Institute for Automation

Department Product Engineering,

Montanuniversität Leoben

Automated System for Panoramic Depth Imaging and Visualization

MASTER THESIS

Jakob König

Field of Study:
Mechanical Engineering

Supervisor:

PRIV.-DOZ. DR.MONT. MATTHEW HARKER
O.UNIV.-PROF. DIPL.-ING. DR.TECHN. PAUL O'LEARY

Abstract

This thesis presents the development and verification of concepts, methods and possible implementations of a system for the automatic visual inspection of deep mine shafts. Although the concept is developed for shafts, it can be applied to any large object for which a contiguous visual surface inspection is required. The concept is based on mosaicking large numbers of pictures, e.g. 60000, to structured panoramas which can be mapped as texture on a geometric model. The large number of images together with their resolution enable a precise metric representation of the observed surfaces. These are stored in a reduced resolution set format. This data model is consistent with open source mapping tools so that data fusion from different sources, e.g. depth based geological information, may be performed. Standard web based visualisation tools can then be used to view the data.

Zusammenfassung

Diese Diplom-Arbeit befasst sich mit der Entwicklung und Verifizierung von Konzepten, Methoden und möglichen Implementationen eines Systems für die automatische visuelle Inspektion von vertikalen Minenschächten. Obwohl die Konzepte für Minenschächte entwickelt wurden, können sie im Allgemeinen für die visuelle Inspektion von großen Objekten mit geschlossenen Oberflächen verwendet werden. Durch das Zusammenfügen einer hohen Zahl, ca. 60000, hochauflösender Bilder werden strukturierte Panoramen erstellt, die als Textur auf geometrischen Objekten abgebildet werden können. Die Kombination aus Anzahl und Auflösung der Bilder ermöglicht eine präzise metrische Darstellung der betrachteten Oberfläche. Die Panoramen werden als Reduced Resolution Set gespeichert. Dieses Format macht es möglich die Bilder mit Daten aus anderen Quellen, zum Beispiel geologischen Tiefeninformationen, zu verknüpfen. Zur Darstellung von Daten, die in dieser Form gespeichert werden, können öffentlich zugängliche Kartographie Applikationen wie z.B. Google Maps verwendet werden.

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 an Eides statt, dass ich diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Leoben, Tue 14th Mar, 2017 Jakob König

Acknowledgements

To all those who shared their thoughts and time with me so I could grow and learn from their experience I say with love and respect, thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents the development and verification of concepts, methods and possible implementations of a system for the automatic visual inspection of vertical deep mine shafts.
The system was developed for the KIC-Raw Materials Project entitled Maintained Mining Machines. The goal of this project is to create a holistic maintenance support system for modern mining operations. Mine maintenance plans include a periodic visual inspection of vertical deep mine shafts. These inspections are important because changes in the shaft surface, for example crack propagation, and the reason for these changes, for example geological events in the area surrounding the shaft, can be indicative of serious dangers to the structural integrity of the shaft.
The process of manually inspecting the shaft surface puts humans in potentially dangerous situations. Developing a system for the automatic visual inspection of vertical deep mine shafts is therefore important to reduce the risk to the mine inspection personnel by providing means to perform the inspections remotely.
The challenge in creating a remote inspection system lies in creating a representation of a surface that is too large to be captured with one image. For a vertical mine shaft which is roughly cylindrical creating one image of the entire surface is the equivalent of creating an image with a field of view of 360° and the same height as the shaft. The problem of creating images with a field of view of 360° has been thoroughly studied by many researchers [1]. One approach, discussed by S.Peleg et al. [2], uses a camera that rotates around a vertical center axis and captures images at increasing angles of rotation; whereby each angle is chosen in a way that creates horizontally overlapping images. Such horizontally overlapping images can also be created by a fixed number of cameras positioned at evenly spaced angles over 360°, see Figure 1.1. These overlapping images can then be used to produce a panoramic image covering a field of view of 360°, as discussed by Szeliski [3]. To cover the height of the mine shaft the vertical position of the center of the camera system is adjusted in increments that ensure a vertical overlap between each set of horizontal images. Every horizontal set of images is used to produce a panoramic im-

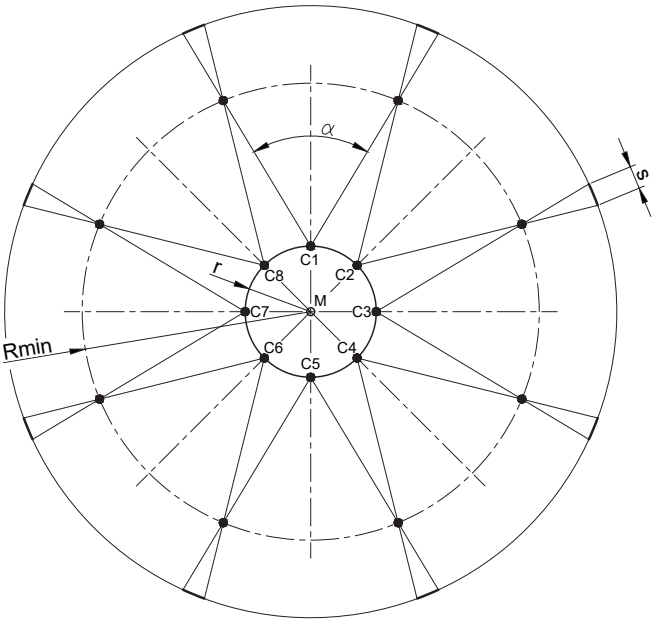


Figure 1.1: Camera system composed of eight cameras $C1$ to $C8$ at a distance r from the center point M observing a cylindrical surface. R_{min} denotes the radius at which the fields of view α overlap. s denotes the overlap between images on the surface.

age.These vertically overlapping panoramic images can then be used to create one large image of the entire shaft surface by applying similar techniques used for the horizontal panoramas as shown by Brown and Lowe [4].

Another challenge that comes with a remote inspection system is the need for a representation of the shaft surface that can be used to perform measurements on objects in the surface. The representation must therefore provide data in three dimensional space, length, width and depth. Measurements in two dimensional space, i.e. length and width, in an image through rectification has been addressed by Szeliski [3]. Peleg et al. have created an approach to regain the depth information from panoramic images created with a single rotating camera [2].

In order to perform the visual inspections of the deep mine shaft remotely the panoramic images created by the system must be presented in a way that can be used by the mine personnel to perform each aspect of an inspection of the entire shaft surface. By using techniques described by Peleg et al. [2] to create the panoramic images, the software krpano [5] provides the possibility of creating several types of virtual reality tours. Virtual Reality refers to computer technologies that use software to replicate a real environment to be viewed with a Head Mounted Display. Viewing these tours with a Head Mounted Display, the inspection personnel can perform the inspection of the shaft in a virtual recreation of the shaft from a remote location.

Using virtual representations for the remote inspection also gives the ability to add information from other sources, such as geological depth maps, to correlate geological events

and changes in the shaft surface. This technique has been used by researchers in many fields to find connections between geological events, based on their geographic location, and their ramifications [6] and is called Geographic Information System GIS [7].

This thesis is a contribution to the Maintained Mining Machine Project by KIC-Raw Materials in the form of a feasibility study of an automatic panoramic depth imaging system and the visualization of the acquired data for the visual inspection of vertical deep mine shafts. Known methods are tested and applied to create a system capable of generating panoramic images of large contiguous surfaces and visualize them in a way that allows the mine personnel to perform inspections remotely.

1.1 Overview

Chapter 2 shows the development of a single rotating camera system for panoramic imaging. Chapter 4 gives a short introduction to the most important concepts of image processing. In Chapter 5 these concepts are applied to the geometric registration process of panoramic imaging in general, and for this specific project. In Chapter 6 the visualization of the panoramic images is discussed. The data handling of the system is shown in Chapter 7. Chapter 8 gives a conclusion and outlook for future work.

Chapter 2

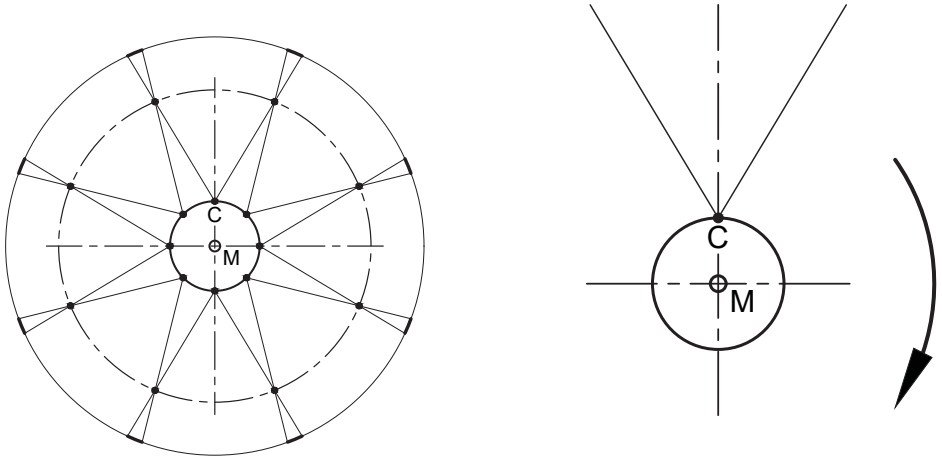
Panoramic imaging system

This chapter describes the design stages and the decision making process for the laboratory set up of the automated panoramic imaging system.
To create a panoramic image as described in the introduction, the system needs to be able to capture images at different angles of rotation. The system is controlled with the embedded system Raspberry Pi 3 Model B [8]. The rotational angle between the images can be achieved by either having a fixed number of cameras pointing in different directions with the same rotational axis, Figure 2.1a, or using one camera that can rotate around a center axis 2.1b.
The difficulty that arises with a system that uses multiple cameras controlled by a Raspberry Pi 3 Model B is its inability to activate the cameras at the same time. This drawback combined with the advantage of a system with a rotating camera of being able to control the number of images per rotation made a prototype with a rotating camera controlled by a Raspberry Pi 3 Model B the clear choice for the investigation into the feasibility of an automated panoramic depth imaging system for the Maintained Mining Maintenance Project.
Section 2.1 shows the capabilities of the Raspberry Pi 3 Model B in conjunction with its camera module. During the first stage, described in section 2.2, it was important to test the capabilities of the Raspberry Pi 3 Model B and its camera module and determine their limiting factors.Section 2.3 describes the first manually operated version of the rotating camera system. The results from prototype A2 made it possible to build a fully automated prototype, which is described in Chapter 3.

2.1 Processing device and camera

The Raspberry Pi 3 Model B [8] is a cheap and small embedded processing device. Some of its features [9] are listed here:

1. BroadcomBCM2387 chipset



(a) Fixed number of cameras (b) Rotating camera

Figure 2.1: Panoramic capturing system realized with a fixed number of cameras, Figure 2.1a, and a single rotating camera, Figure 2.1b. *M* denotes the center of rotation, *C* the center of the camera.

2. 1.2GHZ Quad-Core ARM Cortex A53
3. 802.11 bgn Wireless LAN and Bluetooth 4.1
4. 1 GB RAM
5. 4 USB ports
6. 10/100 BaseT Ethernet socket
7. CSI Camera port
8. Micro SD port
9. Micro USB power source

In Figure 2.2 the Raspberry Pi 3 Model B is shown in a top down perspective. The most important features for this study include the wifi capability, the csi port with the corresponding camera module and the general purpose IN/OUT (GPIO) pins.
The wifi module makes it possible to control the system remotely with a program called VNC viewer [10]. This program can take control of a device which is running a so called VNC server and is connected to the same local network. Editing the start up options of the Raspberry Pi 3 Model B enables an automatic start of such a server. This means that once the Raspberry Pi 3 Model B starts, it can be controlled by another device in the same network with the VNC viewer.
Together with the release of the Raspberry Pi 3 Model B, a new camera version, called

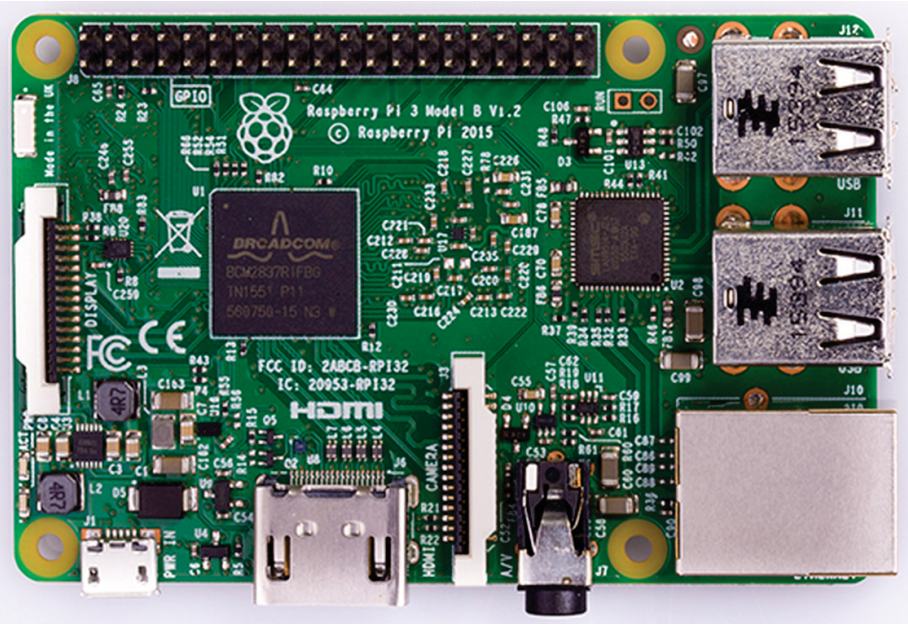


Figure 2.2: Raspberry Pi 3 Model B top down view [14].

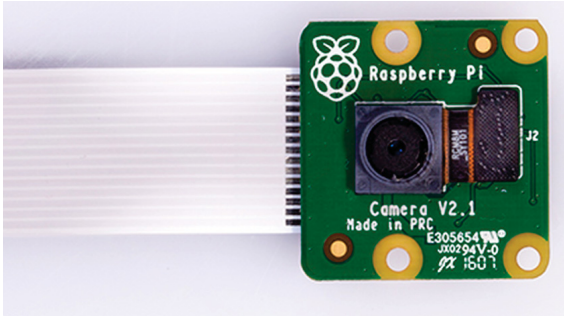


Figure 2.3: Camera Module v2.0 for the Raspberry Pi 3 Model B [15].

camera module v 2.0 [11] was released which uses a Sony IMX 219 PQ sensor [12]. At frame rates below 30 fps this sensor allows for resolutions up to 3280 by 2464 pixels per frame [13], see Table 2.1, with a horizontal field of view α of 62.2°. This angle is important since it is closely related to number of images needed to create panoramic images, see Chapter 4.

The GPIO pins on the Raspberry Pi 3 Model B, see Figure 3.3, can be controlled through different programming techniques and provide a logic signal of either zero or 3.3V to external electrical devices and components.

