOverlayAdd(e) {
            // DRAW Layer
            if(e.layer._leaflet_id===getDrawFeutureGroup()._leaflet_id){
                if(activeLayerID!==null){
                    var activeLayer = jsonLayerGroup.getLayer(
    activeLayerID);
                    var drawL = getDrawFeutureGroup();
                    var layers = activeLayer.getLayers();

                    setTimeout(function() {
                        map.removeLayer(activeLayer);
                    }, 1);

                    setTimeout(function() {
                        for(var l=0; l<layers.length; l++){
                            drawL.addLayer(layers[l]);
                        }
                    }, 5);
                }
            }

            // JSON Layer with Reports
            if(jsonLayerGroup.hasLayer(e.layer)){
                if(firstLoad && sbOpen===false){
                    w3_toggle();
                    firstLoad = false;
                }
                $('#listings').empty();
                var layerName = e.name;
                var layerID = e.layer._leaflet_id;
                datum = layerName.substr(0, layerName.lastIndexOf('.')) ||
     layerName;
                buildReportList(e.layer._layers);
                // Exclusive JSON Layers
                jsonLayerGroup.eachLayer(function (layer) {
                    if(layer._leaflet_id !== layerID){
                        setTimeout(function() {
                            map.removeLayer(layer);
                        }, 1);
                    }
                });
                activeLayerID = layerID;
            }
        }

        function onOverlayRemove(e) {
            var layerID = e.layer._leaflet_id;
            if(layerID===activeLayerID){
                activeLayerID = null;
            }
        }

        function createOverlay(data, layerName) {
            var overlay = L.geoJSON(data, { // Make the layer from the
    JSON and grab the handle.
                onEachFeature: onEachFeature,
                pointToLayer: function (feature, latlng) {
                    var marker = L.marker(latlng, {icon: blueMarker});
                    marker.on("click", function (e) {
                        var jsonLayers = jsonLayerGroup.getLayers();
                        for(var j=0; j<jsonLayers.length;j++){
                            Object.keys(jsonLayers[j]._layers).forEach(
    function(key){
                                    jsonLayers[j]._layers[key].setIcon(
    blueMarker);
                            });
                        }
                        e.target.setIcon(redMarker);
                        flyToPoint(e.target.feature);
                        var activeItem = document.getElementsByClassName(
    activeMark);
                        if (activeItem[0]) {
                            activeItem[0].classList.remove(activeMark);
                        }
                        $("div #" + e.target._leaflet_id).addClass(
    activeMark);
                        //Scroll to active element on the sidebar
                        /*location.href = '#';
                        location.href = '#' + e.target._leaflet_id;*/
                    });
                    return marker;
                }
            });
            //overlay.addTo(map); // Add the data to the map
            jsonLayerGroup.addLayer(overlay);
            control.addOverlay(overlay, layerName); // Add the layer to
    the Layer Control.
        }

        function onEachFeature(feature, layer) {
            layer.on('contextmenu', function (e) {
                rcPopup(e, "marker");
            });
        }

        var enableFlyTo = true;
        function flyToPoint(currentFeature) {
            if(enableFlyTo){
                map.flyTo(currentFeature.geometry.coordinates.reverse(),
    mapMaxZoom - overzoom, {
                    animate: true,
                    duration: 2 // in seconds
                });
                currentFeature.geometry.coordinates.reverse();
            }
        }

/**
*  -------------------------------------------------------------
*  CUSTOM MARKERS
*  -------------------------------------------------------------
*/
        var markerUrl = '/Sources/Icons/';
        var markerIcon = L.Icon.extend({
            options: {
                shadowUrl: markerUrl+'marker-shadow.png',
                iconAnchor: [13, 40]
                //shadowAnchor: [2, 58],
                //popupAnchor: [1, -36]
            }
        });

        var blueMarker = new markerIcon({iconUrl: markerUrl+'
markerInactive.png'}),
            redMarker = new markerIcon({iconUrl: markerUrl+'markerActive.
png'});

