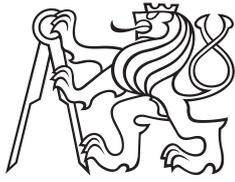


Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

3D reconstruction of indoor boulders

Martin Forman

Supervisor: Ing. Jan Čech, Ph.D.

Field of study: Open Informatics

Subfield: Artificial intelligence

January 2024

I. Personal and study details

Student's name: **Forman Martin** Personal ID number: **498842**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Open Informatics**
Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

3D Reconstruction of Indoor Boulders

Bachelor's thesis title in Czech:

3D rekonstrukce vnitřních lezeckých boulderů

Guidelines:

The climbing sport has been very popular lately. There are many climbing gyms, where the routes, called problems, are regularly changed. Basic walls are usually piecewise planar with different slopes and contain holds of the same color. The athlete's task is to climb the route, usually to the top hold or to climb over the top platform.

Create a tool for automatic 3D reconstruction of bouldering routes, which will allow measuring distances between holds, or various angles in the scene (for example, the deviation of the basic wall from the gravitational vertical). The tool should work with a small number of photographs of the route, taken with an ordinary mobile phone.

Evaluate the accuracy of the method using several physical measurements in the scene and using a depth camera.

Optional:

- Develop an application including GUI, which will allow interactive measurement from photographs.
- Use monocular depth estimation for the reconstruction.

Bibliography / sources:

- [1] R. Hartley, A. Zisserman. Multiple View Geometry in Computer Vision. 2nd edition, Cambridge University Press, 2004.
- [2] Godard, Clément, Oisín Mac Aodha, and Gabriel J. Brostow. "Unsupervised monocular depth estimation with left-right consistency." Proc. CVPR, 2017.
- [3] Ming Yan, Xin Wang, Yudi Dai, Siqi Shen, Chenglu Wen, Lan Xu, Yuexin Ma, Cheng Wang. CIMI4D: A Large Multimodal Climbing Motion Dataset under Human-scene Interactions. Proc. CVPR, 2023.
- [4] Katsuhito Sasaki, Keisuke Shiro, Jun Rekimoto. ExemPoser: Predicting Poses of Experts as Examples for Beginners in Climbing Using a Neural Network. Proc. of the Augmented Humans International Conference, 2020.

Name and workplace of bachelor's thesis supervisor:

Ing. Jan Čech, Ph.D. Visual Recognition Group FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **02.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Jan Čech, Ph.D.
Supervisor's signature

prof. Dr. Ing. Jan Kybic
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

Acknowledgements

I would like to thank my supervisor Ing. Jan Čech Ph.D. for the help he provided with writing and researching my thesis. I would also like to thank the climbing gym Smichoff ¹ in Prague and Hudy boulder Karlín ² for letting me collect a dataset of boulders for my work. Many thanks too belongs to the contributors and authors of libraries used in the implementation of this project, namely: Kornia, PyQt, Open CV, sklearn, GCS and more.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

A LLM was used for help with some coding tedious tasks, L^AT_EX formatting and testing research ideas.

¹Smichof climbing gym website:
<https://www.lezekecentrum.cz/cs/>

²Hudy climbing gym website:
<https://www.hudysteny.cz/boulderkarlin/>

Abstract

In this thesis, the 3D reconstruction of indoor climbing boulders from two images was investigated. An application was developed to measure distances between holds, as well as other objects and angles within the scene. The methodology combined classical Structure from Motion (SfM) techniques with recent learned descriptors for matching, specifically LoFTR.

The application's performance was evaluated for accuracy, revealing that multiple reconstructed scenes achieved measurement errors around 5%. However, some cases exhibited higher errors due to factors such as low texture, poor image quality, and sub-optimal baselines. The evaluation results informed a discussion on potential improvements and identified limitations of the current approach.

Keywords: SfM, Structure from Motion, 3D reconstruction, Boulder 3D reconstruction, boulder, 3D models, point cloud, 2D to 3D, RANSAC, GCS, depth estimation, indoor climbing, LoFTR, feature matching, image-based reconstruction, photogrammetry, measurement accuracy, texture mapping, computer vision, scene reconstruction, distance measurement, angle measurement, image quality, baseline, error analysis, application development

Supervisor: Ing. Jan Čech, Ph.D.
Fakulta Elektrotechnická ČVUT,
Na Zderaze 269/4,
121 35 Praha 2

Abstrakt

V této práci byla zkoumána 3D rekonstrukce vnitřních lezeckých boulderů ze dvou snímků. Byla vyvinuta aplikace pro měření vzdáleností mezi chyty, stejně jako jiných objektů a úhlů ve scéně. Metodologie kombinovala klasické techniky Structure from Motion (SfM) s nedávnými učenými deskriptory pro párování, konkrétně LoFTR.

Výkon aplikace byl hodnocen z hlediska přesnosti, přičemž bylo zjištěno, že více rekonstruovaných scén dosáhlo měřicí chyby kolem 5%. Některé případy však vykazovaly vyšší chybu kvůli faktorům jako nízká textura, špatná kvalita snímků a neoptimální baseline. Výsledky hodnocení vedly k diskuzi o možných vylepšeních a identifikaci omezení současného přístupu.

Klíčová slova: SfM, Structure from Motion, 3D rekonstrukce, 3D rekonstrukce boulderů, boulder, 3D modely, mračno bodů, 2D na 3D, RANSAC, GCS, odhad hloubky, indoor lezení, LoFTR, párování prvků, rekonstrukce založená na obrazech, fotogrammetrie, přesnost měření, mapování textur, počítačové vidění, rekonstrukce scény, měření vzdáleností, měření úhlů, kvalita obrazu, základna, analýza chyb, vývoj aplikací

Překlad názvu: 3D rekonstrukce vnitřních lezeckých boulderů

Contents

1 Introduction	1	6.3 Automatic scale detection.....	45
2 Related Work	3	6.4 Inventing new use cases and adding features	45
3 The solution	7	7 Conclusion	47
3.1 Getting the 3D model	7	Bibliography	49
3.1.1 Creating the data set (capturing images)	7		
3.1.2 Camera calibration	9		
3.1.3 Correspondence matching ...	10		
3.1.4 Epipolar Geometry	11		
3.1.5 Rectifying images	13		
3.1.6 Dense stereo matching	13		
3.1.7 Interpolating disparity	14		
3.1.8 Directly relating images (projection)	14		
3.1.9 Triangulation into 3D	15		
3.2 Getting measurements from 3D data	16		
3.2.1 Measuring real world distance	18		
3.2.2 Measuring angles in the scene	18		
4 Implementation details	19		
4.1 Wrapping a library for use in python	19		
4.2 Creating a GUI application	20		
4.2.1 Data selection	20		
4.2.2 Processing	21		
4.2.3 Result Visualization	21		
5 Experiments	25		
5.1 cross-validation method	25		
5.2 Measurement accuracy evaluation	25		
5.2.1 Measuring only using regularities	26		
5.2.2 How distance from reference measurement impacts results	27		
5.3 Angle measurement evaluation .	27		
5.4 3D point cloud evaluation.....	28		
5.5 Expansion on results	28		
5.5.1 Dataset quality	28		
5.5.2 Reference measurement quality	29		
5.5.3 Other factors influencing quality	29		
6 Limitations and future Work	43		
6.1 Improving accuracy	43		
6.1.1 Creating datasets	44		
6.1.2 Self-calibration	44		
6.2 Building a mobile app	44		

Figures

<p>1.1 Boulder and its distinct features (chalk and shoe marks)..... 2</p> <p>2.1 COLMAP [17, 16] result with 5 images. 4</p> <p>2.2 Depth-Anything [22] result from a single image (a model based monodepth algorithm approximates depth without any other data.) 5</p> <p>3.1 3D reconstruction pipeline overview. 8</p> <p>3.2 Example image pairs from the collected dataset. 9</p> <p>3.3 Camera matrix K and its example from a iPhone camera. 10</p> <p>3.4 Comparison between results of SIFT and LoFTR correspondence matching. See that LoFTR (bottom) returns much denser matches between points. 11</p> <p>3.5 Detail of LoFTR matches (every tenth match drawn for clearness) on a small section of the image. 12</p> <p>3.6 Epipolar geometry [6]. 12</p> <p>3.7 Fundamental matrix problem. . . 12</p> <p>3.8 Rectified images. 13</p> <p>3.9 Disparity map (note that many values are present but not visible). 14</p> <p>3.10 Interpolated disparity. 15</p> <p>3.11 Summary of calculations needed to get possible R and t. 16</p> <p>3.12 Camera projection matrices P_1, P_2. 17</p> <p>3.13 Example of reconstructed 3D points from different views. 17</p> <p>3.14 Original image and a reconstructed model coloured by it. 17</p> <p>4.1 Code snippets from GCS Python interface. 20</p> <p>4.2 Compilation of C++ to create a library to interface with. 21</p> <p>4.3 GUI app home window, image loading and processing views. 21</p> <p>4.4 Visualization widgets and the related menu. 22</p>	<p>4.5 Snippets of Python used to scale distance measurements. 23</p> <p>4.6 Calculation of the angle between two planes using their normal vectors. 24</p> <p>5.1 Statistical formulas used to evaluate results. Here, d_i represents the estimated distances, m_i represents the ground truth distances, N is the number of data points, and \bar{d} is the mean of the data points. For each of these statistics we show the mean value and standard deviation over the cross-validation bins. 26</p> <p>5.2 Snippets of Python used for cross-validation. 30</p> <p>5.3 Regular elements with known distance. Mounting holes spaced every 20 cm diagonally (L), and square panels of dimensions 100×100 cm (R). 30</p> <p>5.4 Example of measurements by author (L), measurements using regularities (R). 31</p> <p>5.5 Measurements and their statistics for 26 measurements. 31</p> <p>5.6 Measurements and their statistics for 15 measurements. This is one of the fail cases, where probably the lack of texture compared to other cases led to much greater error. 32</p> <p>5.7 Example of relative mean error, observe that almost no bias is present overall, since the individual measurements cancel out. 32</p> <p>5.8 Measurements and their statistics for 11 measurements. 33</p> <p>5.9 Measurements and their statistics for 10 measurements. 34</p> <p>5.10 Measurements and their statistics for 10 measurements. 35</p> <p>5.11 Measurements and their statistics for 10 measurements. 36</p>
---	--