2.2 Prototype A1: Camera investigation

The first prototype was used to determine the capabilities of the Raspberry Pi 3 Model B and the camera module v 2.0. To test different settings of the camera it was mounted on a 3D printed holder and placed on the markings, seen in Figure 2.4. The angles between the markings represent eight camera positions evenly spaced to cover 360°. Capturing images at each marking produces eight horizontally overlapping images. From these overlapping images a panoramic image with a field of view of 360° can be produced. A number of

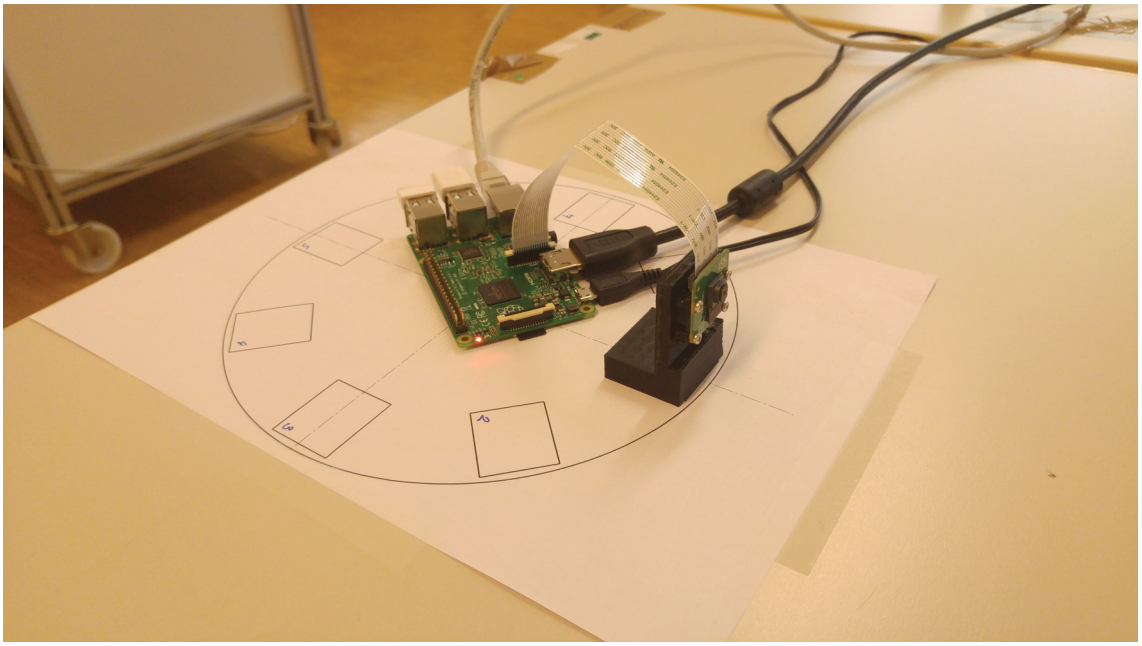


Figure 2.4: The first prototype was used to test the Raspberry Pi 3 Model B hardware.

conclusions can be drawn from the experiments with this prototype:

1. The camera requires approximately 1s to adapt to lighting conditions prior to acquiring an image
2. The camera can produce images in a number of different formats, see Table 2.1.
3. A rotating camera head, with a high rotational positioning accuracy, is required to investigate the limits of this image acquisition technique.

Figure 2.5: Panoramic image created with the software AutoStitch [16] from eight images, each with a resolution of 640 by 480 pixels.

Table 2.1: Possible resolutions for image acquisition with the camera module v2.0 [17] and the Raspberry Pi 3 Model B.

#	Resolution	Aspect Ratio	Framerates	Video	Image	FoV	Binning
1	1920x1080	16:9	0.1-30fps	x		Partial	None
2	3280x2464	4:3	0.1-15fps	x	x	Full	None
3	3280x2464	4:3	0.1-15fps	x	x	Full	None
4	1640x1232	4:3	0.1-40fps	x		Full	2x2
5	1640x922	16:9	0.1-40fps	x		Full	2x2
6	1280x720	16:9	40-90fps	x		Partial	2x2
7	640x480	4:3	40-90fps	x		Partial	2x2

2.3 Prototype A2: Panoramic image characterization

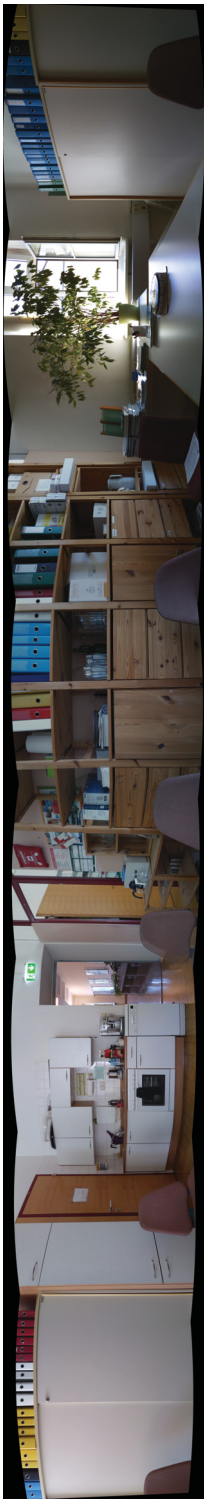
The goal of the second prototype, is to manually investigate different degrees of horizontal overlapping between the sequences of images. The camera module v 2.0 and the Raspberry Pi 3 Model B were mounted on a Manfrotto 410 tripod [18], which has spirit levels built in. Using the spirit levels to level the system it was possible to take images without vertical tilt, the importance of this feature is discussed in Chapter 4.
An increasing number of images per rotation with varying resolutions between sets where captured with the system. Each set of images was used to create a panoramic image with the software AutotStitch [16]. The following conclusions can be drawn from working with this prototype:

1. A slit-camera approach provides the simplest means of generating distortion free panoramas. It trades off the time required to acquire a higher number of images against post processing computations effort.

2. The slit-camera enables stereoscopic panoramas to be generated, see Section

3. All currently available APIs to generate panoramic images assume a single viewpoint. This prerequisite can not be fulfilled in the application being addressed. Consequently the development of a device to inspect mine shafts will require the programming of a dedicated stitching algorithm.

4. A motorized rotating camera system would enable a flexible variable resolution generation of panoramic images.



(a) Single Layer Panorama

(b) Multi Layer Panorama

Figure 2.6: Figure 2.6a shows a single layer panoramic image created from eight images with the software AutoStitch [4]. Figure 2.6b shows a panoramic image created from three layers of eight images each, with the software AutoStitch.

Chapter 3

Automatic variable resolution system

3.1 Mechanical

This chapter describes the development of the automatic panoramic imaging system with variable resolution. The system will be referred to as prototype A3. A camera rotated by a stepper motor controlled by a Raspberry Pi 3 Model B are the core parts of the automatic panoramic imaging system. Figure 3.1 shows the CAD assembly for the main components of the system. Table 3.1 contains the corresponding part list. Controlling the system is a Raspberry Pi 3 Model B [8]. A number of brackets and holders to enable the assembly of the system were designed and produced using a FDM 3D-Printer. A Nema 17 stepper motor [19] combined with an Easydriver [20] stepper motor controller are used to rotationally position the camera. The power for the components comes from an Anker powerbank [21].The power for the stepper motor is adjusted with a boost converter [22]. The camera [11] is mounted 10 cm from the center of rotation looking outward. A limit switch in combination with the 3D printed base plate of the system is used to create a reference point. The base plate is connected to a quick release plate for the Manfrotto tripod [18]. The 3D printed part on which the components are mounted and which in turn is mounted on the Manfrotto tripod [18] were designed using CAD software. The plans can be found in Appendix B.

Table 3.1: Part list for the single camera panoramic depth imaging system. The number corresponds to the one in Figure 3.1.

Part number	Part	Part number	Part
1	Raspberry Pi Camera Module v 2.0	6	Raspberry Pi 3 Model B
2	Anker Powerbank	7	Boost Converter
3	Rolling Limit Swtich	8	Nema 17 Stepper Motor
4	Easy Driver Stepper Driver	9	3D printed base plate
5	3D printed mounting plate	10	Manfrotto Quick Release Plate

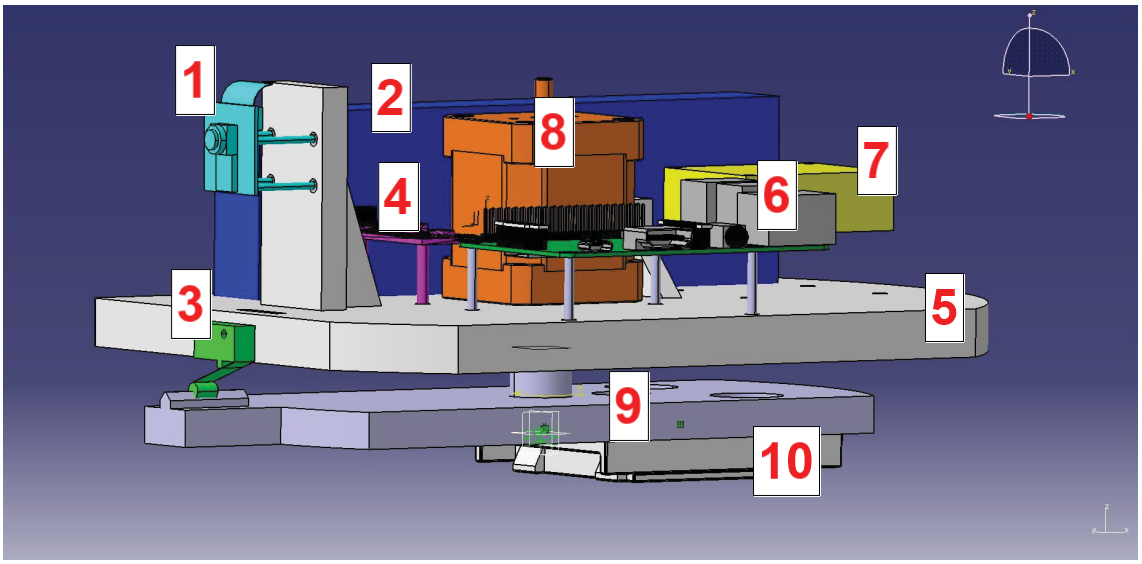


Figure 3.1: Isometric view of the CAD assembly of the automated panoramic imaging system.

3.1.1 Electric Circuit

The system was designed in a manner that made it possible to test it in different environments. This made a portable power source necessary. As power source the Anker powerbank with a capacity of 20100 mAh was chosen because it has two USB power ports that can provide power simultaneously. One port powers the Raspberry Pi 3 Model B with a standard USB to MicroUSB cable. The second port provides the power for the stepper motor, to this end the power out of the powerbank is connected to the boost converter. The boost converter uses DC power input. A standard USB to microUSB cable was adapted so it could be used to connect the two components.
The powerbank can provide a voltage of U_B =5V at a current of I_B =4.8A. A Raspberry Pi 3 Model B under maximum load has an approximate power consumption of P_{Pi} =2.6W with a voltage of U_{Pi} =5V.

$$P = UI \tag{3.1}$$

From Equation 3.1 follows that the Raspberry Pi 3 Model B needs a current of approximately I_{Pi} =520 mA under maximum load. After adding another 15 percent for reserve the design current needed for the Raspberry Pi 3 Model B is about I_D =600mA. This means that the powerbank can provide a current up to 4.2 A to the stepper motor. The power for the stepper motor driver is provided by the Raspberry Pi 3 Model B through GPIO, see Section 3.1.2.
A stepper motor is a synchronous electrical motor, i.e. the rotors halting positions are in synchronization with the stator flux. When the motor is in the so called full step mode, the stator flux is rotated by 90 degrees every step, these are called two phase on positions. To

achieve this only two current modes are needed, I_{on} and I_{off}. With a stepper motor driver that is able to create current levels between I_{on} and I_{off} it is possible to halt the stator flux at any degree [23], this is called micro stepping. The holding torque is the torque which can be held by the motor while preventing rotation. In the manner in which its being used here is akin to applying a break. The holding torque of a stepper motor describes the torque that can be applied to the shaft of the motor without rotating it. The torque M of a rotating system is related to its moment of inertia J and its angular acceleration $\bar{\alpha}$ according to equation 3.2. Through testing it was determined that the optimal supplied voltage to the stepper motor driver from the boost converter were 12V.

$$M = J\bar{\alpha} \tag{3.2}$$

The boost converter was used to convert the output voltage of the powerbank to the input voltage of the motor. The output voltage of a boost converter can be approximated using the law of conservation of energy, see Equation 3.3.

$$\begin{aligned} P_1 &= P_2 \\ U_1 I_1 &= U_2 I_2 \end{aligned} \tag{3.3}$$

Using Equation 3.3 where $I_1 = 4.2$A is the amperage coming from the powerbank, $V_1 = 5$V is the voltage from the powerbank and $V_2 = 12$V is the desired voltage, we can see that $I_2 = 1.75$A are available for the stepper motor.Figure 3.2 shows the electrical circuit for the main components of prototype A3.

3.1.2 Controlling the system

The way the Raspberry Pi 3 Model B communicates with the electrical components are so called General Purpose In/Out Pins (GPIO). A GPIO pin can be set to output a signal within a python program with the python library *RPi.GPIO* for the Raspberry Pi 3 Model B. The GPIO pins on the Raspberry Pi 3 Model B can provide an electrical signal of up to 3.3V. In this system the GPIO pins are used to control the stepper motor driver, which in turn controls the stepper motor. The rolling limit switch , when activated, provides a reference point in the rotational plane for the automatic panoramic imaging system. The rolling limit switch is activated by a cam on the base plate which is passed by the limit switch once per rotation.
Figure 3.3 shows a list of the capabilities the GPIO pins on the Raspberry Pi 3 provide. The Pins 9, 12, 13, 16 and 18 are used to control the stepper motor, Pins 3 and 6 are used to observe the limit switch.

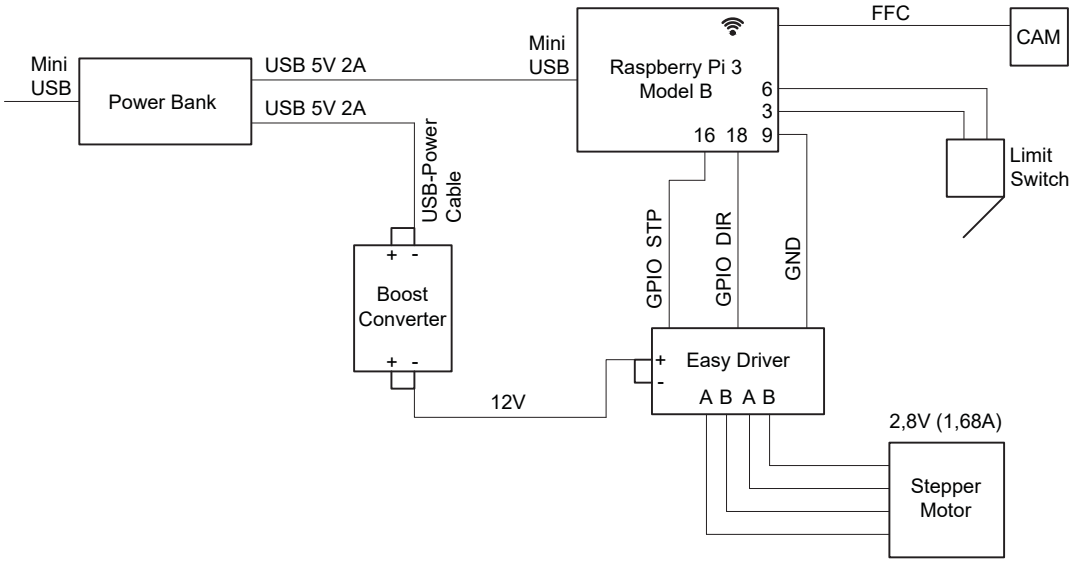


Figure 3.2: Electric circuit for the main components of the prototype A3.

	NAME	NAME	
1	3.3v DC Power	DC Power 5v	2
3	GPIO 2 (SDA1, I²C)	DC Power 5v	4
5	GPIO 3 (SCL1, I²C)	Ground	6
7	GPIO 4 (GPIO_GCLK)	(TXD0) GPIO 14	8
9	Ground	(RXD0) GPIO 15	10
11	GPIO 17 (GPIO_GEN0)	(GPIO_GEN1) GPIO 18	12
13	GPIO 27 (GPIO_GEN2)	Ground	14
15	GPIO 22 (GPIO_GEN3)	(GPIO_GEN4) GPIO 23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO 24	18
19	GPIO 10 (SPI_MOSI)	Ground	20
21	GPIO 9 (SPI_MISO)	(GPIO_GEN6) GPIO 25	22
23	GPIO 11 (SPI_CLK)	(SPI_CE0_N) GPIO 8	24
25	Ground	(SPI_CE1_N) GPIO 7	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO 5	Ground	30
31	GPIO 6	GPIO 12	32
33	GPIO 13	Ground	34
35	GPIO 19	GPIO 16	36
37	GPIO 26	GPIO 20	38
39	Ground	GPIO 21	40

Figure 3.3: GPIO Header Pins and their properties for the Raspberry Pi 3 [24].