/**
*  -------------------------------------------------------------
*  ORIGINAL IMAGE BACKTRACKING
*  -------------------------------------------------------------
*/
        var imgNames;
        $.ajax({
            dataType: "json",
            url: "/OriginalImages/jsonNames.json",
            success: function (data) {
                imgNames = data;
            }
        }).error(function () {
        });

        var imgRefMatrix;
        $.ajax({
            dataType: "json",
            url: "/originalImages/jsonRefMatrixV2.json",
            success: function (data) {
                imgRefMatrix = data;
            }
        }).error(function () {
        });

        map.on('contextmenu', function (e) {
            rcPopup(e, "map");
        });

        function rcPopup(e) {
            var x = Math.round(e.latlng.lng);
            var y = Math.round(e.latlng.lat);
            L.popup()
                .setContent("<h6>X:" + x + "<br>Y:" + y + "</h6><a class
=\"plink\" href=\"#\">" +
                    "<h6>Show Original Images</h6></a>")
                .setLatLng([e.latlng.lat, e.latlng.lng])
                .openOn(map);

            var imgName = [];
            for(var imgID=1; imgID<=imgRefMatrix.length; imgID++){
                var xMin = imgRefMatrix[imgID-1][0];
                var xMax = imgRefMatrix[imgID-1][1];
                var yMin = imgRefMatrix[imgID-1][2];
                var yMax = imgRefMatrix[imgID-1][3];
                if( (x>=xMin)&&(x<=xMax)&&(y>=yMin)&&(y<=yMax) ){
                    imgName.push(imgNames[imgID-1]);
                }
            }

            var linkToImg = $("a.plink");
            linkToImg.on("click",function(){
                for(var g=0;g<imgName.length; g++){
                    $("#slideImages").append("<a href=\"#\" onclick=\"
window.open('/OriginalImages/Images/"+
                        imgName[g]+".JPG', '_blank');\">\n"+"<img class=\"
mySlides\" src=\"/OriginalImages/Images/"+
                        imgName[g]+".JPG\" style=\"width:100%\">\n"+"</a>"
);
                    $("#slideButtons").append("<button class=\"w3-button
demo\" onclick=\"currentDiv("+(g+1)+")\">"+
                        (g+1)+"</button>")
                    //window.open('/originalImages/Images/' + imgName[g]+
'.JPG');
                }
                openModal();
            });
        }

/**
* ------------------------------------------------------------
* LEAFLET DRAW
* ------------------------------------------------------------
*/
        var drawFeatureGroup = L.featureGroup();
        //drawFeatureGroup.addTo(map);
        control.addOverlay(drawFeatureGroup, 'Drawlayer');

        var drawControl = new L.Control.Draw({
            position: 'topright',
            draw: {
                polyline: false,
                polygon: false,
                rectangle: false,
                circle: false
            },
            edit: {
                featureGroup: drawFeatureGroup
            }
        });
        drawControl.addTo(map);

        map.on('draw:editstart', function(){
            enableFlyTo = false;
        });
        map.on('draw:editstop', function(){
            enableFlyTo = true;
        });

        map.on('draw:deletestart', function(){
            enableFlyTo = false;
        });
        map.on('draw:deletestop', function(){
            enableFlyTo = true;
        });

// From https://jsfiddle.net/ve2huzxw/314/
// https://stackoverflow.com/questions/34738805/update-properties-of-
geojson-to-use-it-with-leaflet/34740632#34740632
        map.on('draw:created', function(e) {
            // Each time a feature is created, it's added to the over
arching feature group
            var layer = e.layer,
                feature = layer.feature = layer.feature || {};
            feature.type = feature.type || "Feature";
            var props = feature.properties = feature.properties || {};
            props.name = null;
            props.file = "";
            drawFeatureGroup.addLayer(layer);
            addPopup(layer);
        });

        function addPopup(layer) {
            var content = document.createElement("textarea");
            content.addEventListener("keyup", function () {
                layer.feature.properties.name = content.value;
            });
            layer.on("popupopen", function () {
                content.value = layer.feature.properties.name;
                content.focus();
            });
            layer.bindPopup(content).openPopup();
        }