5.12 Graph of relative error based on distance from reference measurement, showing the increase in error the further the reference.	36
5.13 Example boulder sketch from Hudy Karlín boulder.	37
5.14 Example angle measurements in a single scene shown in fig. 5.15. . . .	37
5.15 Angle measurement statistics for 5 measurements. The red and blue points mark the selected planes and the area is transparently colored. . .	38
5.16 Angle measurement statistics for 2 measurements.	39
5.17 Angle measurement statistics for a single measurements (only angle in the reconstructed pointcloud).	40
5.18 A untextured 3D reconstruction and its detail showing flat and uneven regions depending on angle towards camera. The original image is also shown for comparison.	41
6.1 A Lidar 3D scan of a boulder captured by a iPhone 14 Pro lidar scanner, using the application 3D scanner. The same image is shown with (R) and without (L) texture, to better display the accuracy and spatial textures captured.	44



Chapter 1

Introduction

This thesis focuses on building a python based application that lets users reconstruct a 3D model from images of a climbing boulder and use the captured model to retrieve data about angles and distances in the scene. To provide a seamless experience and make the application widely available, we opted to reconstruct the 3D model only using two images, that can be captured by any image capturing device (phone camera, DSLR, etc.), that only has to be calibrated once (can be done offline) and allows for hand held image capture and a varied baseline (distance between the capture points of the two images).

The ability to capture accurate 3D data without the use of specialized technologies like LiDAR or stereo camera rigs, is the main goal of this thesis. To achieve this goal, a classical 3D Reconstruction pipeline is employed, with some modern tools from the Kornia [8] library, that uses machine learning to improve on the performance of traditional algorithmic approaches to some tasks.

As mentioned, the limitation of two images was selected to create a user friendly system which could be integrated into a desktop or mobile app. It is also interesting to experiment with the limitations of creating 3D models with limited data and how specific domain knowledge can substitute this data. For boulders, we can assume we are reconstructing a partially planar scene (a wall with mostly neutral colours) that contains easily recognisable texture, like climbing holds, shoe marks or chalk marks (seen in figures 1.1).

This texture is what allows good performance of the selected pipeline, mostly regarding the dense stereo reconstruction phase, where detection of the same physical points in the provided images is performed. The linear interpolation method was also selected based on the planar nature of the boulder.

Since accuracy is part of the goal of 3D reconstruction, 3D information gathered from the scene by different methods will be used for comparison. We show the viability of the more traditional two picture SfM approach against open source tools like COLMAP [16, 17] (a very powerful reconstruction tool-set, however mainly focused on big scenes with many input images). We also compare to real world collected measurements and calculate the error of our reconstruction.

For completeness, it is better to state that 3D reconstruction from a hand-

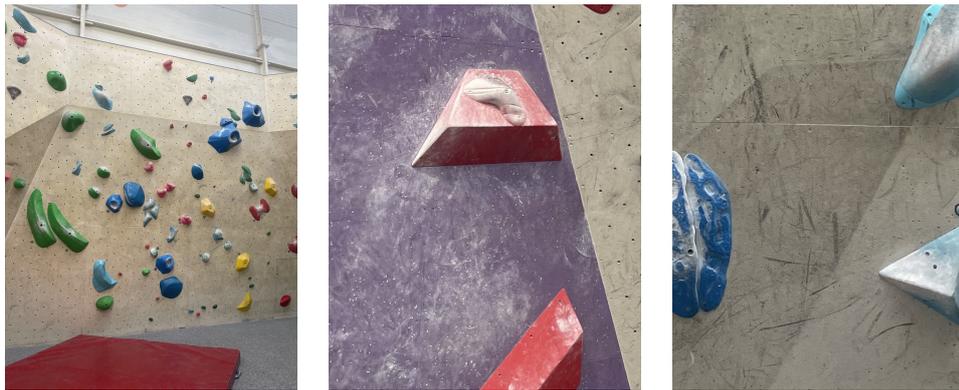


Figure 1.1: Boulder and its distinct features (chalk and shoe marks).

held camera will never be the most accurate way of obtaining a 3D model, as many steps in its process are based on some form of estimates. Other techniques of scanning 3D space can actually capture the depth information in a scene or more accurately estimate it. LiDAR is specifically a technology that accurately captures the full 3D scene and is not affected by aspects like lighting which can impact photogrammetry results (photogrammetry includes SfM and our approach). Other techniques like infrared 3D scanning or photogrammetry using a stereo camera (calibrated with fixed distance between two cameras) can also produce more accurate results.



Chapter 2

Related Work

Searching for related work to the specific task of photogrammetric reconstruction of 3D indoor boulders, to the best of our knowledge, no similar research comes up. The closest research project [13] was concerned with reconstructing outdoor climbing walls (natural not artificial) and mainly the individual climbing routes on them. Although the research also uses a fairly traditional tool-set of SfM methods, an approach of using 400 images was taken by this research, and the focus on outdoor climbing walls means there is a difference in the textures and also the scale of the reconstruction process. Another related research article [20, p.7] we found also focused on outdoor boulders of much greater dimensions and used many images, but provides a interesting insight about scaling and model accuracy:

It was shown that control points are not only necessary to scale the model but also to compensate for the non-linear model misalignment. Indeed, the accuracy can be considerably improved by using control points and performing the optimisation process in Photoscan.

Based on this research, our pipeline will have to overcome these issues in another way, since we do not have control points (manually added elements on the boulder) for the sake of ease of use.

Most of the cited work that was used to build the basis for the 3D reconstruction program is general SfM content (research or otherwise). Even though the theory for 3D reconstruction is generally well defined and easily implemented, many of the methods perform differently for each data set and a lot of experimental work has to be done to get good performance. To get accurate results, tiny mistakes have to be corrected or eliminated, otherwise the error multiplies in the pipeline and the results suffer accordingly.

This could be viewed as one of the benefits of this work, as no other research we found tackles the specific issues encountered in creating a good 3D reconstruction application for indoor boulders while not using a lot of input data but only two images.

Even though the specific problem of boulders is not commonly researched, we need to take into account many general reconstruction tools that try to offer good result for any object (boulder or other). Such tools include COLMAP [17, 16], which is a very powerful tool, but not well suited for our specific task, managing mainly reconstruction from thousands of pictures, videos or

3D scans. However when we try to give it limited data it fails to produce good results. See figure 2.1, where after inputting 5 images, COLMAP only reconstructed around 5000 points of the scene. Using only two images produced even worse results.

There are other commercial tools that reconstruct 3D data from images or

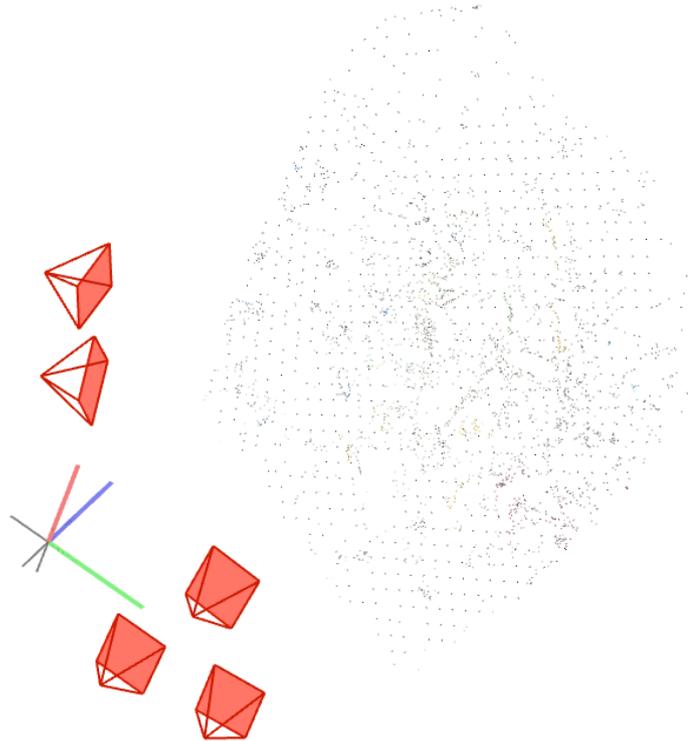


Figure 2.1: COLMAP [17, 16] result with 5 images.

video like Capturing reality [1], the distinguishing factor compared to this work being again the amount of images.

Other research it is interesting to compare to is Depth-Anything [22]. This is a mono-depth model trained to estimate depth from a single view. Although the relative results visually seem quite accurate (as seen in figure 2.2), this can only serve as part of the pipeline (providing relative depth information), and we need to follow it up with some SfM processing to retrieve a 3D model. Other mono-depth tools exist and research into them includes [9], however Depth-Anything is the state of the art solution at the moment of writing.



Figure 2.2: Depth-Anything [22] result from a single image (a model based monodepth algorithm approximates depth without any other data.)

Chapter 3

The solution

In this chapter, the process of getting a 3D reconstruction out of only two images while still providing an accurate result will be discussed together with relevant theory. Parts of the experiments will also be shown through comparisons between available methods and algorithms with examples of the effect on the resulting 3D model.

3.1 Getting the 3D model

Constructing a 3D model is a multi-step process that starts with the raw source images and uses SfM theory to get inter-image correspondences, estimate Fundamental and Essential matrices (discussed further) and using triangulation to calculate the 3D points estimation. Along this process many steps can be taken to improve the result, starting with camera calibration, outlier filtration and interpolation. All these steps combined get us a more accurate and more complete result. See the fig. 3.1 for a quick overview or read the description of each step below.