3.1.3 Stepper motor driver

The power for the motor, coming from the boost converter, is put on Power In as seen on Figure 3.4. The pins Motor Coil A and B on the stepper driver are connected to the corresponding wires from the stepper motor. In the bottom left corner of the driver as shown in Figure 3.4 is a connector which switches the operation Voltage from 5V to 3V. Without doing this the driver could not be controlled with the Raspberry Pi 3 Model B since the signal provided by the GPIO's has a voltage of 3.3V. Finally the GND connector is connected with the GPIO 9 on the Raspberry Pi because it acts as ground, see Figure 3.3. Step Input is connected with GPIO 16 and Direction Input with GPIO 18. With the two states of the GPIO's, True and False, the motor can now be given a direction impulse and a step impulse with, for example, a python program.

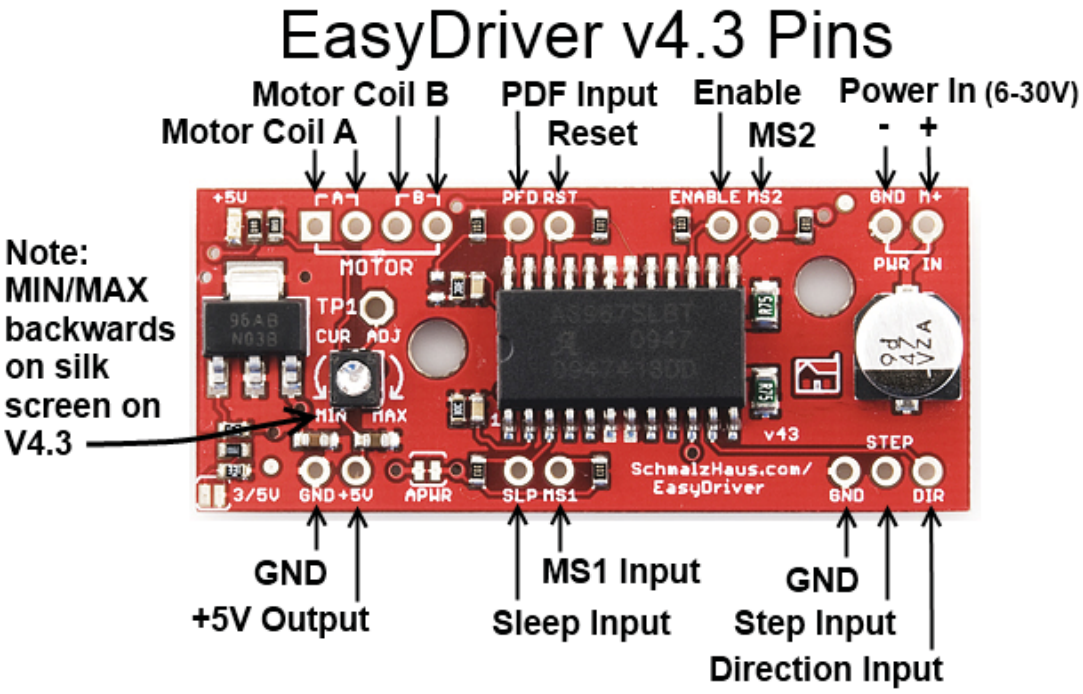
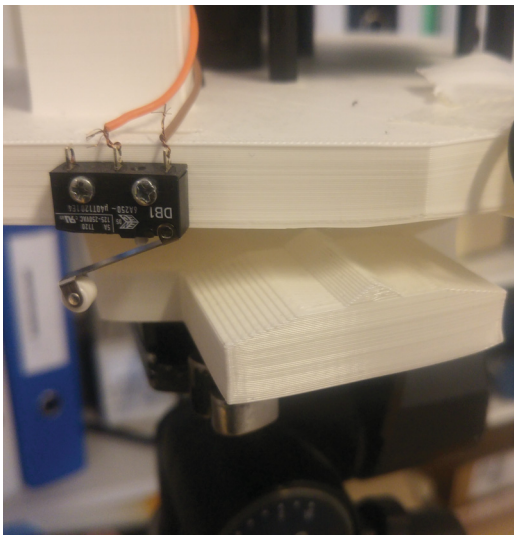


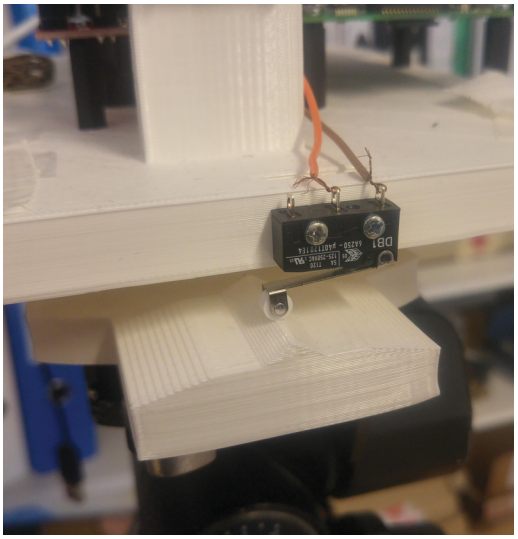
Figure 3.4: EasyDriver [20] stepper motor driver was used to control the stepper motor.

3.1.4 Limit switch

To create a reference or zero point for the rotation of the system a limit switch is used. The limit switch can open or close an electric circuit with a button press, depending on the wiring. For this system the circuit is open on default and closed when the limit switch is activated. This type of limit switch uses a lever with a roll at the end to perform the button press. When the lever is pushed the button is pressed and the electric circuit closed.The logical signal from this circuit is used as parameter in the python code to determine if



(a) Close up of the open rolling lever limit switch.



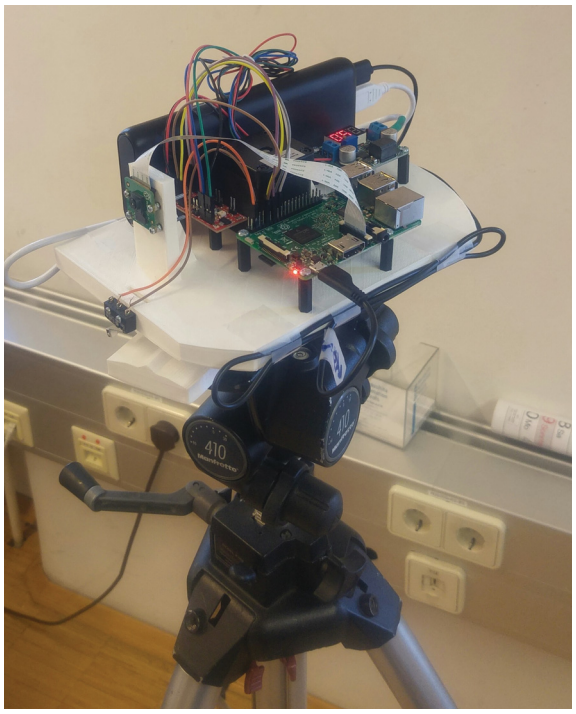
(b) Close up of the closed rolling lever limit switch.

Figure 3.5: Close up of the rolling lever limit switch that is used to create a reference point in the rotational plane.

the switch was pressed. To make sure that the signal from the limit switch is correctly interpreted by the Raspberry Pi 3 Model B it is connected through a pull down resistor circuit.A pull down resistor is a electric circuit that holds the logic signal of the device, e.g. the limit switch, near zero volts, i.e. False, while it is not activated, i.e. the circuit is open. The GPIO pin 3 has a built in function to act as a pull down resistor, no separate electrical circuit is needed. When the rolling lever of the limit switch passes the elevation of the base plate the electric circuit is closed and the logic level increased to 3.3V, i.e. True. This change can be observed by a running python program to determine the move through the reference point.

Figure 3.6 shows the assembled device.

(a) Automated prototype mounted on tripod.



(b) Close up of the automated prototype.

Figure 3.6: Figure 3.6b shows the assembled device. Figure 3.6a shows the device mounted on the tripod.

3.2 Software

The goal of this work is to investigate the feasibility of a high resolution panoramic imaging system as a whole. As a prototype it was decided to implement the investigatory software as a number of independent components, which can be concatinated to demonstrate the full functionality. In this manner the components can be programmed independently and mixed with commercially available elements to implement the prototype. The main components are:

1. image acquisition
2. image processing to account for projective effects
3. mosaicking the individual images to a panorama
4. computation of stereoscopic disparity to determine depth (still to be done)
5. generation of panoramic stereoscopic pairs
6. three dimensional visualization for viewing and total immersion in virtual reality

3.2.1 Image acquisition

The image acquisition is performed by the embedded system, in this case a Raspberry Pi 3 Model B. Python has been selected to programme the image acquisition and local storage; python was chosen because of the high level ob abstraction provided while being platform independent The python code written to capture a variable amount of images follows the pseudo code.

Data: Desired number of images
Result: Images
Move camera to zero position;
while *Rolling switch not activated* **do**
    while *Steps performed is smaller than the desired steps between each image* **do**
        Step;
        Steps performed + 1;
    end
    Capture image;
    Store image;
    Steps performed 0;
end

The number of steps n the stepper motor makes between each image depends on the number of images desired k and the number of steps the system performs to perform a full rotation. The stepper motor used in this system performs 400 steps per full rotation,

this equates to an angle ϕ =0.9° per step. The microstepping mode of the stepper driver module reduces the rotation angle per step to $\frac{1}{8}$ of the regular angle. Therefore the system rotates 0.1125° per step, performing 3200 steps for a full rotation. With Equation 3.4 the number of steps between each image is calculated every time the program is started.

$$n = \frac{3200}{k} \tag{3.4}$$

It is necessary to know the rotational position at which each image was acquired to enable vertical geometric registration of multiple panoramas. The knowledge simplifies the vertical geometric registration by defining a starting solution close to the final solution for the optimization. The limit switch in combination with the specifically designed ground plate deliver a parameter that can be used in the machine code to determine the rotational position of the camera when the image was acquired. From there the number of steps provided to the motor i and the step resolution ϕ make it possible to calculate the rotational angle θ of the camera.

$$\theta = i\,\phi \tag{3.5}$$

3.2.2 Data management

The current implementation supports the acquisition of n=3200 images per 360° rotation, whereby each image can have a resolution shown in Table 2.1. In the first step the images are stored locally on an SD-memory card. Whenever an image is captured it is given an automatically generated file name and stored on the Raspberry Pi 3 Model B.
The file name is generated on the basis of the structure shown in Figure 3.7.

```
<projectName>_<DateTime>_<Set#>_<Layer#>_<Image#>
```

Here *projectName* would be the name or location of the mine shaft, *DateTime* is the date and the time at which the image was captured, *Set#* includes every image captured of one shaft, *Layer#* refers to the vertical position of the camera system and *Image#* is the number of the image of one rotation starting with zero at the reference point of the capturing system. This naming convention permits the unique identification of each image with the corresponding orientation of acquisition. In Listing 3.1 the python code for the file name creation is shown. This structure ensures that every image is assigned to a clear position in the representation of the shaft. Once the data acquisition is over, the data can be moved to the main storage/processing unit via FTP.

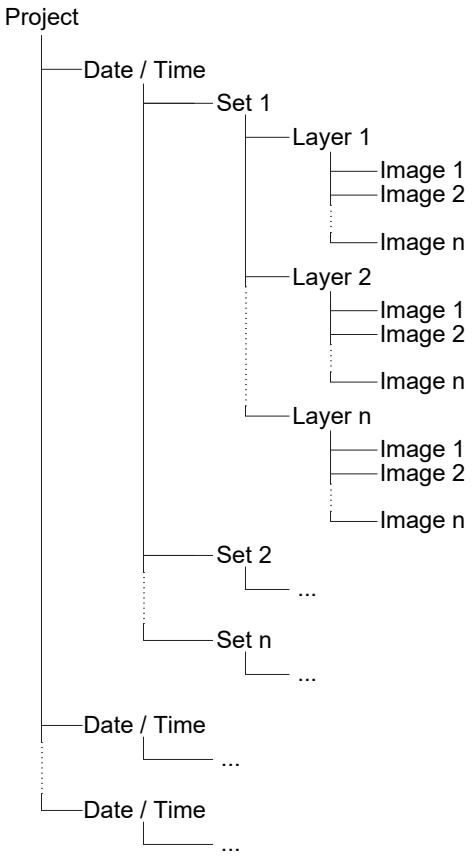


Figure 3.7: The hierarchical data structure ensures that every image can be associated with the rotational orientation of acquisition. The increasing level of abstraction ensures a unique file name.

```
1 def dataManagement(layerNum,newSet):
2     ''' This function is designed to check if directories exist and
       if not create
3     them. It returns the directory in which the pictures of the
      current session are
4     to be saved as string.'''
5     # layerNum is the number of the current layer
6     # newSet is a boolean that indicates if the layer needs to be
7     # created in a new set folder (i.e. in the previous run the
      last layer of a
8     # set was created)
9
10    # Get current date
11    now = datetime.datetime.now()
12    # Only use Day Month and Year
13    today = now.strftime("%d%m%Y")
14
15    # Get list of directories that exist in the current working
      directory
```

```
16      directories = next(os.walk(os.getcwd(),topdown=True))[1]
17      # create pathname for the current date
18      pathToday = os.path.join(os.getcwd(),today)
19
20      # Check if folder with current date exists
21      if today in directories:
22          print("Todays directory already exists")
23          # if newSet is True a new folder SETN will be created
24          if newSet:
25              # create new Set folder and layer n folder
26              curSetPath = setManagement(pathToday,newSet)
27              curLayPath = layerManagement(layerNum,curSetPath)
28              return curLayPath
29          # if newSet is False no new SET folder will be created
        insted a
30          # new layer folder will be created in the current SET
        folder
31          else:
32              # create layer folder
33              curSetPath = setManagement(pathToday,newSet)
34              curLayPath = layerManagement(layerNum,curSetPath)
35              #print("This should happen if newSet False")
36              return curLayPath
37      else:
38          # if current date does not exist as folder, create folder
39          os.makedirs(pathToday)
40          print("New directory {} created".format(pathToday))
41          curSetPath = setManagement(pathToday,newSet)
42          curLayPath = layerManagement(layerNum,curSetPath)
43          return curLayPath
```

Listing 3.1: Python code used to manage the file naming.

Chapter 4

Image Processing

This chapter gives a brief introduction to some of the concepts used in image processing to create panoramic images. The concepts are then applied to the images produced by the automatic panoramic imaging system described in Chapter 3. The device described in Chapter 3 delivers images in the form seen in Figure 4.1.

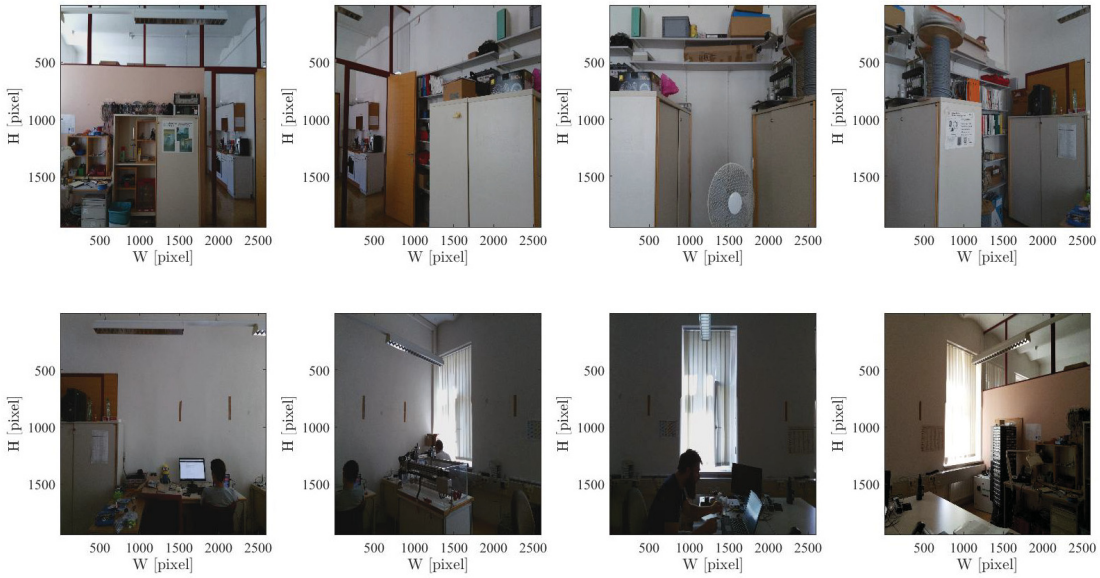


Figure 4.1: Eight images acquired at one vertical position. One panoramic ring is constructed from these images. The images were acquired with the prototype discussed in Chapter 3.