        // on click, clear all layers
        document.getElementById('clear').onclick = function() {
            drawFeatureGroup.clearLayers();
        };
        document.getElementById('export').onclick = function() {
            // Extract GeoJson from featureGroup
            var data = drawFeatureGroup.toGeoJSON();
            //console.log(JSON.stringify(data, null, 2));

            // Stringify the GeoJson
            var convertedData = 'text/json;charset=utf-8,' +
encodeURIComponent(JSON.stringify(data));

            // Create export
            //var saveName = prompt("Enter Filename", 'new.json');
            document.getElementById('export').setAttribute('href', 'data:'
 + convertedData);
            //document.getElementById('export').setAttribute('download',
saveName);
            document.getElementById('export').setAttribute('download', '
new.json');
        };


/**
 *  --------------------------------------------------------------
 *  ADDITIONAL FUNCTIONS
 *  --------------------------------------------------------------
 */
        function nextPowerOf2(number){
            var power = Math.ceil(Math.log(number)/Math.log(2));
            return Math.pow(2,power);
        }

        function getDrawFeutureGroup(){
            return drawFeatureGroup;
        }
    }
</script>

</head>
<body onload="init()">

<div class="w3-sidebar w3-bar-block w3-card-2 w3-animate-left" style="display:
none; z-index:2" id="sidebar">
<div class="w3-container w3-blue-gray" style="position:fixed; z-index:1;
width:20%">
    <h2>Reports</h2>
</div>
<ul id="listings" class="w3-ul w3-border w3-hoverable" style="position:
relative; top:65px"></ul>
</div>

<div class="w3-main" id="main" style="height: 100%; background-color: #f2f2f2"
>

<div class="w3-teal" id="header">
    <button style="position: relative; z-index: 1" class="w3-button w3-
```

```
teal w3-xlarge" onclick="w3_toggle()">
            &#9776;
        </button>
        <div class="w3-container">
            <h1>Maps for Mines</h1>
        </div>
        <img src="/Sources/ideepmon-logo.svg" alt="projectLogo" style="width
:200px; height:200px; position:absolute;
            z-index:1; top: -30px; right: 200px">
        <img src="/Sources/uniLogo.png" alt="uniLogo" style="width:110px;
height:110px; position:absolute; z-index:1;
            top: 8px; right: 20px">
    </div>

    <div class="w3-display-container" id="mapContainer">
        <div id="map" class="Middle"  style="height: 100%; width: 100%; z-
index: 0">
        </div>
        <a href='#' id='clear' style="position:absolute; right: 10px; bottom:
40px; z-index: 2">
            <img src="/Sources/Icons/clearBtn.png" style="height:30px; width
:30px;">
        </a>
        <a href='#' id='export' style="position: absolute; right: 10px; bottom
: 80px; z-index: 2">
            <img src="/Sources/Icons/exportFileBtn.png" style="height:30px;
width:30px;">
        </a>
    </div>
    <div class="w3-container w3-teal w3-bottom w3-right-align" id="footer">
        <p>&copy; 2017 Chair of Automation, Leoben, Austria, Michael Brandner</
p>
    </div>
    <!--
    <div>
        Icons made by <a href="https://www.flaticon.com/authors/google" title=
"Google">Google</a> from
        <a href="https://www.flaticon.com/" title="Flaticon">www.flaticon.com
</a> is licensed by
        <a href="http://creativecommons.org/licenses/by/3.0/" title="Creative
Commons BY 3.0" target="_blank">CC 3.0 BY
        </a>
    </div>
    <div>
        Icons made by <a href="https://www.flaticon.com/authors/anton-saputro"
 title="Anton Saputro">Anton Saputro</a>
        from <a href="https://www.flaticon.com/" title="Flaticon">www.flaticon
.com</a> is licensed by
        <a href="http://creativecommons.org/licenses/by/3.0/" title="Creative
Commons BY 3.0" target="_blank">CC 3.0 BY
        </a>
    </div>
    -->
</div>
```