3.1.1 Creating the data set (capturing images)

For purposes of testing the thesis code, a dataset of images of boulders had to be collected. As no publicly available datasets were found, data collection was performed by the author, by capturing bordering gyms in Prague (mostly Smichoff and Hudy Boulder Karlín).

Important aspects of the images are: a good baseline distance (translation) between the two capture points (for the same scene) was chosen to make the results of SfM more accurate; the same lens (the same camera) was used for capturing the image pair (a camera matrix estimated from calibration was applied to both images); Distinct parts of the scene were captured in both images (the majority of the actual subject - the boulder, was in view for both pictures), while also fitting in a part of the ground for better context.

Elaborating on the baseline choice, we always need to find a compromise between matching accuracy and reconstruction error. Regarding matching accuracy, a smaller baseline enables better performance of LoFTR [18] and subsequent dense stereo GCS [5], which yields more matched points, improving

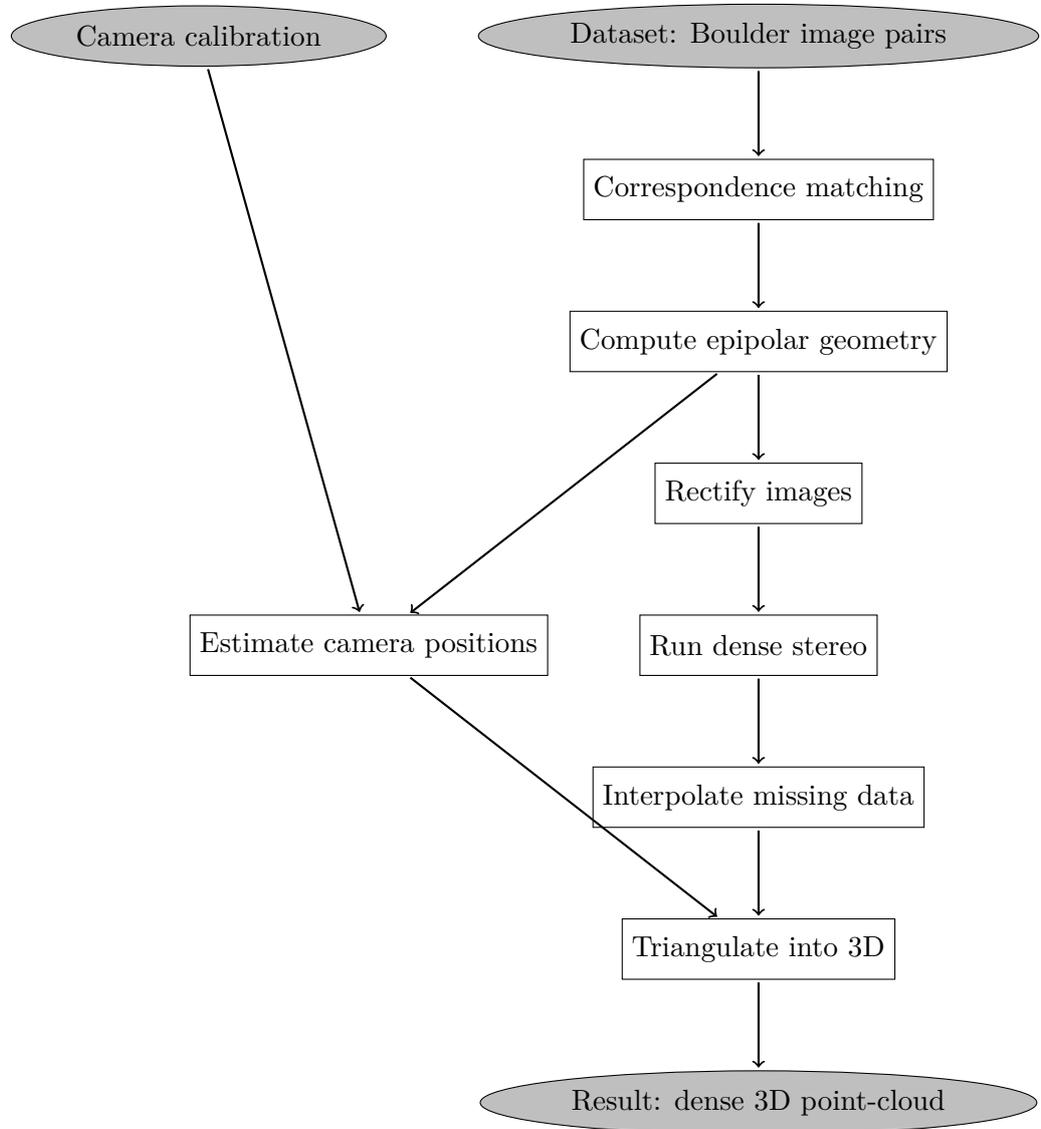


Figure 3.1: 3D reconstruction pipeline overview.

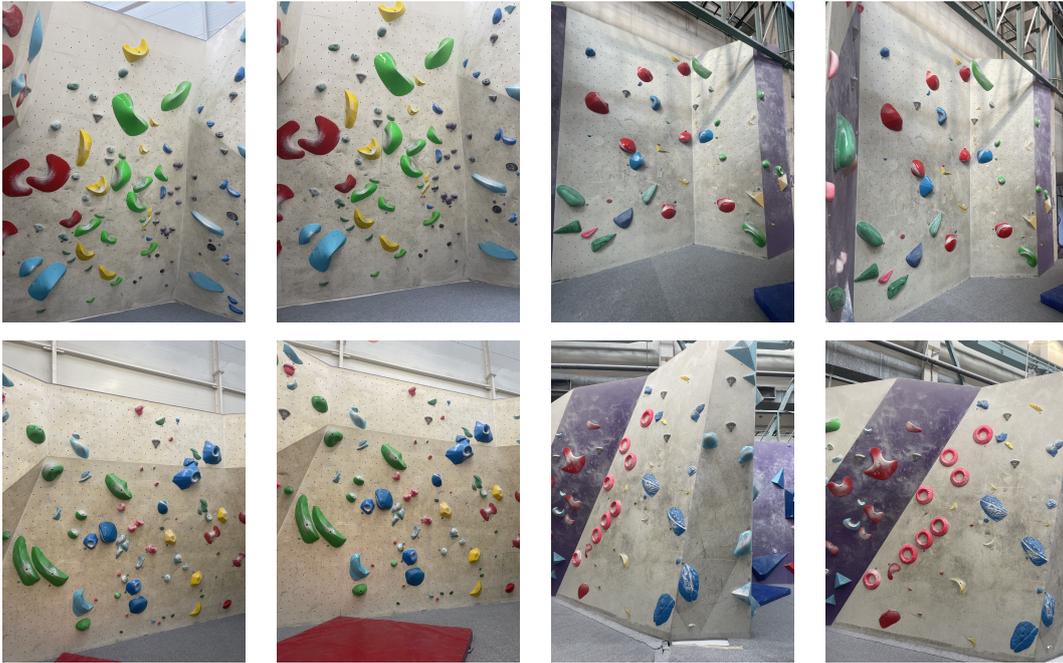


Figure 3.2: Example image pairs from the collected dataset.

the resulting 3D point-cloud (making it denser). On the other hand, with smaller baselines, the reconstruction error increases, as data triangulation performs better with bigger angle between the two camera positions (a greater baseline). We will discuss these effects further in the experiments section 5. Additional data captured for a subgroup of the dataset includes real world measurement of distance between parts of the scene (either on a single plane or at a angle between objects). This data was used to evaluate the accuracy of the 3D reconstruction in a quantitative way. It enables to asses if any distortion occurred (as will be discussed in the results). This dataset can be used for any SfM processes as it possesses the above described qualities which enable the use of methods of SfM with minimal error.

■ 3.1.2 Camera calibration

As was briefly mentioned in the previous section, we calibrated the camera which is used to capture the dataset. This is a crucial step in ensuring the quality of outputs of SfM methods. A calibration collects parameters of 3D to 2D projection, influenced partially by the radial or tangential distortions in the camera lens, and produces the camera matrix K . We can use the camera matrix to compensate for distortion and create more geometrically accurate images (can be done retrospectively).

The camera matrix K (shown in fig. 3.3) is a 3×3 matrix that holds values f_x, f_y that describe the focal length in direction of x, y . The value

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad \text{Example } K = \begin{pmatrix} 3110.29 & 0 & 2048.40 \\ 0 & 3109.29 & 1473.52 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 3.3: Camera matrix K and its example from a iPhone camera.

s describes whether the axes x and y are perpendicular to each other and so in practice it is often 0. The values c_x , c_y define the principal point of the camera, which is the optical center where the optical axis intersects the image plane.

In practice, calibration is a step that is required only once and can for example be realized through capturing a series of images of a chess board from different angles and using code to calculate the camera matrix. This is the method we chose, using the calibration toolbox provided in Open-CV [19], but there are other methods to estimate the camera matrix. Even better, some devices may already come calibrated and have a internally stored camera matrix that can be used. Contrastly, if there is not a chance to calibrate the camera, we could estimate it using the knowledge about the values, however this will always impact result in some way.

After applying a camera matrix to the image, we have the unaltered image (to the extent of the quality of the calibration). This establishes the ground truth for the image and we can start using SfM processes without much error.

3.1.3 Correspondence matching

Correspondence matching is the process of finding the same real object in both images, even though they appear in a different place and from a different angle (Since the two images are captured from a different angle). This establishes a relation between the images and is further on used for finding the Fundamental matrix, triangulation etc.

Multiple approaches are possible to find the correspondence points, and while traditional algorithms implemented in the Open-CV library [19] provide decent results, using a pre-trained model LoFTR [18] implemented in the Kornia [8] library yields more detected correspondence points while not causing a big performance penalty. As is apparent from figure 3.4 and 3.5, LoFTR yields matches in the thousands while SIFT manages only hundreds (after outlier elimination was performed on both to only count relevant matches).