Within one layer the images from one closed 360° view overlap on the right and left ends. The overlap between consecutive images can be set through the number of steps between each image. This is done so that the panorama stitching algorithm, which is discussed in Section 5, delivers the best possible results. Additionally the images overlap from layer to layer to ensure good registration of the individual panoramic rings to each other. The spirit level integrated in the Manfrotto tripod [18] also guarantees that there is

no tilt in the images.

4.1 Modelling

The aim of this section is to provide the mathematical framework to link projective geometry and individual images to a panoramic view.

4.1.1 Normalized device coordinates

To be able to work with any image at any resolution normalized device coordinates are used [3]. Mapping the pixel coordinates $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y})$ to normalized device coordinates $\boldsymbol{x} = (x, y)$ is done with:

$$x = \frac{2\bar{x} - W}{S} \tag{4.1}$$

and

$$y = \frac{2\bar{y} - H}{S} \tag{4.2}$$

where $S = max(W, H)$. W is the Width and H is the height of the image [3].

4.1.2 3D Transformation

Mapping a point from 3D coordinates $\boldsymbol{p} = (X, Y, Z)$ to 2D coordinates $\boldsymbol{x} = (x, y, 1)$ onto an image plane at a distance f, called focal length, along the z-axis through a pinhole in the camera center is called central projection,

$$\begin{aligned} x &= f\frac{X}{Z}, \\ y &= f\frac{Y}{Z}. \end{aligned} \tag{4.3}$$

The field of view θ and the focal length f have the following relationship:

$$f^{-1} = \tan\frac{\theta}{2} \tag{4.4}$$

A perspective projection can then be written as

$$\tilde{x} \sim \left[\begin{array}{c|c} \mathsf{K} & 0 \\ \hline 0^T & 1 \end{array}\right] \boldsymbol{p} = \mathsf{P}\boldsymbol{p} \tag{4.5}$$

where the homogeneous point vector $\boldsymbol{p} = (X, Y, Z, 1)^T$ is mapped to a homogeneous screen vector $\tilde{x} = (x, y, 1, d)$. The matrix $\mathsf{K} = diag(f, f, 1)$ is called intrinsic camera cal-

ibration matrix and P is the projection matrix. By using this notation, the inverse screen depth information d is kept and can be used to map points between images of a 3D scene [25].

4.1.3 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 E_0,

$$\boldsymbol{x}_0 = \begin{bmatrix} \mathsf{R}_0 & \boldsymbol{t_0} \\ 0^T & 1 \end{bmatrix} \boldsymbol{p} = \mathsf{E}_0 \boldsymbol{p} \tag{4.6}$$

and a perspective projection P_0,

$$\tilde{\boldsymbol{x}}_0 \sim \mathsf{P}_0 \mathsf{E}_0 \boldsymbol{p}. \tag{4.7}$$

If the value for d_0 is known, it is possible to back project the image point $\tilde{\boldsymbol{x_0}}$ to a 3D point $\boldsymbol{p}$,

$$\boldsymbol{p} \sim {\mathsf{E}_0}^{-1} {\mathsf{P}_0}^{-1} \tilde{\boldsymbol{x}}_0 \tag{4.8}$$

and project it into another image

$$\tilde{\boldsymbol{x}}_1 = \mathsf{P}_1 \mathsf{E}_1 \boldsymbol{p} = \mathsf{P}_1 \mathsf{E}_1 \mathsf{E}_0^{-1} \mathsf{P}_0^{-1} \tilde{\boldsymbol{x}}_0 = \mathsf{M}_{10} \tilde{\boldsymbol{x}}_0. \tag{4.9}$$

For a planar scene with $d_0 = 0$, the mapping is reduced to

$$\tilde{\boldsymbol{x}}_1 = \mathsf{H}_{10} \tilde{\boldsymbol{x}}_0 \tag{4.10}$$

where H_{10} is a 3x3 homography matrix and $\tilde{\boldsymbol{x}}_0$, $\tilde{\boldsymbol{x}}_1$ are 2D homogeneous coordinates [25].

4.1.4 Cylindrical and Spherical Coordinates

Instead of calculating the homography, it is also possible to warp the images into cylindrical coordinates and align them by translation. For a camera in the starting position, where $\mathsf{R} = \mathsf{I}$, I being the identity matrix, an (x,y) pixel corresponds to the (x,y,f) 3D ray. A point on a cylindrical surface can be parametrized by the angle θ and the height h.

$$(\sin\theta, h, \cos\theta) \propto (x,y,f) \tag{4.11}$$

The mapped or warped coordinates can then be calculated with

$$x' = s\theta = s\tan^{-1}\frac{x}{f}, \tag{4.12}$$

$$y' = sh = s\frac{y}{\sqrt{x^2+f^2}} \tag{4.13}$$

If the camera is level and rotating around its vertical axis the warped images are related by pure horizontal translation.

Figure 4.2: Cylinder Projection of an image taken with the system and warped using Equations 4.12 and 4.13.

```
[W,H] = size(image(:,:,1));

ydim = W;
xdim = H;

xc = xdim/2;
yc = ydim/2;

f = 1000000;
% Because the focus is near infinity for the camera sensor we get the
% normal image for very big f's

for y=1:ydim
    for x=1:xdim
        theta = (x - xc)/f;
        h = (y - yc)/f;
        xcap = sin(theta);
        ycap = h;
        zcap = cos(theta);
        xn = xcap / zcap;
```

```
21          yn = ycap / zcap;
22          r = xn^2 + yn^2;
23
24          xd = xn;% * (1 + k1 * r + k2 * r^2);
25          yd = yn;% * (1 + k1 * r + k2 * r^2);
26
27          ximg = floor(f * xd + xc);
28          yimg = floor(f * yd + yc);
29
30          if (ximg > 0 && ximg <= xdim && yimg > 0 && yimg <= ydim)
31              out(y, x, :) = [image(yimg, ximg, 1) image(yimg, ximg,
      2) image(yimg, ximg, 3)];
32          end
33
34      end
35 end
```

Listing 4.1: MATLAB code snippet used to perform the cylindrical projection in Figure 4.2 based on the mathematical framework presented in Chapter 4.

4.2 Lens

The camera sensor used in this project is the Raspberry Pi camera module v2.0, which uses a Sony IMX 219 PQ [12] sensor. This sensor has a focal length of $f = 3.04$mm and a viewing angle $\alpha = 62.2°$. The maximum resolution per frame is $3280 by 2464$ pixels, this equates to a pixel size of $1.12 by 1.12 \mu$m. Focus near infinity. To model the radial distortions, barrel (away from the image center) and pincushion (towards the image center) a low order polynomial can be used, e.g.:

$$\begin{aligned} x' &= x(1 + \kappa_1 r^2 + \kappa_2 r^4), \\ y' &= y(1 + \kappa_1 r^2 + \kappa_2 r^4). \end{aligned} \tag{4.14}$$

It is believed that the firmware of the Raspberry Pi accounts for these effects and applies the corrections on the raw image before viewing it [26].

4.3 Registration

The alignment of images through feature registration has many possible solutions [4][27], one approach is normalized cross correlation. Using a subregion of an image as template, the cross correlation between this template and a region in the second image is computed.

In this system the rotation of the camera is known and the search space can be confined to a region where the template must lie. To make the cross correlation insensitive to changes in image lightning between images, it is normalized:

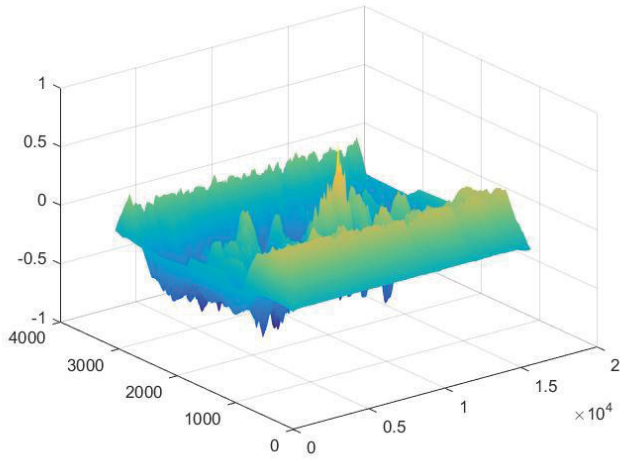
$$C_{ij} = \sum_{i=1}^{p} \sum_{j=1}^{p} \frac{(a_{ij} - \bar{a})(b_{ij} - \bar{b})}{p^2 \sigma(a)\sigma(b)} \tag{4.15}$$

where C_{ij} is the correlation coefficient , p is equal to the size of the template, $\bar{a}$ and $\bar{b}$ are equal to the mean of the region and σ denoting the standard deviation. Using this measure of similarity between feature points in each image the template from Image I_1, can be aligned within the selected subregion of Image I_2. Once the template is aligned the rest of the image is added.

Figure 4.3 shows the process of normalized cross correlation applied to two panoramic images from different vertical positions. Each panoramic image was created from eight images acquired with the system described in Chapter 3.

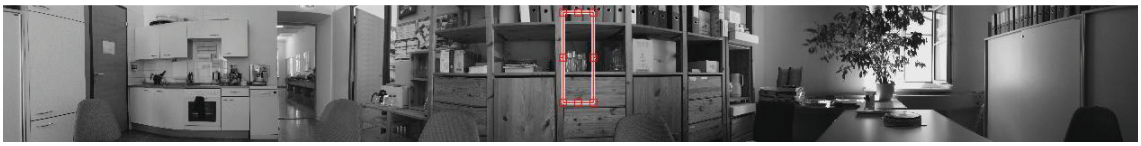
The MATLAB® function *normxcorr*2 was used to perform the computations.

(a)



(b)

(c)



(d)

Figure 4.3: Figure 4.3a shows the template taken from the panorama in Figure 4.3c to determine the offset between the panoramas shown in Figure 4.3d and Figure 4.3c. Figure 4.3b represents the correlation between the template and the panorama in Figure 4.3d, the peak in height corresponds to the x and y offset between the images. The outline in Figure 4.3d represents the location where the template has the highest correlation with the panorama.

Chapter 5

Image stitching

This chapter is about the process of creating panoramic images for an automated panoramic imaging system for the visual inspection of vertical deep mine shafts.
Panoramic imaging is important for the creation of an automatic system for the visual inspection of vertical deep mine shafts because it makes it possible to create images with horizontal fields of view up to 360° and infinite height.This is achieved by combining multiple images from different view points of the same scene to a single image by aligning them along features of the scene that are visible in multiple images. Because of the different camera parameters, i.e. position, from where the images where captured the same feature of a scene can occur at different image coordinates in different images.
The challenge is therefore to recognize corresponding features in different images and determine the shift between images needed to align them.Automatic recognition of features in images is a complex field of study in computer vision that has produced many different approaches [27][4][1] for different requirements.A short introduction to an automated approach to registration, namely normalized cross correlation, is given in Section 4.3. Section 4.3 also provides an example for registration through normalized cross correlation in the context of an automated system for panoramic imaging for the visual inspection of deep vertical mine shafts as described in Chapter 2.

5.1 Panoramic imaging

One way to create panoramic images is to use overlapping aligned images and performing different tasks, such as gain compensation and multiband blending to create a panoramic image. AutoStitch by Brown and Lowe [16] is a software that produces panoramic images as shown in Figure 5.1 from overlapping images. Another way is to produce the panoramic image by combining stripes of a small numbers of pixel columns from each image. The shift between images can be calculated with the algorithm presented in Section 4.3. With the shift between consecutive images the number of pixel rows that con-

Figure 5.1: Panoramic image created by combining eight images with the software AutoStitch [4].

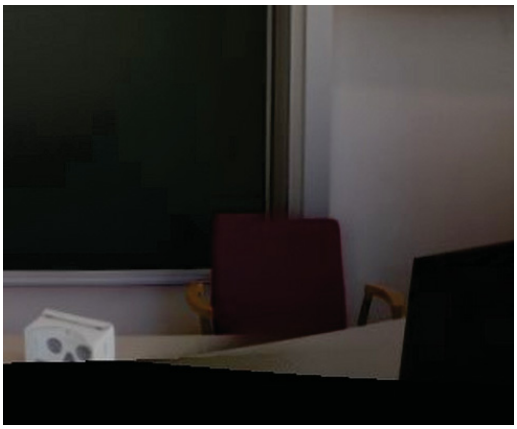
tribute to the entire panorama can be determined. This can be referred to as slit-camera. This approach has the advantage of reducing distortions in the image produced by the camera lens. Using only the center column of pixels per image from 3200 images captured with the system described in Chapter 3 produces a panoramic image as shown in Figure 5.2d. Listing 5.1 shows the code snippet that was written to produce Figure 5.2d from the individual images.

```
1
2 % To determine the number of pixels calculate the shift between two
      images, shift between the other images is the same
3 n = 1;
4 center = 320;
5 m = 480;
6
7 centerColumn = zeros(m,n,3);
8
9 for k=3199:-1:0
10
11     fileName = sprintf('stereoTest640/picture%d.jpg',k);
12
13     image = imread(fileName);
14
15     centerColumn(:,1,:) = image(:,center,:);
16
17     pan = [pan,centerColumn];
18
19 end
20
21 pan = uint8(pan);
```

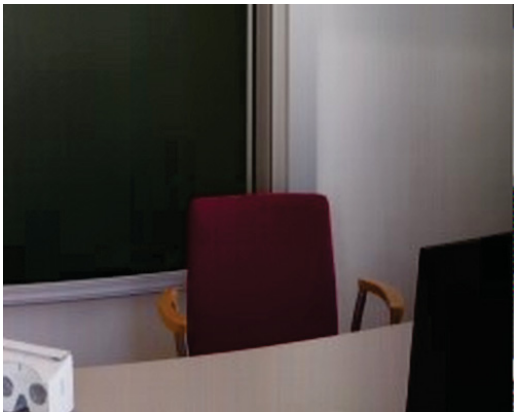
Listing 5.1: Matlab code used to produce Figure 5.2d

5.1.1 Comparison AutoStitch and Slit Camera

The aim of this section is to showcase the differences in the panoramic images created with the software AutoStitch [16], see Figure 5.2c, and the algorithm presented in Section 5.1, see Figure 5.2d. Figure 5.2a shows a detail of the panorama in Figure 5.2c. The chair seen in the detail lies at the intersection of two overlapping images. The stitching algorithm does not properly align the borders. Figure 5.2b provides the same region seen in Figure 5.2a from the panorama created with the slit camera approach, Figure 5.2d. NBy combining the pixel columns from the center of each image, the algorithm can provide a coninuous representation of the scene with a field of view of 360°.



(a) Detail AutoStitch



(b) Detail Slit Camera

(c) Panorama AutoStitch

(d) Panorama Slit Camera

Figure 5.2: Figure 5.2a and Figure 5.2b are details of the panoramas in Figure 5.2c created with AutoStitch [16] from eight images and Figure 5.2d created with the algorithm presented in Section 5.1. The differences in the two approaches can be observed.