```
<div id="myModal" class="w3-modal w3-black" style="width:100%">
    <span class="w3-text-white w3-xxlarge w3-hover-text-grey w3-container w3-
    display-topright" onclick="closeModal()"
          style="cursor:pointer">x</span>
    <div class="w3-modal-content w3-black" style="width:100%">

        <div class="w3-content w3-black w3-animate-zoom" id="slideImages"
    style="width:100%">
            <div class="w3-center w3-black" id="slideButtons">
                <div class="w3-section">
                    <button class="w3-button w3-light-grey" onclick="plusDivs
    (-1)">Prev</button>
                    <button class="w3-button w3-light-grey" onclick="plusDivs
    (1)">Next</button>
                </div>
            </div>
        </div> <!-- End w3-content -->
    </div> <!-- End modal content -->
</div> <!-- End modal -->

<!-- TOGGLE SIDEBAR -->
<script>
    var sbOpen = false;
    function w3_toggle() {
        if(sbOpen){
            document.getElementById("main").style.marginLeft = "0%";
            document.getElementById("sidebar").style.display = "none";
            sbOpen = false;
        }else{
            document.getElementById("main").style.marginLeft = "20%";
            document.getElementById("sidebar").style.width = "20%";
            document.getElementById("sidebar").style.display = "block";
            sbOpen = true;
        }
    }
</script>