As the authors of LoFTR explain, the differentiating factor is that LoFTR first performs matching on a pixel-wise level instead of sequentially performing image feature detection, description, and matching. Using a Transformer and the described approach gives the advantage mainly across low texture areas (where traditional methods are more likely to fail) [18].

The quality of correspondences can make or break the final result, and since we can not assume results returned by LoFTR are correct, we use RANSAC to filter out some of the bad matches (specifically we use MAGSAC [3] - a RANSAC variant without user set thresholds). This internally requires the

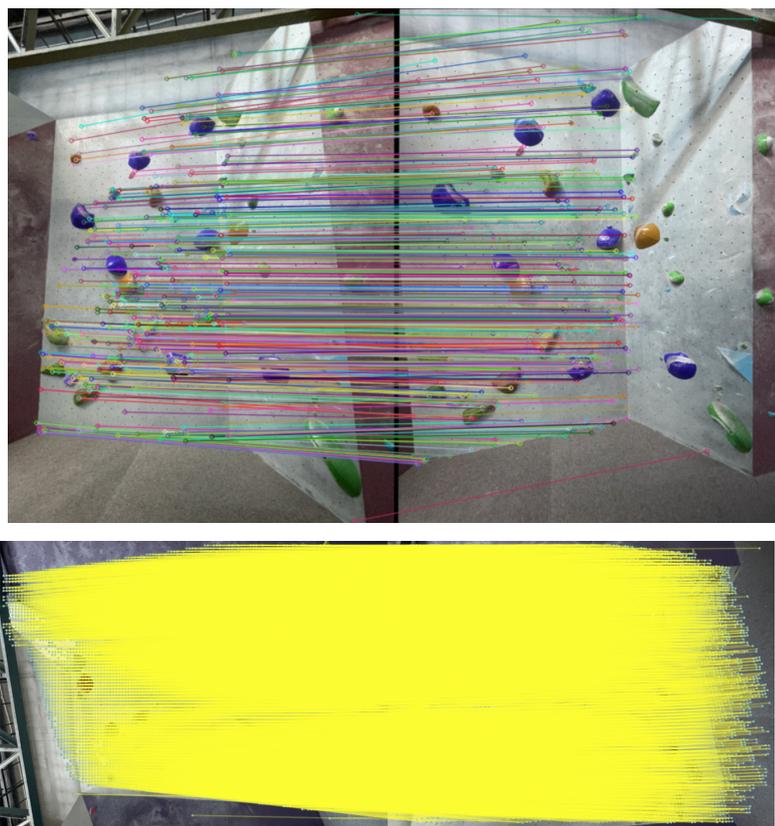


Figure 3.4: Comparison between results of SIFT and LoFTR correspondence matching. See that LoFTR (bottom) returns much denser matches between points.

estimation of the Fundamental matrix using the 7 point algorithm (discussed in a following section).

■ 3.1.4 Epipolar Geometry

Epipolar geometry describes the relationship between both images, the estimated camera positions and the real world object being captured. This mathematical definition of the two different image views allows us to convert the image points between one-another, and is encapsulated in the Fundamental matrix (often denoted F). The Fundamental Matrix relates points in one image to epipolar lines onto which they re-project in the other image. Once we have the correspondences and Fundamental matrix, we can gather more information about Rotation and Translation between the two camera position. Consult the figure 3.6 for better visualization of the relationship of the images.

Calculation of the Fundamental matrix is a algorithmic problem with different approaches available, the basic one being the 7 and 8 point algorithms, which use a system of linear equations to solve for F (the fundamental matrix).

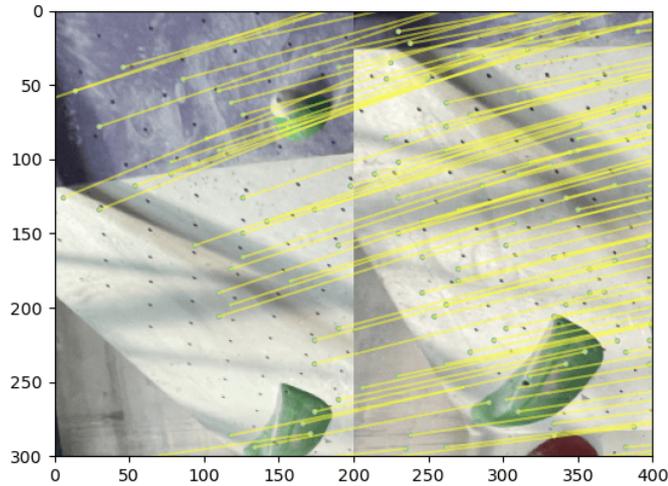


Figure 3.5: Detail of LoFTR matches (every tenth match drawn for clearness) on a small section of the image.

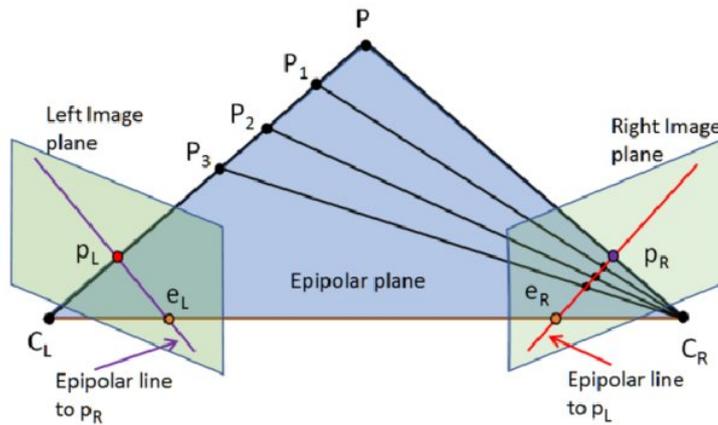


Figure 3.6: Epipolar geometry [6].

Although there are differences between the algorithms, if no significant noise is present in the input points, the solution does not differ. In our process we use the 7 point algorithm and paired it with RANSAC to filter out outlier correspondences (mismatches between unrelated points).

$$\mathbf{x}'^T \mathbf{F} \mathbf{x} = \begin{pmatrix} x' & y' & 1 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$$

Figure 3.7: Fundamental matrix problem.

3.1.5 Rectifying images

Once we have the calibration of our two cameras and information on how points in a image correspond to epipolar lines in the other, we can move on to image rectification, i.e. transform the images so that corresponding epipolar lines become parallel between the images and horizontal (As seen if fig. 3.8). This is achieved by applying a homography matrix to each image.

In fig. 3.8 notice as well, that new dimensions have to be computed for the images to accommodate the warped dimensions which can increase the image size as well as changing its rotation.

The reason for rectifications are, that it improves subsequent steps by making calculations more efficient. It also is a prerequisite for dense stereo matching (growing the amount of corresponding points algorithmically), which is one of the next steps towards 3D reconstruction.

More precisely, in rectification we warp images I_L and I_R using H_1 and H_2 which are computed using the Open-CV [19] implementation of a rectification algorithm for uncalibrated cameras [11]

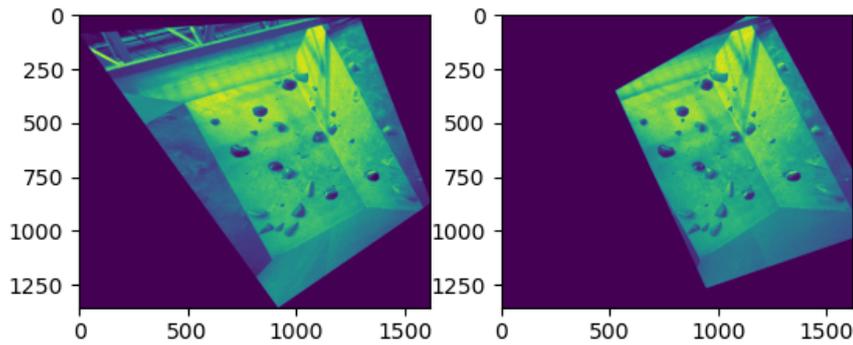


Figure 3.8: Rectified images.

3.1.6 Dense stereo matching

While we use LoFTR to get initial correspondences between images and these would allow us to complete the 3D reconstruction process, the result would be a really sparse 3D point-cloud (we can only get 3D points for known correspondences). To overcome this issue we can employ dense stereo matching, which takes our initial matches as seeds and grows them algorithmically to many more points.

There are many algorithms that provide dense or semi-dense stereo matching and the one chosen is GCS [5], which is a semi-dense stereo algorithm that stays efficient by visiting only a fraction of the disparity space and that guarantees the quality of the new correspondence points. With this we can turn our thousands of correspondences to hundreds of thousands (assuming a 4K image where theoretical limit is in the millions).

If we express all the matches in terms of disparity (enabled by having rectified images), we can start deriving the depth, which becomes the basis for 3D

reconstruction. Disparity represents the horizontal shift between the position of a point in the left image and its corresponding point in the right image. Mathematically, each point from one image is shifted in the other image (specifically its x coordinate) using $x' = x - d$. Mapping a color spectrum onto the disparity we get a visual representation that starts to resemble depth (fig. 3.9)

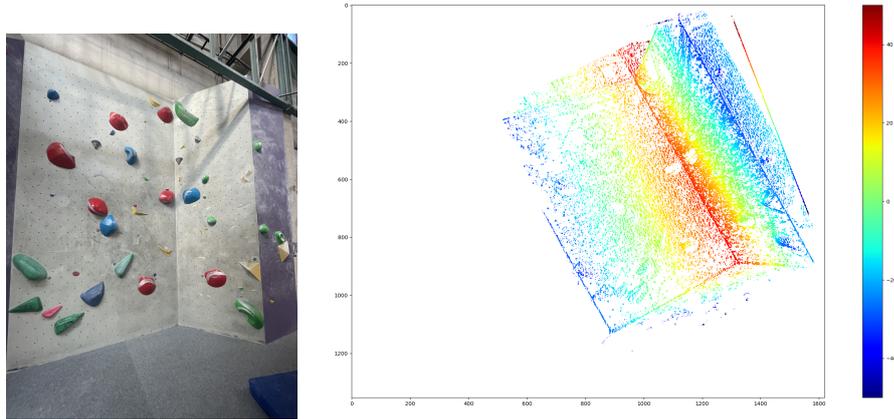


Figure 3.9: Disparity map (note that many values are present but not visible).