5.2 Panoramic depth imaging

To create the perception of depth when viewing the representation of the shaft surface by the inspection personnel, panoramic stereo imaging (omnistereo) is used. Calculating the actual depth values lies outside the scope of this thesis, it can however be achieved using the approaches presented by Peer and Solina [28]. Two panoramas, one representing the left eye the other representing the right eye view are used to create an omnistereo panorama. Section 5.1 explains how the panoramic images are mosaicked. To set up the left and right eye view a special projection called the circular projection is used.

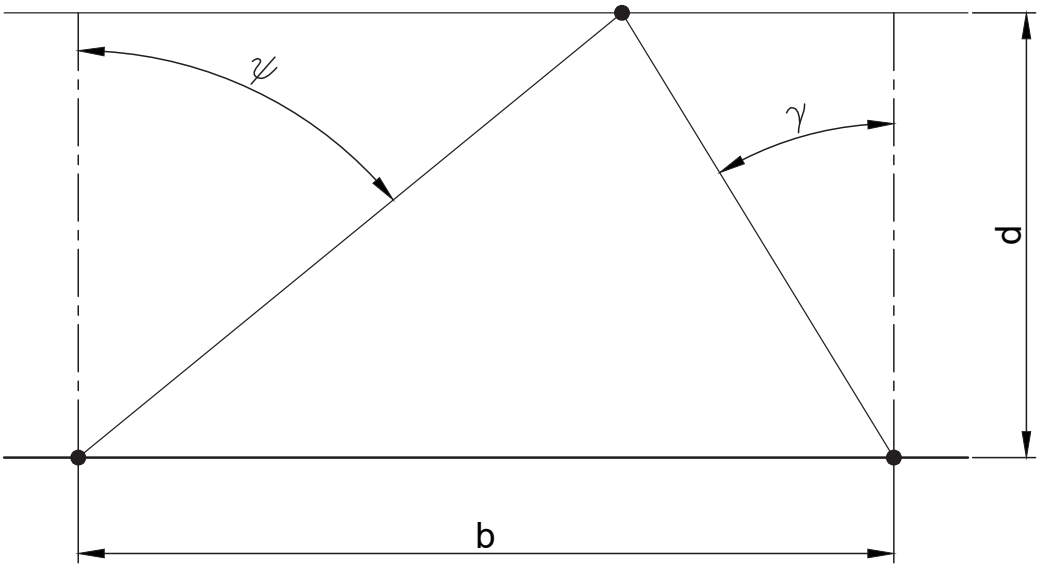


Figure 5.3: With a known distance b between viewpoints, i.e. baseline, and the angles ψ and γ the distance of a point from the baseline d can be calculated.

5.2.1 Stereoscopy

In many cases panoramic images are created with one camera from one viewpoint. A stereo pair is created by two images with viewpoints corresponding to the position of the eyes. The brain interprets the angular difference between each point as depth. From this set-up the sensation of depth can only be created in the direction perpendicular to the baseline[1] see Figure 5.4.

[1]The baseline is a line connecting the viewpoints of the camera.

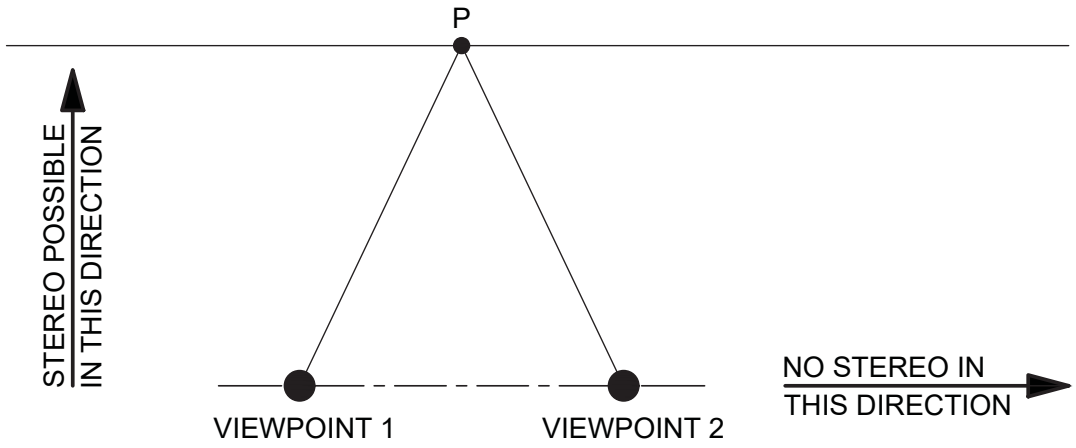


Figure 5.4: A stereo pair cannot give the perception of depth in every direction [29].

5.2.2 Circular Projection

Normal images are usually created with a central projection, Figure 5.5. A central projection is a special case of perspective projection where all projecting lines pass through one single point, the viewpoint. To create the left and right eye panorama a special projection called circular projection is used. In a circular projection the projecting lines pass through multiple view points. These view points lie on a with the path of the rotating camera concentric circle, the viewing circle, Figure 5.5(b-c). The stereo perception is created by the resulting view points for the left, Figure 5.5 (b) and right eye panorama, Figure 5.5(c) on the viewing circle. Since the left eye panorama uses tangent lines on the clockwise direction of the viewing circle and the right eye panorama uses tangent lines on the counterclockwise direction, every point on the viewing circle represents a view point and a viewing direction. This means that stereo perception is possible in every direction.

5.2.3 Omnistereo

Figure 5.6 shows the process of creating stereo panoramas from a single rotating camera. From every image three stripes are kept and combined with those from the following picture. The center stripe represents a normal panorama as described in Section 5.1. Two stripes, where each has an offset of v pixels from the center stripe, create the left and right eye view panorama respectively. The stripe on the right side of the center is used to mosaic the left eye panorama. The stripe on the left side of the center is used to create the right eye panorama. The Listing 5.2 shows the MATLAB® code written to create the left eye view and right eye view panorama from 3200 images caputred in one rotation with the system described in Chapter 3, Figure 5.7 shows the results.

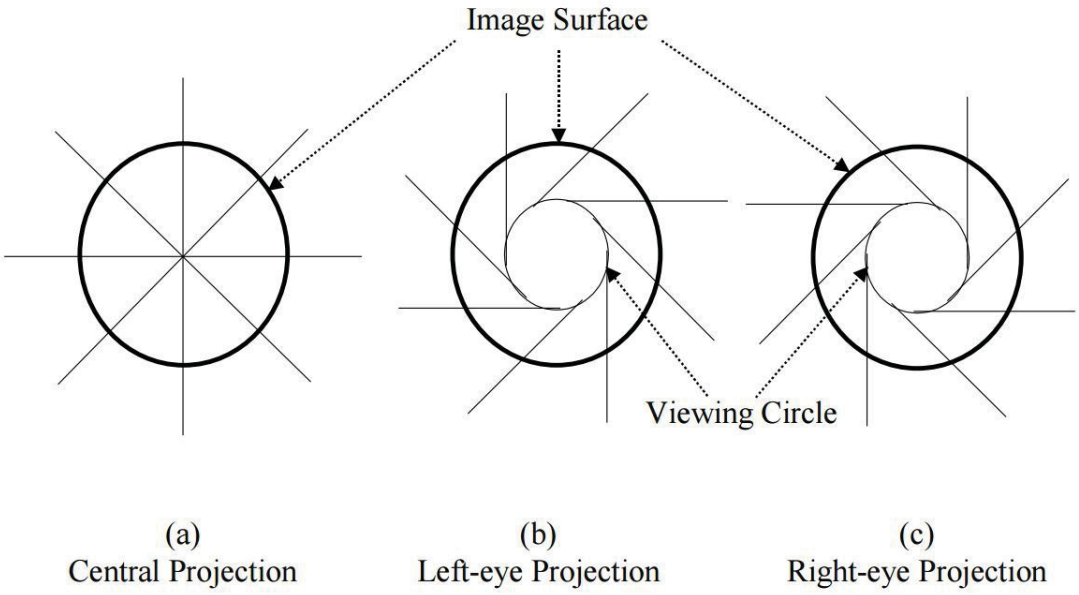


Figure 5.5: (a) Central projection, (b) and (c) circular projection according to [2].

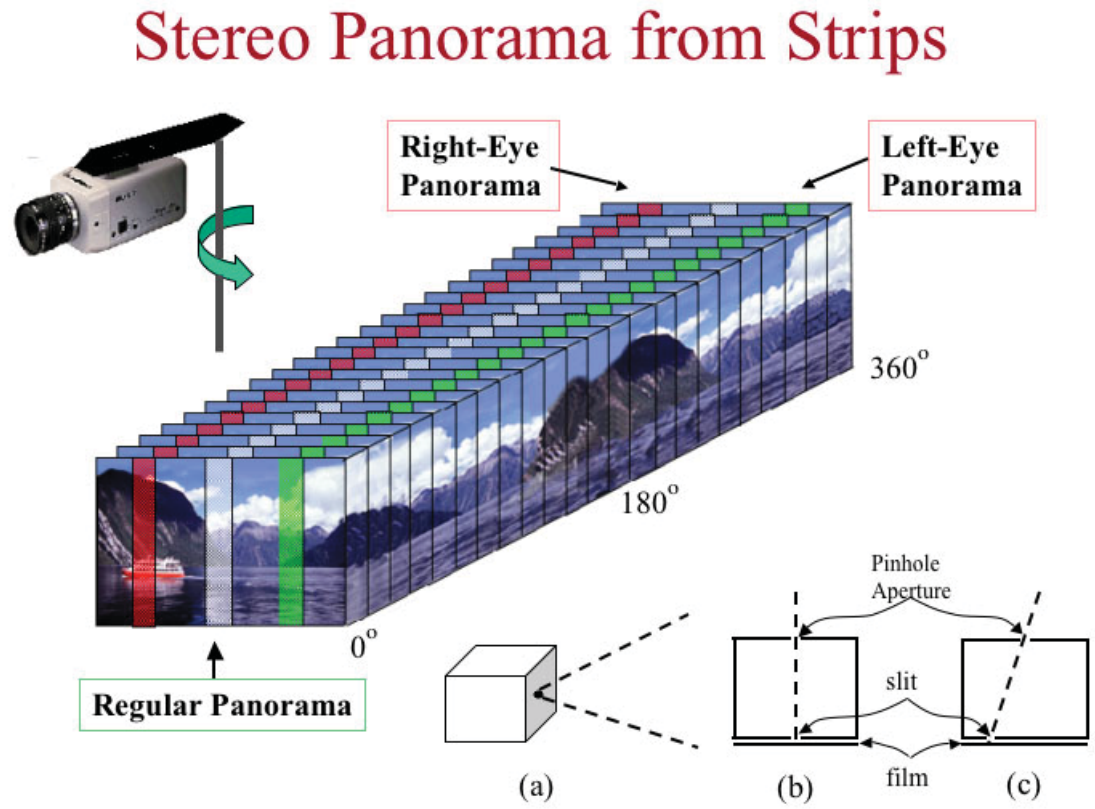


Figure 5.6: Creation of the left eye view panorama and the right eye view panorama with a single rotating camera according to S.Peleg [2].

```
n = 1;
d = 10;
center = 320;
m = 480;

centerColumn = zeros(m,n,3);
rightView = zeros(m,n,3);
leftView = zeros(m,n,3);

for k=3199:-1:0
    fileName = sprintf('stereoTest640/picture%d.jpg',k);
    image = imread(fileName);

    rightView(:,1,:) = image(:,center-d,:);

    centerColumn(:,1,:) = image(:,center,:);

    leftView(:,1:n,:) = image(:,center+d,:);


    pan = [pan,centerColumn];

    leftPan = [leftPan, leftView];
    rightPan = [rightPan, rightView];
end
```

Listing 5.2: MATLAB code snippet that was written to create a normal panorama the left eye view panorama and the right eye view panorama from a single rotation of the prototype described in Chapter 3

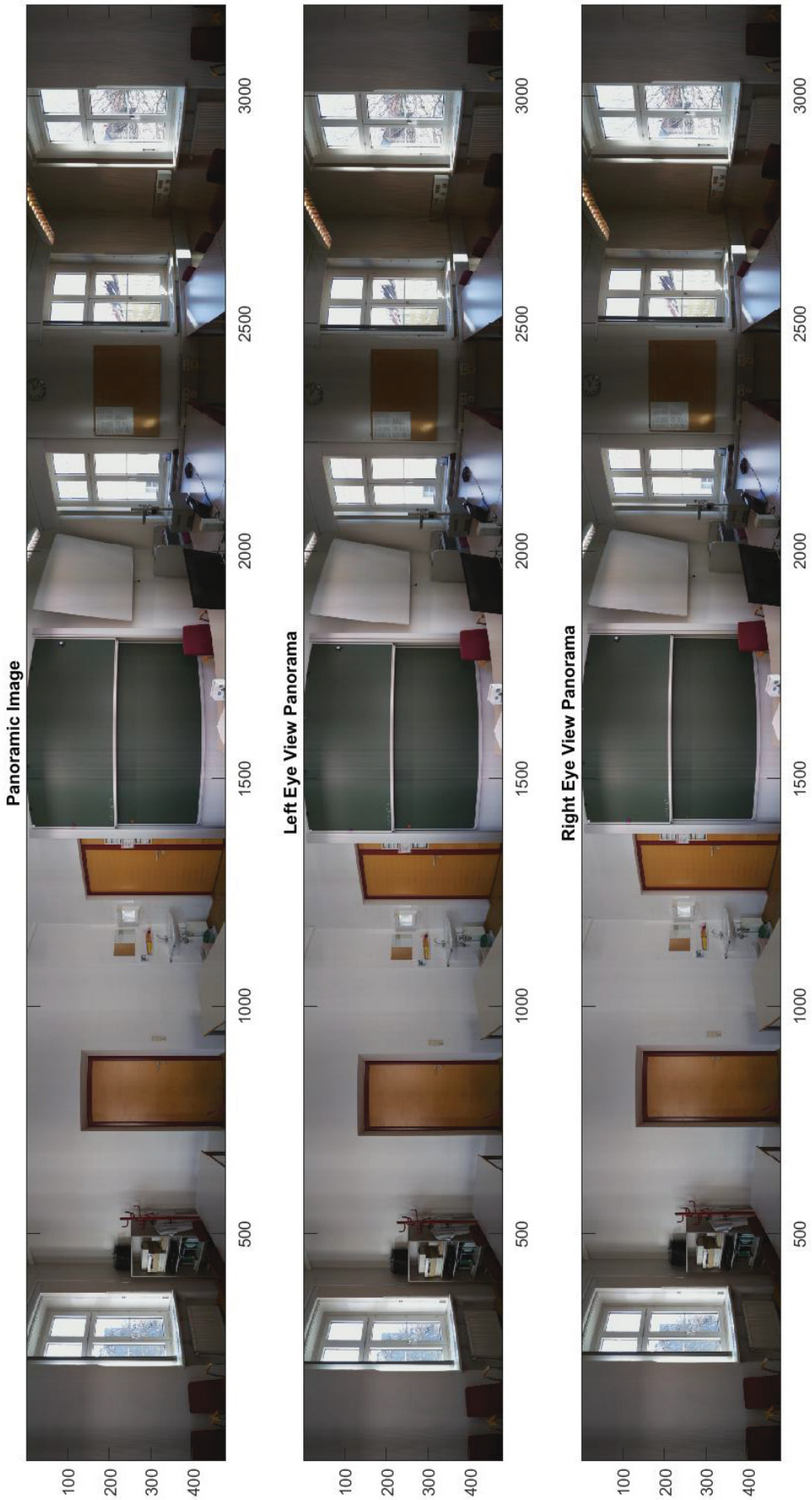


Figure 5.7: Panoramic image created from 3200 separate images. Left eye view and right eye view were created with the algorithm presented in Section 5.2.3.

Chapter 6

Visualization

This chapter discusses different possibilities to present the stereo pair of panoramic images, Section 5.2.3, created from images acquired with the system discussed in Chapter 3 in a way that creates a depth perception in the viewers brain.
The depth perception is necessary for the inspection personnel to observe changes in the surface of the shaft. There are several ways to display a stereo panoramic pair to create the perception of depth in the viewers brain:

1. Anaglyph Glasses with suitably prepared images
2. Head Mounted Display

Other options not covered in this thesis include: LCD Shutter Glasses [29] and FreeView [29].