<!-- IMAGE SLIDESHOW -->
<script>
    var slideIndex = 1;
    function openModal() {
        document.getElementById('myModal').style.display = "block";
        showDivs(slideIndex);
    }
    function closeModal() {
        document.getElementById('myModal').style.display = "none";
        $(".mySlides").remove();
        $(".demo").remove();
        slideIndex = 1;
    }
    function plusDivs(n) {
        showDivs(slideIndex += n);
    }
    function currentDiv(n) {
        showDivs(slideIndex = n);
    }
    function showDivs(n) {
        var i;
        var x = document.getElementsByClassName("mySlides");
        var dots = document.getElementsByClassName("demo");
        if (n > x.length) {
            slideIndex = 1
        }
        if (n < 1) {
            slideIndex = x.length
        }
        for (i = 0; i < x.length; i++) {
            x[i].style.display = "none";
        }
        for (i = 0; i < dots.length; i++) {
            dots[i].className = dots[i].className.replace(" w3-blue-gray", "")
;
        }
        x[slideIndex - 1].style.display = "block";
        dots[slideIndex - 1].className += " w3-blue-gray";
    }
</script>
</body>
</html>
```

Source Code 1: Maps for Mines

Bibliography

[1] N Benecke, 'iDeepMon – intelligent deep mine shaft inspection and monitoring', 2017, pp. 411–418, ISBN: 9780992481070.

[2] *DMT*, 2017. [Online]. Available: `http://www.dmt-group.com/en/about-us/about-dmt.html` (visited on 21/10/2017).

[3] M. Brandner, 'Generic Data Ingestion Method for Machine Data', Bachelor Thesis, University of Leoben, 2016.

[4] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, 2nd ed. Cambridge: Cambridge University Press, 2003, ISBN: 0521540518.

[5] A Badshah, P O'Leary, M Harker and C Sallinger, 'Non-rigid registration for qualitiy control of printed materials', in *Tenth International Conference on Quality Control by Artificial Vision*, ser., vol. 8000, 2011, 80000J. DOI: `10.1117/12.890901`.

[6] A Badshah, P O'Leary, M Harker and D Tscharnuter, 'Strain analysis by regularized non-rigid registration', in *Image Processing: Machine Vision Applications V*, ser., vol. 8300, 2012, p. 83000D. DOI: `10.1117/12.920003`.

[7] *Slippy Map – OpenStreetMap Wiki*. [Online]. Available: `https://wiki.openstreetmap.org/wiki/Slippy_Map` (visited on 24/10/2017).

[8] *Tiles à la Google Maps: Coordinates, Tile Bounds and Projection*. [Online]. Available: `http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/` (visited on 26/10/2017).

[9] *Map and Tile Coordinates — Google Maps JavaScript API — Google Developers*. [Online]. Available: `https://developers.google.com/maps/documentation/javascript/coordinates` (visited on 26/10/2017).

[10] *Bing Maps Tile System*. [Online]. Available: `https://msdn.microsoft.com/en-us/library/bb259689.aspx` (visited on 26/10/2017).

[11] J. P. Snyder, 'Map Projections: A Working Manual', *U.S. Geological Survey Professional Paper 1395*, p. 383, 1987, ISSN: 00941689. DOI: `10.2307/1774978`. arXiv: `arXiv:1011.1669v3`.

[12] J. P. Synder, 'Emergence of Map Projections, from Flattening the Earth: Two Thousand Years of Map Projections', in *The Map Reader: Theories of Mapping Practice and Cartographic Representation*, 2011, pp. 164–169, ISBN: 9780470742839. DOI: `10.1002/9780470979587.ch23`.

[13] E. W. Weisstein, *Cylindrical Projection*. [Online]. Available: http://mathworld.wolfram.com/CylindricalProjection.html (visited on 10/11/2017).

[14] ——, *Mercator Projection*. [Online]. Available: http://mathworld.wolfram.com/MercatorProjection.html (visited on 10/11/2017).

[15] *Images as Maps - macwright.org*. [Online]. Available: https://macwright.org/2012/08/13/images-as-maps.html (visited on 02/11/2017).

[16] J.-F. Faudi, *gdal2tilesG.py*. [Online]. Available: https://gist.github.com/jeffaudi/9da77abf254301652baa (visited on 14/08/2017).

[17] FileFormat.info, *JPEG File Interchange Format File Format Summary*, 2015. [Online]. Available: http://www.fileformat.info/format/jpeg/egff.html (visited on 07/10/2017).

[18] J. König, 'Automated System for Panoramic Depth Imaging and Visualization', Master Thesis, University of Leoben, 2017.

[19] *The TRL Scale as a Research Innovation Policy Tool, EARTO Recommendations*, 2014. [Online]. Available: http://www.earto.eu/fileadmin/content/03_Publications/The_TRL_Scale_as_a_R_I_Policy_Tool_-_EARTO_Recommendations_-_Final.pdf (visited on 21/10/2017).

[20] S. Kolios, A. V. Vorobev, G. R. Vorobeva and C. Stylios, 'Geographic Information Systems', in *GIS and Environmental Monitoring: Applications in the Marine, Atmospheric and Geomagnetic Fields*. Cham: Springer International Publishing, 2017, pp. 3–45, ISBN: 978-3-319-53086-4. DOI: 10.1007/978-3-319-53086-4_1. [Online]. Available: https://doi.org/10.1007/978-3-319-53086-4_1.

[21] *Leaflet - a JavaScript library for interactive maps*. [Online]. Available: http://leafletjs.com/ (visited on 03/11/2017).

[22] *Tile Map Service Specification - OSGeo*. [Online]. Available: http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification (visited on 26/10/2017).

[23] H. Butler, M. Daly, H. Inc., Cadcorp and A. Doyle, *The JavaScript Object Notation (JSON) Data Interchange Format*, T. Bray, Ed., 2014. DOI: 10.17487/rfc7159. [Online]. Available: https://tools.ietf.org/html/rfc7946 (visited on 03/11/2017).

[24] *PNG (Portable Network Graphics) Specification , Version 1 . 2*, 1999. [Online]. Available: http://www.libpng.org/pub/png/spec/1.2/png-1.2.pdf (visited on 18/10/2017).