■ 3.1.7 Interpolating disparity

Since the dense stereo reconstruction still does not result in dense enough points but it already has many data points covering the whole image, we can use linear interpolation (we can assume many parts of the scene are partially planar and also that from our GCS correspondence graphing, details like the holds have multiple) to fill in the whole image area and improve our basis for point-cloud reconstruction. Linear interpolation could be replaced by another method better reconstructing non planar shapes, but specifically for boulders, most of the scene is partially planar and thanks to running the dense stereo algorithm prior to interpolation, we have enough correspondence points on most non-planar objects to have a close to truth reconstruction.

As can be seen in figure 3.10, the linear interpolation manages to accurately estimate disparities of unknown points based on sampling the value from a triangle drawn between three of the closest known points. This is a good state in which we can start converting into 3D.

■ 3.1.8 Directly relating images (projection)

As described above, the Fundamental matrix F holds some information about the relationship between points in our two images. However, if we want to convert points directly between images, we need to obtain so called projection matrices (P_1 and P_2).

To obtain P_1 and P_2 we need to first get Rotation R and a translation vector

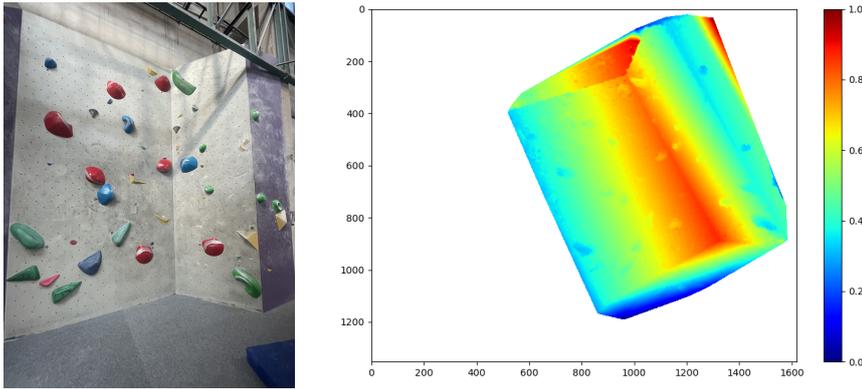


Figure 3.10: Interpolated disparity.

t , which directly show the relation between images (how much did the camera move and rotate between capturing the images). We get R and t from the essential matrix E , which is derived using the Fundamental matrix combined with the Camera matrix (see fig. 3.11 for detail). As can be seen in 3.11, After performing SVD on E , we get multiple possible rotations R_1 , R_2 and translation vectors t_1 , t_2 . We can however test to see which one of these is the correct Rotation and translation by using both for triangulation and finding the combination where most 3D points are in front of both cameras (i.e., the depth values are positive for both views) [12].

Calculating Projection (formulas shown in fig. 3.12) for each matrix once uses the found values R and t and for the other projection we provide a Identity matrix 3x3 and a zero vector (projections are relative and one camera is the start position so no rotation and translation occurs). Having the projection matrices P_1 , P_2 ready, we can move on to triangulating the point-cloud using the disparity as well.

■ 3.1.9 Triangulation into 3D

We use Triangulation to convert 2D corresponding points to 3D, computing the structure in the process. The implementation is based on DLT [10]. DLT: Direct Linear Transform is a method of computing the 3D information using a set of linear equations. It also takes steps to address some geometric error and does not fail when some noise is introduced in the points (However the projection matrices have to be correct).

This approach is taken because we assume enough points were found by the dense stereo algorithm in combination with the scenes being partially planar, that a linear interpolation is a good choice for filling the unknown areas.

The output of this triangulation is a list of 3D coordinates, providing a dense point-cloud for visualization.

Essential matrix from F , K_1 and K_2
 ($K_1 = K_2$ if images were taken with the same camera):

$$\mathbf{E} = \mathbf{K}_2^T \mathbf{F} \mathbf{K}_1$$

Singular Value Decomposition (SVD) of \mathbf{E} :

$$\mathbf{E} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$$

Enforce structure of E by modifying Σ to :

$$\mathbf{\Sigma} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Defining matrix \mathbf{W} to retrieve possible rotations:

$$\mathbf{W} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Possible rotations:

$$\begin{aligned} \mathbf{R}_1 &= \mathbf{U} \mathbf{W} \mathbf{V}^T \\ \mathbf{R}_2 &= \mathbf{U} \mathbf{W}^T \mathbf{V}^T \end{aligned}$$

Possible translations:

$$\begin{aligned} \mathbf{t}_1 &= \mathbf{u}_3 \\ \mathbf{t}_2 &= -\mathbf{u}_3 \end{aligned}$$

Figure 3.11: Summary of calculations needed to get possible R and t .

■ Relation between 2D and 3D points

Since we have the original 2D coordinates as well as the new 3D coordinates, we can directly back-project these points, which is helpful not only for assigning the original image colors to the point-cloud but is also used in user interaction with the point-cloud that enables measurement of distances and angles between planes (to be discussed later)

■ 3.2 Getting measurements from 3D data

As the goal of this work is not only to visualize the reconstruction of 3D models but also to provide measurements of distance and angles between planes in the image, we need to describe how we use the resulting 3D point-cloud to

$$\mathbf{P}_1 = \mathbf{K} \begin{bmatrix} \mathbf{I}_3 & | & \mathbf{0} \\ \mathbf{R} & | & \mathbf{t} \end{bmatrix}$$

$$\mathbf{P}_2 = \mathbf{K} \begin{bmatrix} \mathbf{I}_3 & | & \mathbf{0} \\ \mathbf{R} & | & \mathbf{t} \end{bmatrix}$$

Figure 3.12: Camera projection matrices P_1, P_2 .

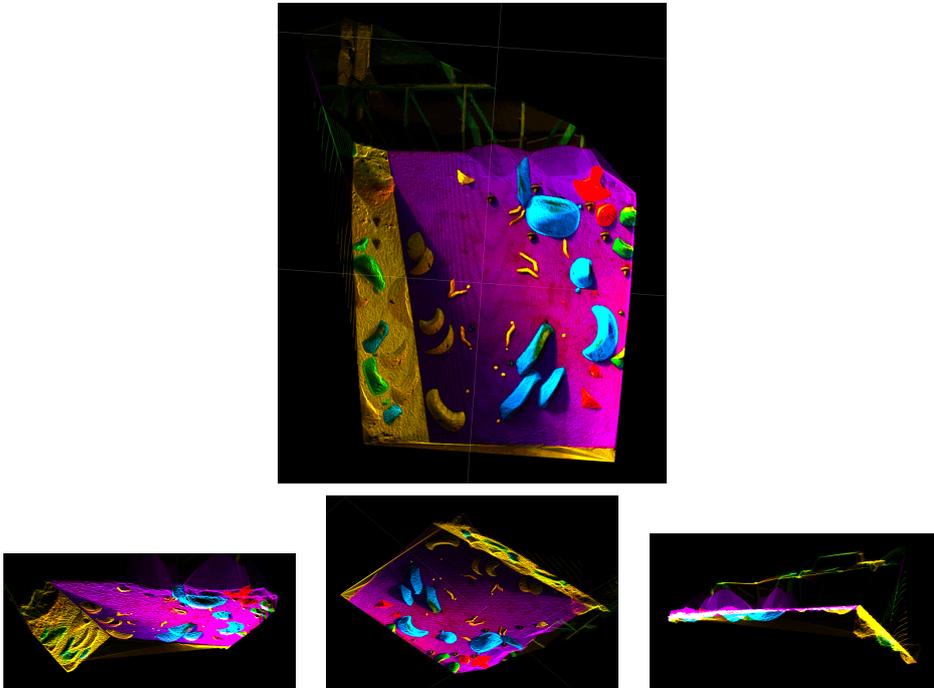


Figure 3.13: Example of reconstructed 3D points from different views.

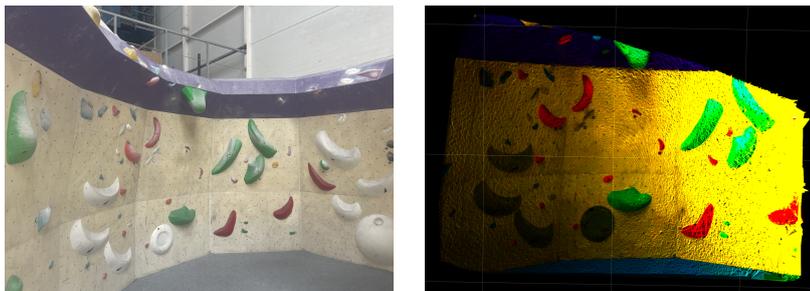


Figure 3.14: Original image and a reconstructed model coloured by it.

get this data.

The reconstruction is metric, but up to scale, which is to be set by a measurement in the real world. This is a limitation of the work that is however very hard to overcome without requiring placing a scaling target in the image (any item of known dimensions) or without using other specialized measuring or additional data (a potential solution is described in Limitations).

■ 3.2.1 Measuring real world distance

After reconstructing the point cloud, we can measure the distance between points using the L2 norm. However this presents the issue of real world scale. We only can measure in relative units without retrieving scale information from the scene or in another way. In this work, user defined scale was used, where a user inputs a distance between any two points which gets used for scale for further measurements. This could cause some error, especially for measurements taken far away from the reference distance.