6.1 Anaglyph

By applying specific color filters to the right and left eye view panorama, the perception of depth is created in the viewers brain when viewing the image with special, to the filter corresponding, anaglyph glasses.

The left and right side view from Figure 5.7 can be used to produce an anaglyph with the MATLAB® function *StereoAnaglyph*, Figure 6.2 shows the result. In an anaglyph the three main color layers are separated to the left and right eye view. In this case the left view represents the red layer, and the right view blue and green, i.e. cyan. These views are then stacked over each other to create the anaglyphic material [30]. When this image is viewed through color coded anaglyph glasses, the brain fuses them into a perception of a three dimensional scene [31]. Figure 6.1 shows the process to create a stereo anaglyph for a pair of stereo panoramas.

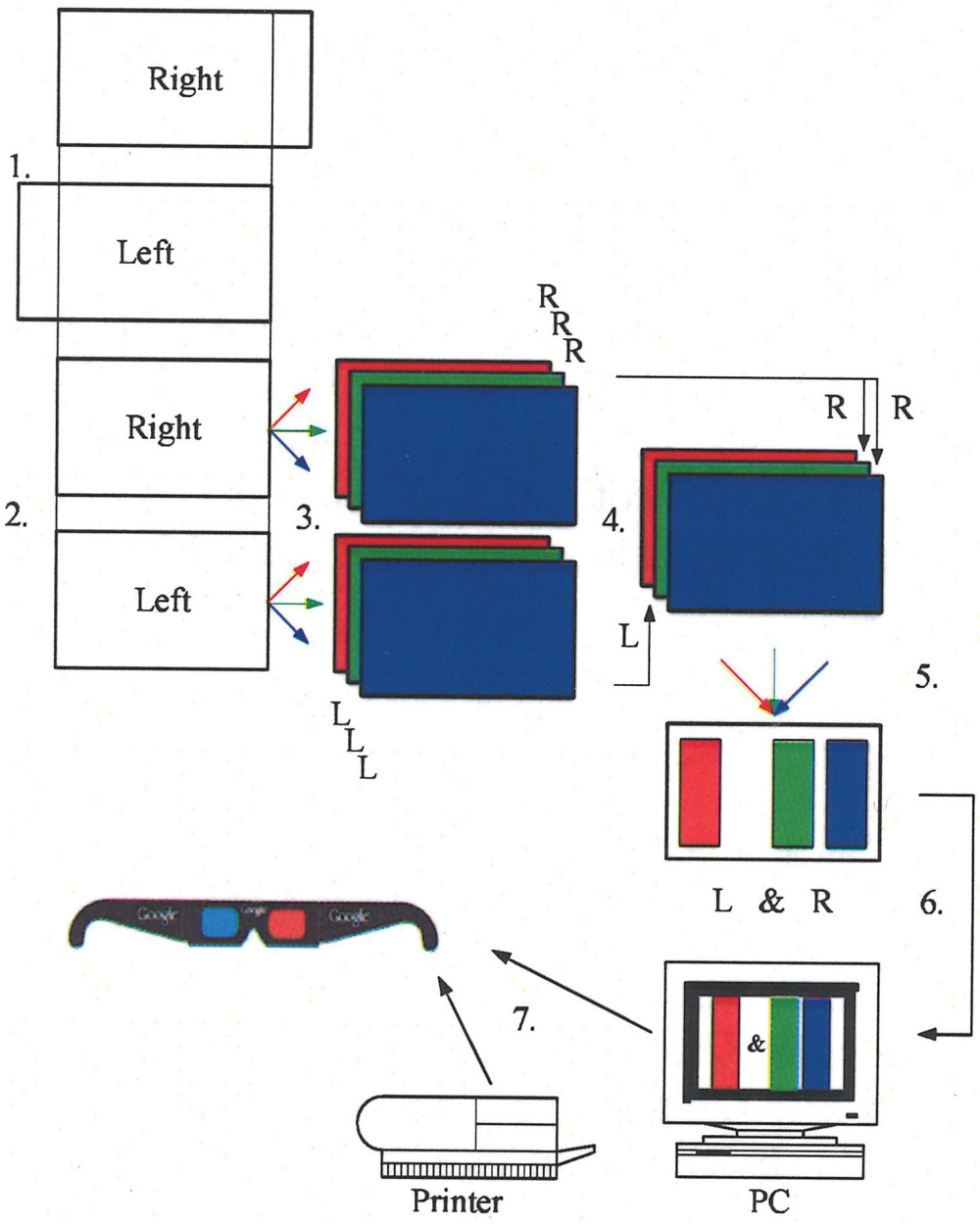


Figure 6.1: A stereo anaglyph is produced in five steps: 1. Create a stereo pair of panoramic images using circular projection, 2. Cutting the panoramas according to the shift between them, 3. Separation of the three color bands, 4. Combination of the three color bands into one stereo panorama, 5. View the stereo panorama with anaglyph glasses [29].

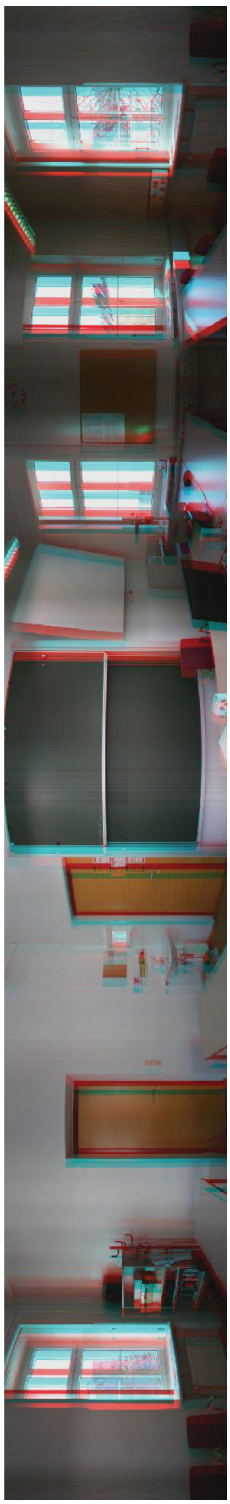


Figure 6.2: Anaglyph created from images taken with prototype A3 and the MATLAB® function *stereoAnaglyph*.

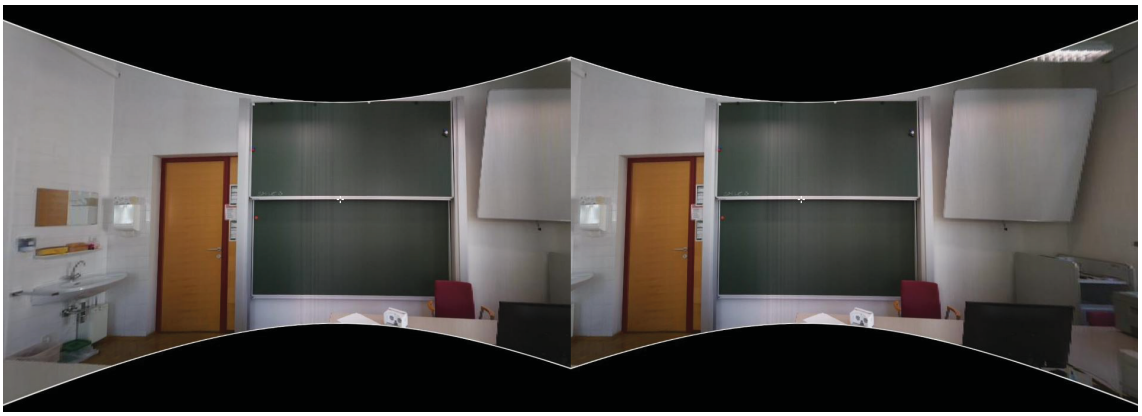
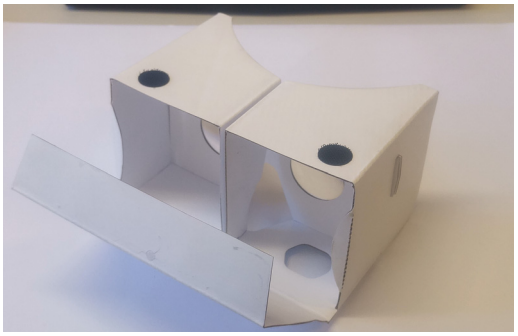


Figure 6.3: The pair of stereo panoramas created in Section 5.2.3 prepared to be viewed with a head mounted display.

6.2 Head Mounted Display

The stereo panorama created in Section 5.2.3 are compatible to be viewed with a Head Mounted Display, HMD. A Head Mounted Display uses two lenses that are placed in front of a stereo pair of separate images to produce a single three dimensional image.By using a smart-phone to display these images it is possible to view a pair of stereo panoramas as one three dimensional panorama. The panorama can be explored by intuitively moving the head while holding the display. The acceleration sensors and the gyroscope of the phone provide with the necessary data to calculate the viewing direction. One commercially available software to produce a virtual reality tour is krpano [5]. Figure 6.3 shows the pair of stereo panoramas created by the process explained in Section 5.2.3 prepared for the HMD shown in Figure 6.4b.

(a) VR Tour



(b) Google Cardboard

Figure 6.4: The VR representation created with krPano [5] can be viewed on a smart phone screen. The phone is then mounted in the HMD Google cardboard. The lenses in the HMD distort the images in a way that enables the brain to perceive depth.

Chapter 7

Data storage

7.1 Reduced Resolution Set

The panoramic images created in the previous section are stored in a reduced resolution format [27]. MATLAB® offers the function *rsetwrite*. This function splits the image into equally sized images. Each of these sub-images is then resampled at different resolution levels depending on the layer number, see Figure 7.1. When viewing this file the resolution of the displayed image increases with the zoom level [32]. This format makes it possible to view images, that would otherwise be to big to fit into memory.

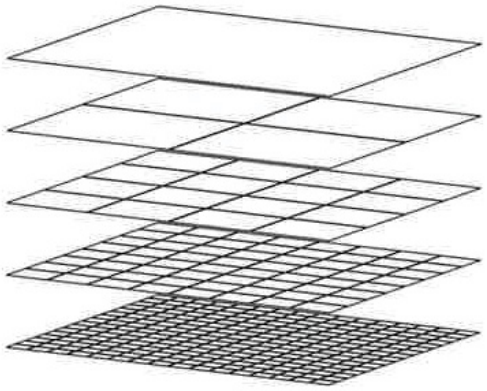


Figure 7.1: Princinpal behind the reduced resolution according to Badash, O'Leary et al. [27].

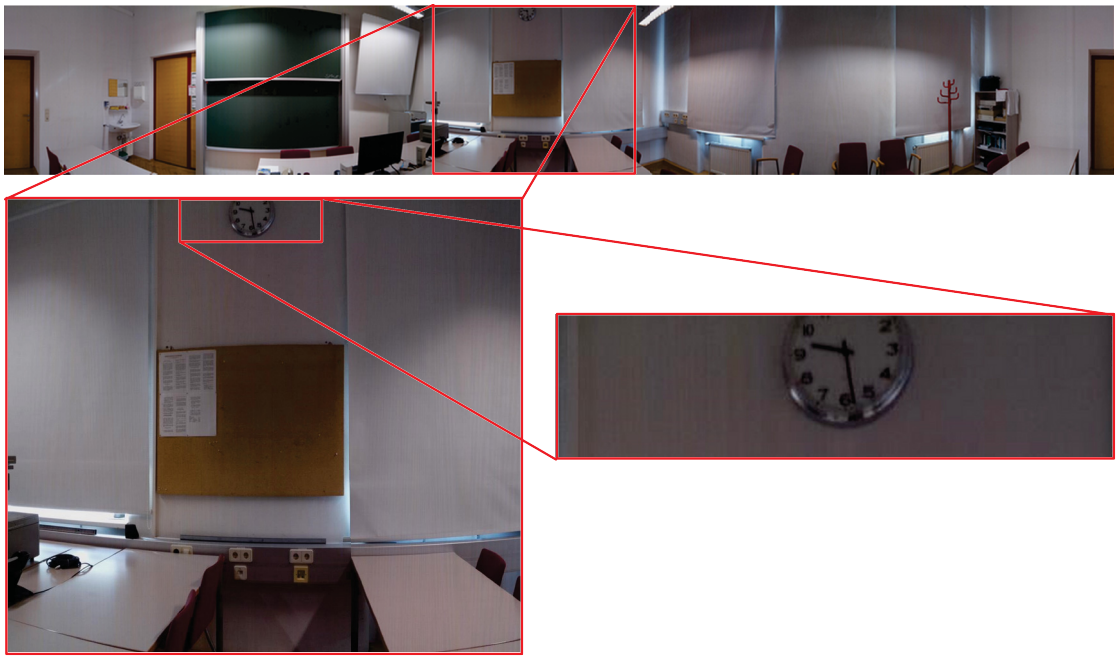


Figure 7.2: An example of an Reduced Resolution Set displaying a high resolution panoramic image created with the system described in Chapter 3 and the algorithm discussed in Section 5.2.3.

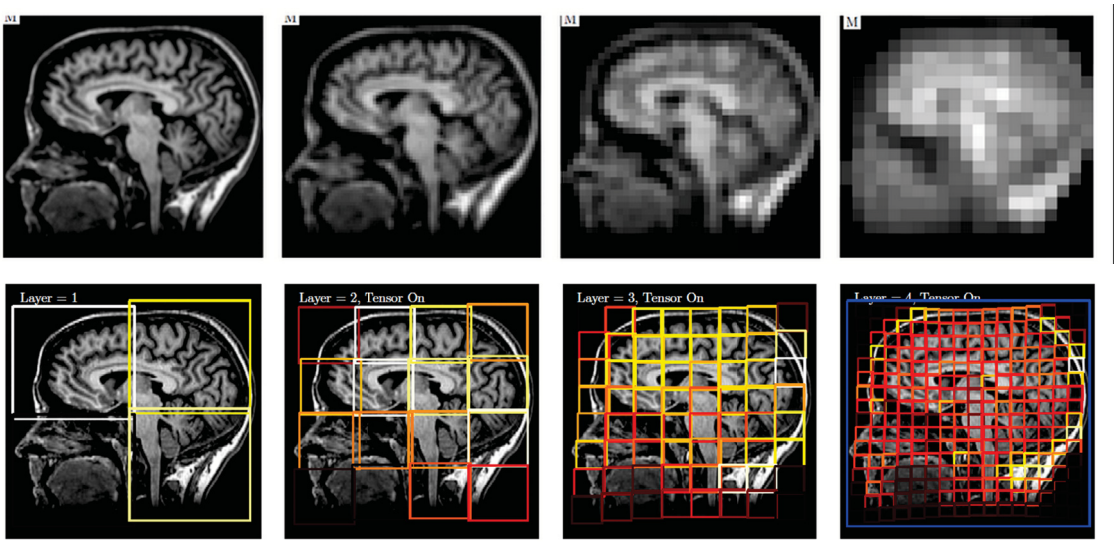


Figure 7.3: Badash, O'Leary et al. [27] used this technique for non-rigid registration.

7.2 Geographic information system

A Geographic Information System makes it possible to connect data with a geographic position [7]. Researchers have used the possibility of overlaying different data set to determine influences and correlations between events and geographic location [6]. For this study the VR reality tours created in section 6.2 as well as other potentially available data, such as geographic depth data, should be viewable by selecting a location on the map, i.e. the location of the mine. One commercial Solution that lets users create custom maps is ArcGIS [33], this program has ample features that surpass the scope of this thesis, hence we will focus on linking a location with the VR Tour. After creating a custom map, a location can be added by searching for street names or coordinates. This location can then be linked to a website URL. For testing purposes a local server provided by krpano is used to host the VR Tour. After starting the server and linking the location to its ip address in the local network it is possible to access the VR Tour from any device connected to the local network. Using a VPN client to connect to this network is a simple solution to make it possible to connect from any other network and view the data.