■ 3.2.2 Measuring angles in the scene

Measuring angles is a more complex task than measuring distance, since we can not rely on single points but have to get planes between which to measure the angle.

However for angle measurement, no reference measurement is required for the process.

■ Plane fitting

Getting a plane on which multiple points lie can be done by using the RANSAC algorithm to find a series of inlier points (ones that all lie on the same plane) and then retrieving the plane model for this plane. The more points are used as input for RANSAC the better the accuracy but as few as 3 points can work as well.

In practice we collect points that mark a area, and RANSAC is then ran on all the points contained in that area. For a more detailed explanation see 4.2.3

■ Angle measurement

Once a plane is fitted, we can get its normal vector. The angle between two planes then is calculated as the arc cosine of the dot product of two of these normal vectors (normalized to the same size).

Chapter 4

Implementation details

After experimenting with the pipeline design and settling on the solution described above, other tasks were undertaken on the way to completion of this thesis. The ones that will be better discussed here are creating a python wrapper for a C++ library and creating a GUI desktop application to present result of reconstruction to any end user.

4.1 Wrapping a library for use in python

Since libraries already used in the project: Open CV [19] and Kornia [8] both don't provide a dense stereo algorithm with good performance, GCS was selected as a better tool, however the library was originally written in C++ and with a Matlab wrapper, using MEX. This meant a new wrapper was needed to be able to run the library directly from Python, which is the primary language used for this thesis and also the GUI application.

Small changes to the C++ source were necessary to remove functions calling back to the Matlab runtime. Instead C++ native alternatives were written and then the library could be compiled standardly as a shared library (.dll or .dylib based on the platform).

To be able to interact with this or any other library, a python native ctypeslib was used. This Python library provides functions to load a library and its contents, and define data types, allocate memory and call functions. An important detail which had to be taken into account was memory order, as numpy arrays which hold most of the data that had to be passed to the C++ library usually store array in C order (row-major), however the C++ library was written for use with Matlab, so it uses a column-major order. This meant changing the memory order as well as allocating a continuous block that fit the expectation of the C++ code.

After resolving some issues, the wrapper is a fully functional alternative to the Matlab interface.

Fig. 4.1 shows some of the code needed to interface with a library using python. This code could theoretically interface with code written in many different programming languages, if they are compiled as a library (see example of compilation in fig. 4.2)

```

1 def allocate_c_arr(arr, c_type):
2     """Function to allocate python arrays for use in C++"""
3     c_arr = arr.ctypes.data_as(POINTER(c_type))
4     #Arr dimensions also have to be passed
5     c_m, c_n = c_int(arr.shape[0]), c_int(arr.shape[1])
6     return c_arr, c_m, c_n
7
8 #Loading the compiled C++ library
9 gcs = cdll.LoadLibrary("dgrow.dylib")
10
11 #Defining all input data types of the C++ function
12 gcs.pyInterface.argtypes = [POINTER(c_double), c_int, c_int,
13                             POINTER(c_double), c_int, c_int,
14                             POINTER(c_double), c_int, c_int,
15                             POINTER(c_double), c_int, c_int,
16                             POINTER(c_double), c_int, c_int,
17                             POINTER(c_double), c_int, c_int,
18                             POINTER(c_double), c_int, c_int,
19                             c_int,c_int,c_uint,c_double,c_double,
20                             c_double,c_double,c_bool,c_bool,c_int,
21                             POINTER(c_double), c_int, c_int,
22                             POINTER(c_double), c_int, c_int,
23                             POINTER(c_double), c_int, c_int]
24 #Defining return type
25 gcs.pyInterface.restype = c_void_p
26
27 #Converting array c_D that was changed inside the C++ function back to numpy form
28 ret_D = np.ctypeslib.as_array(c_D, shape=(c_D_n.value,c_D_m.value)).T

```

Figure 4.1: Code snippets from GCS Python interface.

4.2 Creating a GUI application

There are many choices when it comes to implementing a GUI for a python application. For this thesis, the Qt library with its Python interface PyQt [7] was chosen for good documentation, performance and customizability.

For plotting 3D and 2D data and creating the widgets for measurements, the library PyQtGraph [4] was chosen for having a shared Qt core and providing performant 3D plots (crucial for 3D visualizations).

4.2.1 Data selection

On the launch of the application, the user is free to upload any image pair to try 3D reconstruction on. The app interfaces with the OS to load images from a selected location and keeps a reference to them for future processing. The other element required from the user is the Camera matrix of the device which captured the uploaded image pair. Only after providing this information does the application let the user to continue with the reconstruction.

```

1 g++ -c dgrow0.cpp
2 g++ -dynamiclib -fPIC -o dgrow.dylib dgrow.o

```

Figure 4.2: Compilation of C++ to create a library to interface with.

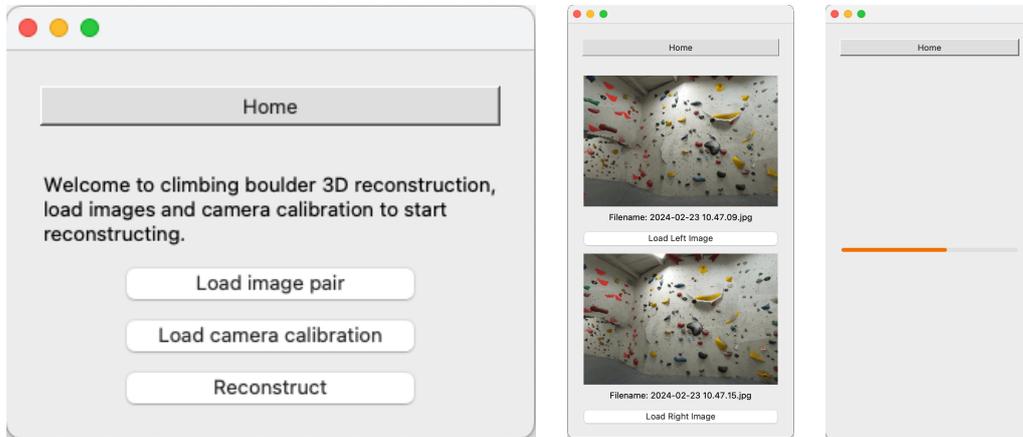


Figure 4.3: GUI app home window, image loading and processing views.

■ 4.2.2 Processing

After all the data is present, the application launches a separate thread which computes all the steps in the pipeline described above, until all results are ready and stored. While waiting, the app displays a loading bar with the progress of the reconstruction (as shown in fig. 4.3).

■ 4.2.3 Result Visualization

After processing is done, the user can choose between visualizing the 3D point-cloud itself or launching one of the measurement widgets that let him interactively measure in the scene to his liking (where data is present). See figure 4.4 to see all the widgets and a menu presented to the user to launch them.

■ Point-cloud visualization

Even though the point-cloud mainly serves as underlying data for the measurements, it can be interesting for the user to visualize it. This is enabled by a interactive widget that can zoom, rotate and move around the points in 3D space.

The points get displayed together with the colours sourced from the images so the user can easily see and evaluate the quality of the reconstruction.

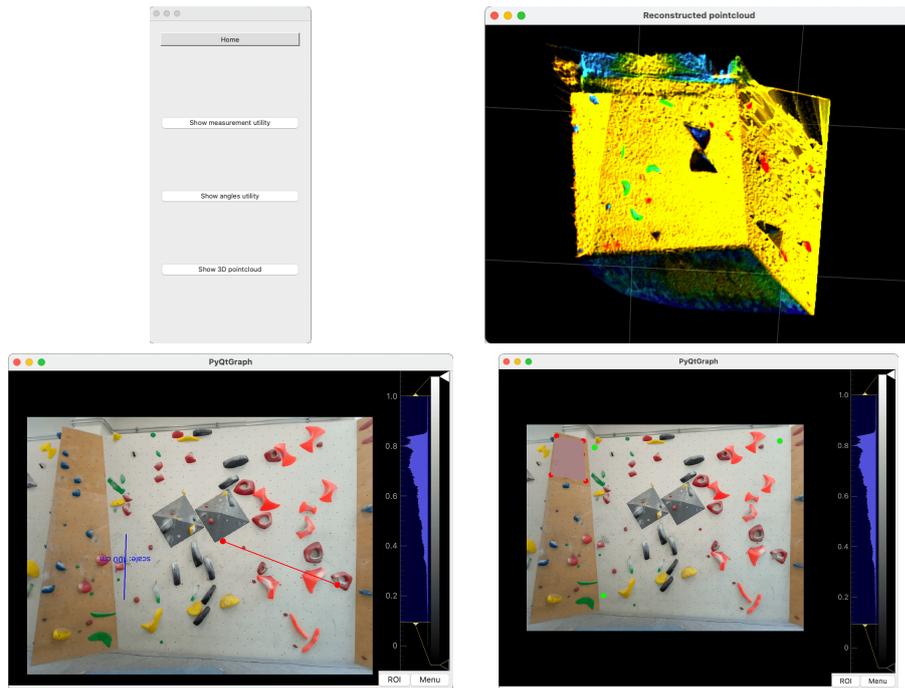


Figure 4.4: Visualization widgets and the related menu.

Distance measurement widget

When the user chooses to measure distances, he is presented with one of the images he uploaded and has to first tap two points and enter the distance between them, setting a scale for further measurements. After that, any points he selects get a line drawn between them and the distance gets displayed for the line.

See figure 4.5 to understand the scaling of measurements. Fundamentally, after the user submits the real world distance between select points, a relation between the real world distance in centimeters and the measurement produced by the norm between the two points (arbitrary units). We use this relationship to convert all other measurements from the arbitrary units to centimeters.

The measurement works with the 3D model and not purely image data, so the measurement should correspond with the real world structure of the boulder. The ability to measure in the 3D structure is why reconstruction is needed in the first place. If the goal was to measure only points lying on a single plane, 2D data would be needed. Having the 3D data means we can overcome this limitation and measure across points with different depth etc.

Angle measurement widget

Angle measuring requires the user to click 5 points in the image he gets presented with that should all be in one plane. Then he defines the plane to measure against by another 5 points and when this is complete, he gets a

```

1 def measureDistance(self):
2     """Measure scaled distance between self.point1 and self.point2
3     (QT points clicked by user)"""
4
5     xy1 = np.array([self.point1.x(),self.point1.y()],dtype=np.float32)
6     xy2 = np.array([self.point2.x(),self.point2.y()],dtype=np.float32)
7     distance_img = np.linalg.norm(xy2-xy1)
8
9     # Find nearest point to clicked position in 3D
10    xyz1 = self.pts_3D[self.findClosestPoint(xy1)]
11    xyz2 = self.pts_3D[self.findClosestPoint(xy2)]
12
13    # Calculate cm distance based on real_world_scale
14    distance_real = np.linalg.norm(xyz2-xyz1) * self.real_world_scale
15    return distance_real
16
17 def extractScale(self):
18     """Set real_world_scale - a constant to convert distance to cm
19     based on user given value (through InputDialog)"""
20
21    dist = self.measureDistance(show=False)
22    self.scale_cm = self.scaleInputDialog()
23    self.real_world_scale = (self.scale_cm/dist)
24

```

Figure 4.5: Snippets of Python used to scale distance measurements.

measurement returned. All the points selected are marked in the image. Internally, each of the 5 points is used to calculate a convex hull which is then used to find all 2D points inside it and their corresponding 3D points. RANSAC is then used on the 3D coordinates to find inlier points in the marked plane and after that, a plane model can be found, giving us the planes normal vector which is used to measure the angles in the scene (as is shown in fig. 4.6).

In the thesis we use the scipy library [21] implementation of the convex hull, which is the smallest convex polygon that can enclose all the points in a set (i.e. the 5 points defined by the user), because it provides good methods to then get all points inside of this polygon. After that, the RANSACRegressor class from sklearn [15] was used to identify the plane (fit inlier points form the given area).

The angle θ between two planes is:

$$\theta = \arccos\left(\frac{\mathbf{n}_1 \cdot \mathbf{n}_2}{\|\mathbf{n}_1\| \|\mathbf{n}_2\|}\right)$$

where:

$$\mathbf{n}_1 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}, \quad \mathbf{n}_2 = \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}$$

The magnitudes of the normal vectors are:

$$\|\mathbf{n}_1\| = \sqrt{a_1^2 + b_1^2 + c_1^2}, \quad \|\mathbf{n}_2\| = \sqrt{a_2^2 + b_2^2 + c_2^2}$$

in this case, a_1, b_1, c_1 and a_2, b_2, c_2 are detected using RANSAC.

Figure 4.6: Calculation of the angle between two planes using their normal vectors.

Chapter 5

Experiments

Let us present and evaluate the results of this work. As the target of this thesis is to provide a tool for measuring distances and angles in the captured scene, we will evaluate both of these against the ground truth. As was discussed earlier, to the best of our knowledge, no publicly available datasets exist with boulders and their measurements, that is why measurements were made by the author at the Smichoff climbing gym¹, and angle data was sourced from Hudy boulder Karlín gym², from a simple 3D sketch containing angles of the walls.

To better understand the accuracy achieved by the described 3D reconstruction pipeline, cross-validation was performed on the measurements, and statistics were computed (better presented in fig. 5.1).

5.1 cross-validation method

To better assess our results, a cross-validation script was designed. In a loop over all the known ground truth measurements, one distance is always selected as the reference measurement (used to set the scaling of all other measurements) and the rest of the measurements are evaluated (better shown in fig. 5.2).

This provides more insight into the distortion and error of the 3D data, thanks to the iteration over many measurements.

5.2 Measurement accuracy evaluation

Although capturing accurate measurements by hand of multiple boulders is very tedious, we can use the regularity of some elements to gain more control data without actually recording it. A good example are mounting holes, which not all but many boulders have over their whole surface in a regular pattern, spaced diagonally every 20 cm (as seen in fig. 5.3). Another regularity was found on a specific boulder where panels making up the wall

¹Smichoff climbing gym website: <https://www.lezeckecentrum.cz/cs/>

²Hudy climbing gym website: <https://www.hudysteny.cz/boulderkarlin/>

$$\begin{aligned} \text{Mean Error (ME)} : \quad ME &= \frac{1}{N} \sum_{i=1}^N (d_i - m_i) \\ \text{Relative Mean Error} : \quad RME &= \frac{1}{N} \sum_{i=1}^N \left(\frac{d_i - m_i}{m_i} \right) \times 100 \\ \text{Mean Absolute Error} : \quad MAE &= \frac{1}{N} \sum_{i=1}^N |d_i - m_i| \\ \text{Relative Mean Absolute Error} : \quad RMAE &= \frac{1}{N} \sum_{i=1}^N \left| \frac{d_i - m_i}{m_i} \right| \times 100 \\ \text{Root Mean Square Error} : \quad RMSE &= \sqrt{\frac{1}{N} \sum_{i=1}^N (d_i - m_i)^2} \end{aligned}$$

Figure 5.1: Statistical formulas used to evaluate results. Here, d_i represents the estimated distances, m_i represents the ground truth distances, N is the number of data points, and \bar{d} is the mean of the data points. For each of these statistics we show the mean value and standard deviation over the cross-validation bins.

were squares where each side measures 100 cm.

In total, we tested 10 reconstructions, 7 for distance measurements and 3 for angle measurements.

After manually marking some of these regular points and adding the authors measurements (see both in fig. 5.4), we ran cross-validation on the distances, selecting a different pair of points each time as scale and we captured the following results (in figures 5.5, 5.6).

■ Bias in measurement

One of the metrics we chose to evaluate is Mean error (and relative mean error), showing us if perhaps all the measurements are biased towards being always longer or shorter than the ground truth. This would point to some issues in the reconstruction or measurement evaluation, however for the test cases we observed that the Mean error has a small mean close to zero and only the standard deviation is larger. See an example of the observed behaviour in fig. 5.7, computed for boulders shown in fig. 5.5

■ 5.2.1 Measuring only using regularities

Because manually collecting measurements is problematic (public gyms are usually filled with people wanting to climb so measuring the boulder is tricky), and can't provide good coverage of the boulder (the height of the wall can

reach 4 or 5 meters, so most is not reachable for manual measurements), and measuring with a hand held tape measure over longer distances with an angle is prone to errors, we used the found regularities to enlarge our evaluation set and annotated measurements on boulder where no manually collected data exists.

This restricts all the measurements to lie on physical planes, however it still needs to utilize 3D underlying data to provide results, so we can evaluate the reconstruction accuracy using these measurements as well. Specifically, the 3D data is needed to be able to set one reference measurement in any part of the scene and then use it over all the different planes. For measuring on a single plane, no 3D reconstruction would theoretically be needed.

Measurements and the statistical results compared to the ground truth (calculated based on the known distance between mounting holes) are shown in figures 5.8, 5.9 and 5.10.

■ 5.2.2 How distance from reference measurement impacts results

Even though steps to mitigate distortion are taken in the reconstruction pipeline, some will still be present in the final result. And while some error may be similar among the whole image, some distortion may more affect the edges of the image (because of radial distortion if the image was captured by a wide angle lens for example).

To evaluate if this is the case and by how much, a boulder with a long flat plane was reconstructed and measurements were made of the same distance along its length (see fig. 5.11). The result was then evaluated as all the others with an additional focus on the distance from where the scale was set (shown in fig. 5.12).

■ 5.3 Angle measurement evaluation

To evaluate the accuracy of the angles in the scene, we used a schematic of Hudy boulder Karlín, which has sketches of individual walls with all angles of the boulder sections marked (as difference from a 90 degree angle. see in fig. 5.13).

This ground-truth data allows for measurement against the floor plane, and for boulders with multiple angled sections, measurements of angle between each part.

Although the selected method for plane fitting is dependent on user input, if a big enough portion of the plane is selected, the detected normal vector is always the same, because RANSAC finds the plane based on inliers in the selected area. This leads us to evaluating only by using a single measurement of each angle (as seen in fig. 5.14).

Angle measuring results are shown in figures 5.15, 5.16 and 5.17. Figure 5.14 shows multiple measurements that were taken between wall segments of boulder in fig. 5.15.

■ 5.4 3D point cloud evaluation

Since there is no dataset available for 3D scans of boulders, and the climbing gyms we reached out to also could not provide it, we do not have a statistical evaluation of the accuracy of the 3D points compared to the real world 3D structures of the boulder.

A dataset could however be created in the future by using a 3D scanner (even some phones now come with lidar scanners which can capture a 3D scene like in fig. 6.1), and then this accuracy could be evaluated. Without the dataset we can still observe some of the qualities of the reconstruction visually. Mainly we can see the effect of the interpolation and the "flatness" of the boulder walls depending on their angle towards the camera (portion of the boulder parallel to the camera is flat, whereas tilted planes in the boulder have a more uneven surface) in fig. 5.18 (shown without texture for better evaluation).

■ 5.5 Expansion on results

After evaluating the results of the measurements we observe that the results are not completely accurate. Measurement error differs a lot between different test scenes (different image reconstructions). This section aims to discuss some of the reasons this may be occurring, however the main takeaways are that our evaluation techniques do not point to a single culprit and that many factors all influence the results.

■ 5.5.1 Dataset quality

During evaluation of the results, one thing becomes apparent. The error for each case is quite different. This could be the result of many different factors but one thing that is probably at play is the difference in the image pair quality.