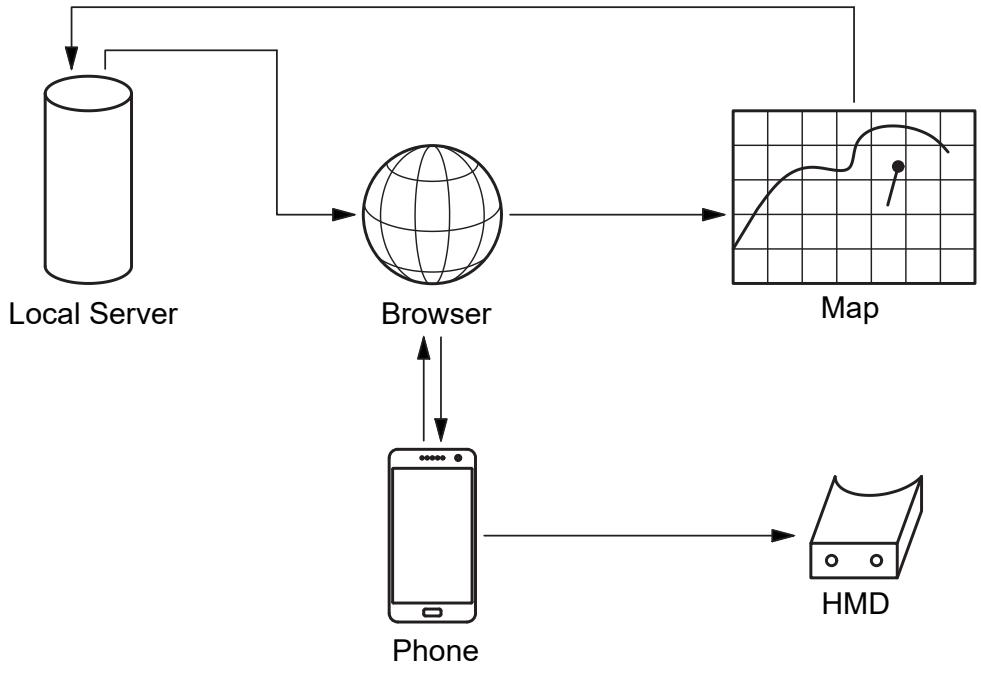


Figure 7.4: Presenting the VR Tour so it can be viewed from any device.

Chapter 8

Conclusion

This thesis investigates concepts and possible implementations for the automatic panoramic depth imaging for the automatic visual inspection of vertical deep mine shafts. This work is akin to a feasibility study; whereby each element in the complete chain, from images to virtual reality, are verified with respect to being feasible.

The complete work flow composed of:

1. Image capturing,
2. Image processing,
3. Panoramic image mosaicking,
4. Stereographic panoramic image computation and
5. Three dimensional visualisation

has been implemented and tested. This method provides the partners of the KIC-Raw Materials Project titled Maintained Mining Machine with the means to develop a system to perform visual inspections of vertical deep mine shafts remotely.

All steps necessary for the development of a suitable capturing device as well as the mathematical background for the computation of omnistereo panoramic images have been verified with respect to being feasible.

It has been shown that all required image data for a three dimensional representation of a large contiguous surface can be provided by a capturing system with a single rotating camera or with eight stationary cameras.

8.1 Future Work

This work has verified the feasability of a high resolution panoramic imaging system for visual inspection of vertical deep mine shafts. It provides the starting point for the development of a deployable device. The future development will need to focus on the following issues:

1. Using vertical displacement to compute the stereoscopic depth information, rather than the panoramic imaging. This will require the identification of the base line length; however, it eliminates the difficulties associated with outward looking stereo.

2. An interesting future area of research is to add metric iformation to the VR perception, i.e., to enable making the measurements in the VR environment.

3. Use affine invariant features to support reliable and precise computations of depth maps from images.

4. The affine invariant features can also be used to support new approaches for non-rigid stitching of the images.

5. More performant VR environments, tools and devices should be investigated.

Appendix A

Code

Listing A.1: Image acquisition and storage.

```
# coding=utf-8
#
# (c) Jakob Koenig 2016
#


# Imports
import os
import datetime
import sys
import logging
import RPi.GPIO as GPIO
import time
from picamera import PiCamera
import logging

# Set up logging
logging.basicConfig(filename='example.log', level=logging.DEBUG)

# Disable warnings
GPIO.setwarnings(False)

# Definitions
def dataManagement(layerNum,newSet):
    ''' This function is designed to check if
    directories exist and if not create
    them. It returns the directory in which
    the pictures of the current session are
    to be saved as string.'''
    # layerNum is the number of the current layer
```

```
31      # newSet is a boolean that indicates if the layer needs to be
32      # created in a new set folder (i.e. in the previous
33      # run the last layer of a set was created)
34
35      # Get current date
36      now = datetime.datetime.now()
37      # Only use Day Month and Year
38      today = now.strftime("%d%m%Y")
39
40      # Get list of directories that exist in the current working
        directory
41      directories = next(os.walk(os.getcwd(),topdown=True))[1]
42      # create pathname for the current date
43      pathToday = os.path.join(os.getcwd(),today)
44
45      # Check if folder with current date exists
46      if today in directories:
47          print("Todays directory already exists")
48          # if newSet is True a new folder SETN will be created
49          if newSet:
50              # create new Set folder and layer n folder
51              curSetPath = setManagement(pathToday,newSet)
52              curLayPath = layerManagement(layerNum,curSetPath)
53              return curLayPath
54          # if newSet is False no new SET folder will be created
        insted a
55          # new layer folder will be created in the current SET
        folder
56          else:
57              # create layer folder
58              curSetPath = setManagement(pathToday,newSet)
59              curLayPath = layerManagement(layerNum,curSetPath)
60              #print("This should happen if newSet False")
61              return curLayPath
62      else:
63          # if current date does not exist as folder, create folder
64          os.makedirs(pathToday)
65          print("New directory {} created".format(pathToday))
66          curSetPath = setManagement(pathToday,newSet)
67          curLayPath = layerManagement(layerNum,curSetPath)
68          return curLayPath
69
70  def setManagement(pathToday,newSet):
71      # create a list with folders in todays directory
72      folders = next(os.walk(pathToday,topdown=True))[1]
73      # Sort folders descending
74      folders.sort(reverse=True)
```

```
75      print("The following SET's were created today {}".format(
        folders))
76
77
78      # Wenn directories leer dann SET 1
79      # This happens even if newSet is False to ensure that
80      # there is always a folder
81      if not folders:
82          print("No previous SET detected, creating SET1")
83          setPath = os.path.join(pathToday,'SET1')
84          #if newSet:
85          os.makedirs(setPath)
86          return setPath
87          #else:
88          #    return setPath
89      # Sonst SETN+1
90      elif newSet:
91          # Find last set Number
92          lastSet = folders[0]
93          setNum = lastSet[3]
94          # Create new SetNumber
95          newSetNum = int(setNum) + 1
96          newSet = 'SET' + str(newSetNum)
97          # Create folder for current Set
98          setPath = os.path.join(pathToday,newSet)
99          os.makedirs(setPath)
100         print("New SET folder created")
101         print("Current Session will be saved in {}".format(newSet))
102         return setPath
103     else:
104         # Find last set Number
105         lastSet = folders[0]
106         setNum = lastSet[3]
107         # Create folder for current Set
108         setPath = os.path.join(pathToday,lastSet)
109         #os.makedirs(setPath)
110         print("Current Session will be saved in {}".format(lastSet)
        )
111         return setPath
112
113 def layerManagement(layerNumber,curSetPath):
114     # Create the new folder name and path
115     newLayer = 'LAYER' + str(layerNumber)
116     layerPath = os.path.join(curSetPath,newLayer)
117     #print("LayerPath {}".format(layerPath))
118
119     # get list of existing folders
```

```
120     folders = next(os.walk(curSetPath,topdown=True))[1]
121     #print("Folders {}".format(folders))
122 
123     # this is to ensure that the program exits safely even if the
        layer
124     # already exists
125     if newLayer in folders:
126         print("Layer already exists.")
127         sys.exit()
128     else:
129         # create the layer folder
130         os.makedirs(layerPath)
131         print("Created Layer {}".format(layerPath))
132         return layerPath
133 
134 #def logSession():
135 
136 def startUp():
137 
138     #cameraSetUp()
139     gpioSetUp()
140     moveToZero()
141 
142     return True
143     # Does not quite work yet!
144 ##    if not GPIO.event_detected(2):
145 ##        moveToZero()
146 ##        return True
147 ##    else:
148 ##        return True
149 
150 def cameraSetUp():
151     camera = PiCamera()
152     camera.vflip = True
153     camera.resolution = (2592,1944)
154 
155 def gpioSetUp():
156     # Set up GPIO
157     # Mode
158     GPIO.setmode(GPIO.BCM)
159 
160     # Inputs
161     # BCM 2 as pull down switch always open power in
162     # is BCM 2 Ground to GND Pin
163     GPIO.setup(2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
164 
165     # Outputs
```

```
166     # Direction BCM 23
167     GPIO.setup(23, GPIO.OUT)
168     # Step BCM 24
169     GPIO.setup(24, GPIO.OUT)
170     # Enable Driver BCM 18
171     GPIO.setup(18, GPIO.OUT)
172
173
174     # Set Direction
175     GPIO.output(24, True)
176
177     # GPIO events
178     GPIO.add_event_detect(2, GPIO.FALLING, callback=callbackOrigin,
        bouncetime=300)
179
180
181
182 def callbackOrigin(channel):
183     global origin
184     # define what happens when origin is passed
185     if not origin:
186         origin = True
187         print("Falling edge on 2; Camera has reached origin \n")
188
189
190 def moveToZero():
191     n = 0
192     waitTime = 0.01
193     # GPIO.input(2) returns True unless it is pressed
194     try:
195         #GPIO.event_detected is True when button is pressed
196         while not GPIO.event_detected(2):
197             GPIO.output(23, True)
198             GPIO.output(23, False)
199             time.sleep(waitTime)
200             n += 1
201
202
203             # Implement rising and falling edge for stepper timing
       here
204         print("Stepnumber {}".format(n))
205         time.sleep(2)
206     except KeyboardInterrupt:
207         GPIO.cleanup()
208     #GPIO.cleanup()
209     return True
210
```

```
211
212 def dataAquisition(nSection,curLayPath):
213     print("Starting Dataaquistion: N Pictures")
214     camera = PiCamera()
215     camera.vflip = True
216     camera.hflip = True
217     camera.resolution = (2592,1944)
218     n = 0
219     i = 1
220     waitTime = 0.01
221     try:
222         # Runs until button press is detected (rolling Switch)
223         while not GPIO.event_detected(2):
224             while n<nSection:
225                 GPIO.output(23, True)
226                 GPIO.output(23, False)
227                 time.sleep(waitTime)
228                 n += 1
229                 # Implement rising and falling edge for stepper
    timing here
230             print("Stepnumber for this Section: {}".format(n))
231             n = 0
232             # Take picture and save here
233             camera.start_preview()
234             time.sleep(2)
235             # Optimize namegiving for pictures
236             picPath = str(curLayPath) + '/PICTURE' + str(i) + '.jpg
    '
237             camera.capture(picPath)
238             camera.stop_preview()
239             i += 1
240             # make a log entry after every pic
241             logging.debug('%s', str(picPath))
242     except KeyboardInterrupt:
243         GPIO.cleanup()
244     #GPIO.cleanup()
245
246 def logFileManagement():
247     # This module reads the last line in log file and returns the
    values for
248     # this session
249     fileHandle = open('example.log',"r")
250     lineList = fileHandle.readlines()
251     fileHandle.close()
252
253     lastLine = lineList[-1]
254     print(lastLine)
```

```
255     lastSession = strManagement(lastLine)
256
257     return lastSession
258
259
260 def strManagement(string):
261     strList = string.split('/')
262     curPic = strList[len(strList)-1]
263     curPic = curPic[0:8] # CHANGE THIS DEPENDING ON PIC NAME FORMAT
264     curLay = strList[len(strList)-2]
265     curSet = strList[len(strList)-3]
266     curDat = strList[len(strList)-4]
267
268     curStrList = [curDat, curSet, curLay, curPic]
269
270     return curStrList
271
272 def sessionManagement(lastSession,newSet):
273     # CHANGE THIS CONDITION IF MORE THAN 8PICS ARE TAKEN
274     if 'PICTURE8' in lastSession and not newSet:
275         layerNum = int(lastSession[2][5]) +1
276         return layerNum
277     elif newSet:
278         layerNum = 1
279         return layerNum
280     else:
281         print("previous Session not completed. Investigate!")
282         sys.exit()
283
284
285 # Program
286 #layerNum = 2 #sys.argv[1]
287 newSet = True#sys.argv[2]
288 nSection = 50   # This needs to be adjusted if stepsize
289                 # changes or more cameras are wanted
290
291
292 # Global variable origin
293 origin = False
294
295 # Get data from last session
296 lastSession = logFileManagement()
297 print(lastSession)
298 layerNum = sessionManagement(lastSession,newSet)
299
300 # if newSet True layerNum must be 1!
301 curLayPath = dataManagement(layerNum,newSet)
```

```
302 print("Return dataManagement: {}".format(curLayPath))
303 startUp()
304 print("Camera has passed through origin:{}".format(origin))
305 dataAquisition(nSection,curLayPath)
306
307 # TODo
308 # create LogFile for propper startup
309 # see if button press is true at startup
```

Listing A.2: Code to create stereo pair of panoramic images from one 360° rotation of the camera.

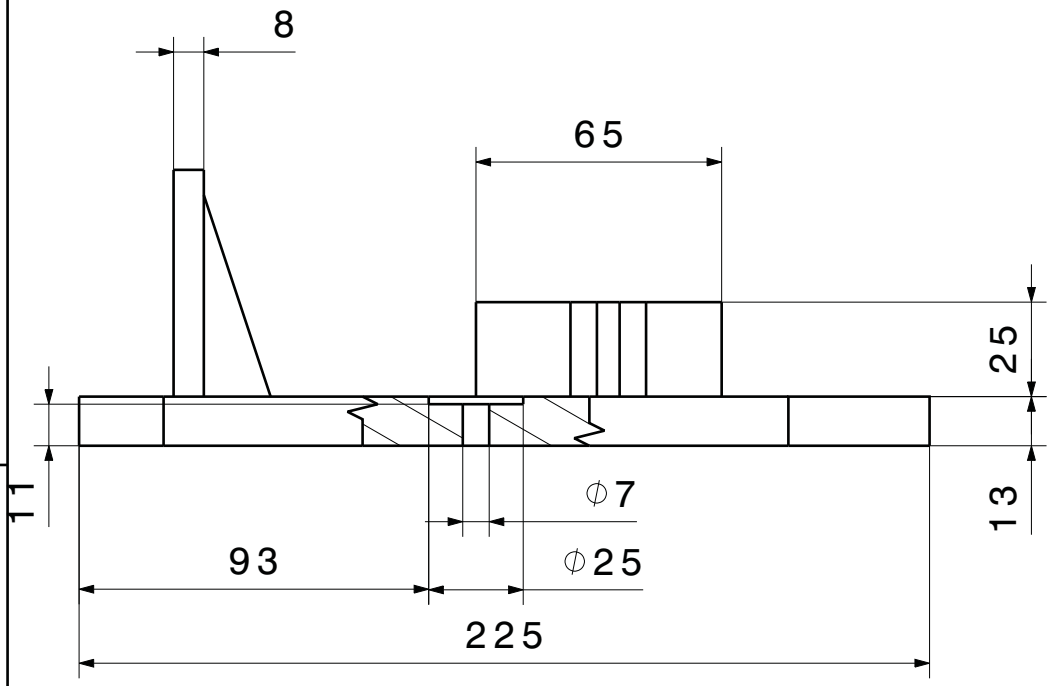
```
%
% (c) Jakob Koenig 2017
%
clear;
close all;

% pan = zeros(480,1,3);
% leftPan = zeros(480,1,3);
% rightPan = zeros(480,1,3);
pan = [];
leftPan = [];
rightPan =[];

n = 1;
d = 10;
center = 320;
m = 480;

centerColumn = zeros(m,n,3);
rightView = zeros(m,n,3);
leftView = zeros(m,n,3);

for k=3199:-1:0
    fileName = sprintf('stereoTest640/picture%d.jpg',k);
    image = imread(fileName);

    rightView(:,1,:) = image(:,center-d,:);
    %rV = image(:,center-d:center-d+n-1,:);
    centerColumn(:,1,:) = image(:,center,:);
    %cV = image(:,center-n+1:center,:);
    leftView(:,1:n,:) = image(:,center+d,:);%560:560+n-1);
    %lV = image(:,center+d:center+d+n-1,:);%560:560+n-1);

    pan = [pan,centerColumn];
    %pC = cat
    leftPan = [leftPan, leftView];
    rightPan = [rightPan, rightView];
end
% figure, imshow(image)
%
% figure
% subplot(1,3,1), imagesc(leftView);
% subplot(1,3,2), imagesc(centerColumn);
% subplot(1,3,3), imagesc(rightView);
```

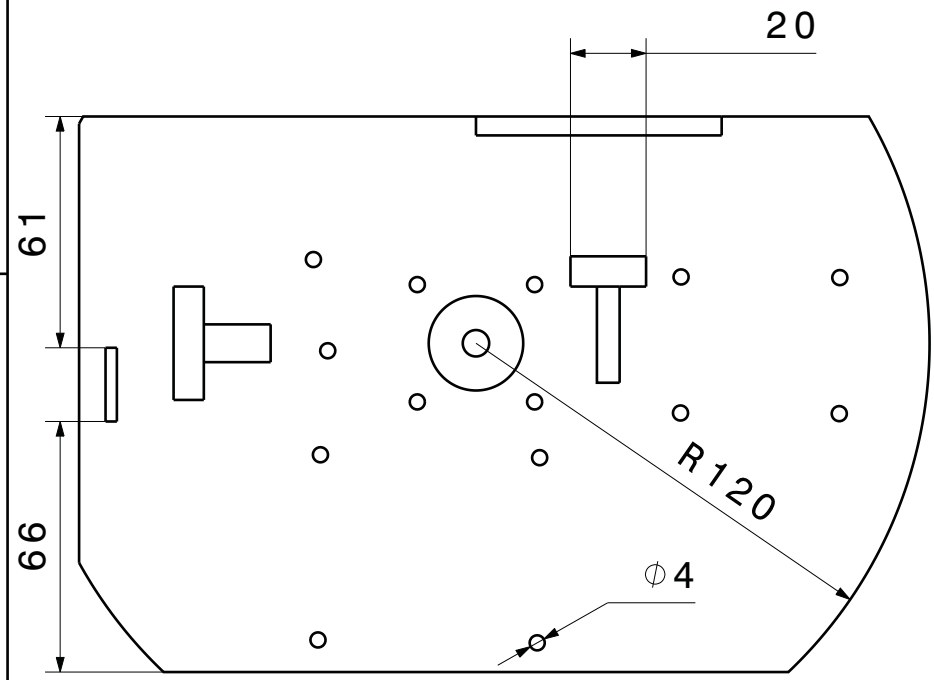
```
45
46 pan = uint8(pan);
47 leftPan = uint8(leftPan);
48 rightPan = uint8(rightPan);
49
50 save('singleCamStereoPan.mat');
51
52 %%
53 clear;
54 close all;
55
56 load('singleCamStereoPan.mat');
57
58 figure
59 subplot(3,1,1), imagesc(pan);
60 title('Panoramic Image');
61 subplot(3,1,2), imagesc(leftPan);
62 title('Left Eye View Panorama');
63 subplot(3,1,3), imagesc(rightPan);
64 title('Right Eye View Panorama');
65 %colormap('gray');
66
67 figure
68 imshow(pan);
69
70 print('panTiff', '-dtiff');
71 print('panJPG', '-djpeg');
72
73
74 %%
75 clear;
76 close all;
77
78 load('singleCamStereoPan.mat');
79
80 anaglyph = stereoAnaglyph(leftPan,rightPan);
81
82 figure
83 imshow(anaglyph);
84
85
86 %% RRS
87
88 clear;
89 close all;
90
91 load('singleCamStereoPan.mat');
```

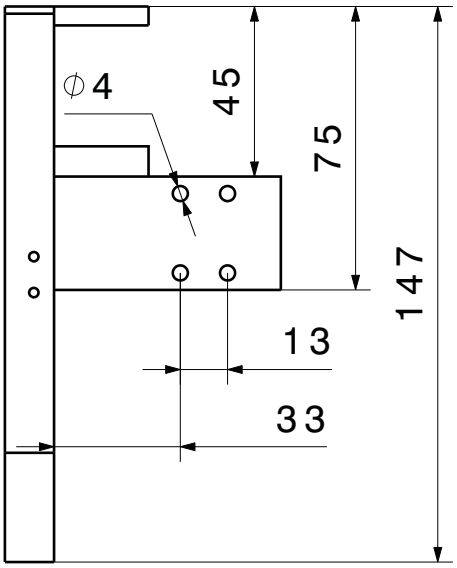
```
92
93 myFile = 'panTiff.tif';
94
95 rset_file = rsetwrite(myFile);
96 imtool(rset_file);
```

Appendix B

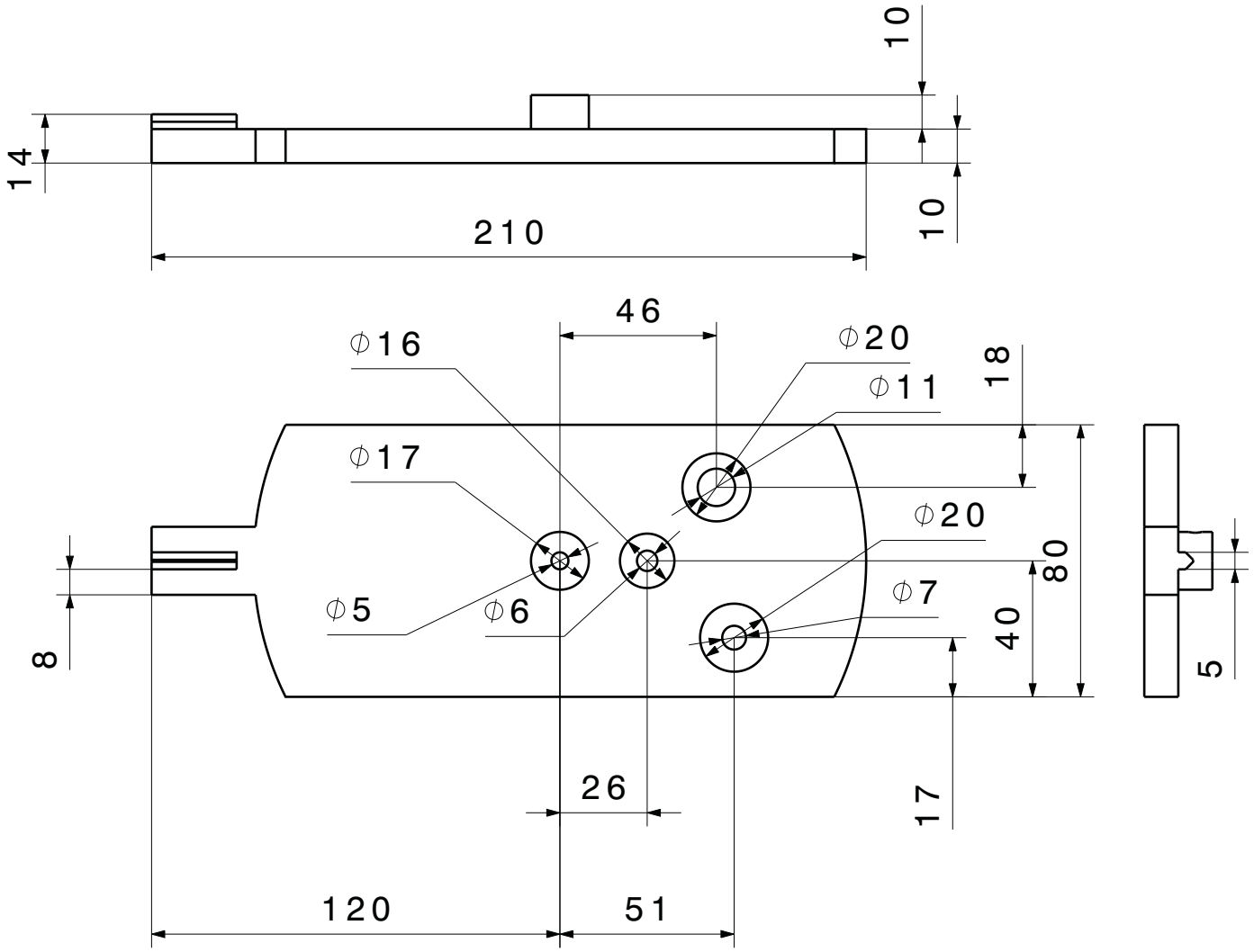
Drawings







DESIGNED BY: Jakob König DATE: 11.03.2017		# Mounting Plate		
CHECKED BY: Jakob König DATE: 11.03.2017				
SIZE A4		## Chair of Automation		
SCALE 1:2	WEIGHT (kg) XXX	DRAWING NUMBER XXX	SHEET 1/1	



DESIGNED BY: Jakob König DATE: 13.03.2017		# Base Plate	I _
CHECKED BY: Jakob König DATE: 13.03.2017			H _
			G _
			F _
SIZE A4		## Chair of Automation	E _
			D _
			C _
SCALE 1:2	WEIGHT (kg)	DRAWING NUMBER MountingPlate	SHEET 1/1 B _
			A _

Bibliography

[1] Duke Gledhill et al. “Panoramic imaging—a review”. In: *Computers & Graphics* 27.3 (2003), pp. 435–445. ISSN: 00978493. DOI: 10.1016/S0097-8493(03)00038-4.

[2] S. Peleg, Y. Pritch, and M. Ben-Ezra. “Cameras for stereo panoramic imaging”. In: *IEEE conference on computer vision and pattern recognition*. IEEE Comput. Soc, 2000, pp. 208–214. ISBN: 0-7695-0662-3. DOI: 10.1109/CVPR.2000.855821.

[3] Richard Szeliski. “Image Alignment and Stitching: A Tutorial”. In: *Foundations and Trends® in Computer Graphics and Vision* 2.1 (2006), pp. 1–104. ISSN: 1572-2740. DOI: 10.1561/0600000009.

[4] Matthew Brown and David G. Lowe. “Automatic Panoramic Image Stitching using Invariant Features”. In: *International Journal of Computer Vision* 74.1 (2007), pp. 59–73. ISSN: 0920-5691. DOI: 10.1007/s11263-006-0002-3.

[5] *krpano.com*. URL: https://krpano.com/.

[6] Kelly E. Arbuckle and John A. Downing. “Freshwater mussel abundance and species richness: GIS relationships with watershed land use and geology”. In: *Canadian Journal of Fisheries and Aquatic Sciences* 59.2 (2002), pp. 310–316. ISSN: 0706-652X. DOI: 10.1139/f02-006.

[7] National Geographic Society. *GIS (geographic information system)*. 26.02.2017. URL: http://www.nationalgeographic.org/encyclopedia/geographic-information-system-gis/.

[8] *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi*. URL: https://www.raspberrypi.org/.

[9] Rajes Ramli. *Latest Features About Raspberry Pi 3 That You Have To Know - Linux Info*. 10.03.2017. URL: https://linuxinfoid.blogspot.co.at/2016/03/the-latest-features-about-raspberry-pi.html.

[10] *Remote Access Software for Desktop and Mobile — RealVNC*. URL: https://www.realvnc.com/.

[11] *8-megapixel Raspberry Pi Camera Module v2*. URL: http://www.raspberrypi-spy.co.uk/2016/04/8-megapixel-raspberry-pi-camera-module-v2/.

[12] *IMX219PQ — Sony Semiconductor Solutions*. URL: http://www.sony-semicon.co.jp/products_en/new_pro/april_2014/imx219_e.html.

[13] *7. Camera Hardware — Picamera 1.10 documentation*. 9.11.2015. URL: http://picamera.readthedocs.io/en/release-1.10/fov.html.

[14] *Raspberry Pi 3 Model B - Raspberry Pi*. 14.03.2017. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[15] *Camera Module - Raspberry Pi*. 14.03.2017. URL: https://www.raspberrypi.org/products/camera-module-v2/.

[16] *AutoStitch*. 31.07.2015. URL: http://matthewalunbrown.com/autostitch/autostitch.html#FAQ.

[17] *6. Camera Hardware — Picamera 1.12 documentation*. 30.01.2017. URL: http://picamera.readthedocs.io/en/release-1.12/fov.html.

[18] *manfrotto*. URL: https://www.manfrotto.us/junior-geared-head.

[19] *— RS Pro Hybrid Schrittmotor 0.9°, 44Ncm, 4-adriger Anschluss, 1,68 A 2,8 V —*. URL: http://at.rs-online.com/web/p/schrittmotoren/5350401/.

[20] Brian Schmalz. *Easy Driver stepper motor driver*. 8.04.2016. URL: http://www.schmalzhaus.com/EasyDriver/.

[21] *PowerCore 20100*. URL: https://www.anker.com/products/A1271012.

[22] *DROK® LTC187 DC Boost-Wandler 3.5-30V 100W Stromversorgung Spannungsregler 5V/12V Step Up Volt-Modul mit Spannungsmesser Eingang Ausgabe abwechselnd LED-Anzeige für Auto-Motor Motorrad, etc: Amazon.de: Beleuchtung*. URL: https://www.amazon.de/Boost-Wandler-Stromversorgung-Spannungsregler-Spannungsmesser-abwechselnd/dp/B00HY3QBUE/ref=sr_1_3?ie=UTF8&qid=1489047774&sr=8-3&keywords=DEOK+DC+12V+Boost.

[23] *Microstepping, Full Step & Half Step*. 2014. URL: http://www.nmbtc.com/step-motors/engineering/full-half-and-microstepping/.

[24] Greg. *Raspberry Pi 3 GPIO Pin Layout*. URL: http://blog.mcmelectronics.com/post/Raspberry-Pi-3-GPIO-Pin-Layout#.WMWhoDs1-bg.

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

[26] *Raspberry Pi*. 10.03.2016. URL: http://www.truetex.com/raspberrypi.

[27] Amir Badshah et al. “Non-rigid registration for qualitiy control of printed materials”. In: ed. by Jean-Charles Pinoli et al. SPIE Proceedings. SPIE, 2011, 80000J. DOI: 10.1117/12.890901.

[28] Peter Peer and Franc Solina. “Panoramic Depth Imaging: Single Standard Camera Approach”. In: *International Journal of Computer Vision* 47.1 (2002), pp. 149–160. ISSN: 1573-1405. DOI: 10.1023/A:1014541807682.

[29] Christian Sallinger. *Panoramic optical servoing - a new dimension in the inspection and repair of refractories using a telerobot: Zugl.: Leoben,Univ., Diss., 2004*. 1. Aufl. Göttingen: Cuvillier, 2005. ISBN: 3865373194.

[30] Anders Johansson. *Stereoscopy: Fooling the Brain into Believing There is Depth in a Flat Image*. 2009. URL: http://www.diva-portal.org/smash/get/diva2:232456/FULLTEXT01.

[31] Leonidas M. Quintana. “Anaglyph 3D in vascular neurosurgery: excellent exercise for the neurosurgeon’s brain”. In: *World neurosurgery* 82.3-4 (2014), e417–8. ISSN: 1878-8750. DOI: 10.1016/j.wneu.2013.03.002.

[32] *Create reduced resolution data set from image file - MATLAB rsetwrite - MathWorks Deutschland*. URL: https://de.mathworks.com/help/images/ref/rsetwrite.html.

[33] *ArcGIS Online*. 21.12.2016. URL: https://www.arcgis.com/home/index.html.