Generally, we know that every reconstruction is affected by a combination of baseline between images, calibration quality and resolution of images. The images sharpness (focus), lighting and the scene captured can affect quality. Although sharpness is controlled in the dataset, lighting differs between images and the scene captured is always different. The main factors affecting the result from the scene point of view are: what is the angle to the subject (the boulder), the distance to the boulder (which also affects if the ground is

captured) and more.

■ 5.5.2 Reference measurement quality

As the manual measurements were performed by hand with a tape measure, the recorded distance may be off a varying amount for each measurement and each boulder. This could be improved by using a different measuring method or having a dataset with measurements and angles in it.

Another factor that is important to consider is that the shorter the distances we try to measure the greater our error will be. The reconstruction process produces quite a good result considering the whole scene, but local errors may be greater, affecting short distance measurements.

■ 5.5.3 Other factors influencing quality

One factor influencing the result quality is the boulder itself. Each wall differs and while some boulders have only a few holds, no mounting holes and few angles, others have many different structures, mounting holes and angles. This all influences the texture that can be detected by feature matchers like LoFTR [18], as well as how dirty or clean the boulder is (for match detection, the more dirty and textured, the better).

```

1 def crossvalidate(self):
2     """function computing all data permutations for cross-validation,
3     part of a class holding all necessary variables
4     (measurements is the ground truth)"""
5     for i in range(len(self.measurements)):
6         dist = self.measureDistance(self.measure_pts[i,0,:],
7                                     self.measure_pts[i,1,:])
8         self.real_world_scale = (self.measurements[i]/dist)
9         for j in range(len(self.measurements)):
10            if j == i:
11                self.results[i][j] = measurements[i]
12                continue
13            self.results[i][j] = self.measureDistance(
14                self.measure_pts[j,0,:],
15                self.measure_pts[j,1,:])
16        self.real_world_scale = 1
17    np.set_printoptions(suppress=True)
18
19 def compute_stats(data):
20     """Calculate statistics about results of experiment (data)
21     compared to ground_truth (measurements)"""
22    np.set_printoptions(suppress=True)
23    signed_relative_error = ((data.T - measurements[:, np.newaxis]) /
24    measurements[:, np.newaxis]) * 100
25    absolute_relative_error = np.abs(signed_relative_error)
26    measurement_stats = {
27        'ME' : np.mean(data.T - measurements[:, np.newaxis], axis=1),
28        'Relative_ME': np.mean(signed_relative_error, axis=1),
29        'MAE': np.mean(np.abs(data.T - measurements[:, np.newaxis]),
30            axis=1),
31        'Relative_MAE': np.mean(absolute_relative_error, axis=1),
32        'std_dev': np.std(data, axis=1),
33        'Root_MSE': np.sqrt(np.mean(
34            (data.T - measurements[:, np.newaxis]) ** 2, axis=1))
35    }
36

```

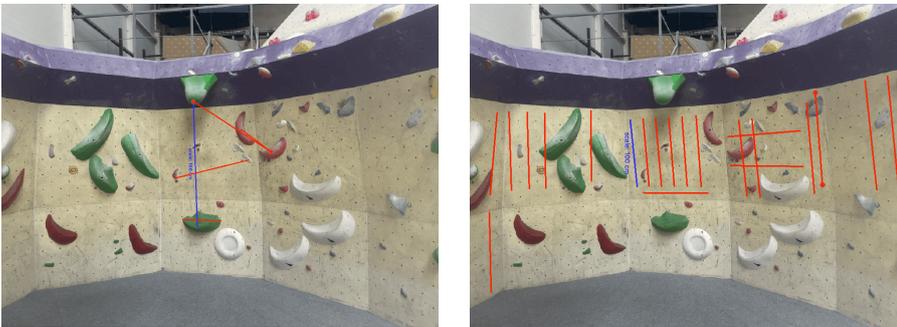
Figure 5.2: Snippets of Python used for cross-validation.



Figure 5.3: Regular elements with known distance. Mounting holes spaced every 20 cm diagonally (L), and square panels of dimensions 100×100 cm (R).



Figure 5.4: Example of measurements by author (L), measurements using regularities (R).



Statistic	Mean value	σ
Mean Error [cm]	1.157	9.550
Relative Mean Error [%]	0.872	9.375
Mean Absolute Error [cm]	10.672	4.046
Relative Mean Absolute Error [%]	10.638	3.498
Root Mean Squared Error [cm]	12.857	4.556

Figure 5.5: Measurements and their statistics for 26 measurements.



Statistic	Mean value	σ
Mean Error [cm]	12.503	40.796
Relative Mean Error [%]	11.176	35.635
Mean Absolute Error [cm]	45.041	20.553
Relative Mean Absolute Error [%]	39.282	17.456
Root Mean Squared Error [cm]	57.355	25.806

Figure 5.6: Measurements and their statistics for 15 measurements. This is one of the fail cases, where probably the lack of texture compared to other cases led to much greater error.

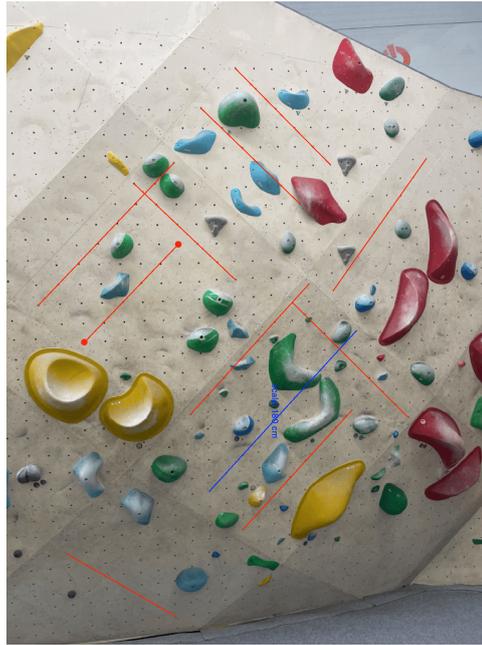
Assessing the bias in error, we look at the non absolute value of relative mean error. Completely non-biased results would make this error be 0. The calculated mean error is very close to zero, therefore we have a almost unbiased result.

Statistic	Value	σ
Relative Mean Error [%]	0.8726	9.3751

The table below shows relative mean error (in %) for each cross-validation bin (after all iterations), from which the overall statistic was computed.

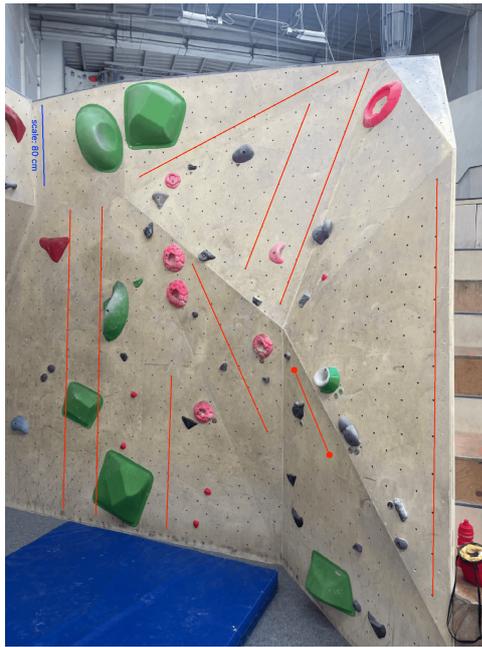
7.91	-0.76	-2.25	-2.75	-2.00
-2.67	-1.02	0.29	3.37	3.34
2.96	-9.46	-6.36	-7.84	-14.42
1.89	4.99	-16.95	11.83	7.98
14.13	17.36	-6.98	-12.84	18.68
14.26				

Figure 5.7: Example of relative mean error, observe that almost no bias is present overall, since the individual measurements cancel out.



Statistic	Mean value	σ
Mean Error [cm]	0.112	2.570
Relative Mean Error [%]	0.041	2.024
Mean Absolute Error [cm]	2.780	1.310
Relative Mean Absolute Error [%]	2.171	0.994
Root Mean Squared Error [cm]	3.451	1.325

Figure 5.8: Measurements and their statistics for 11 measurements.



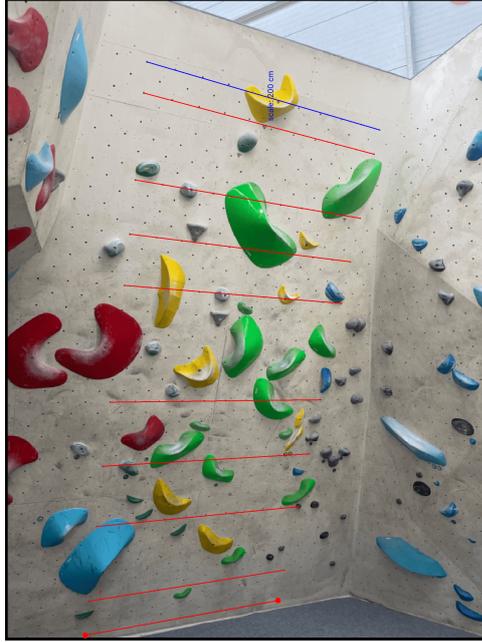
Statistic	Mean value	σ
Mean Error [cm]	-0.708	2.358
Relative Mean Error [%]	0.035	1.885
Mean Absolute Error [cm]	3.317	0.977
Relative Mean Absolute Error [%]	2.007	0.795
Root Mean Squared Error [cm]	4.280	1.281

Figure 5.9: Measurements and their statistics for 10 measurements.



Statistic	Mean value	σ
Mean Error [cm]	-0.335	3.791
Relative Mean Error [%]	0.115	3.309
Mean Absolute Error [cm]	3.550	2.392
Relative Mean Absolute Error [%]	3.345	1.733
Root Mean Squared Error [cm]	4.739	2.160

Figure 5.10: Measurements and their statistics for 10 measurements.



Statistic	Mean value	σ
Mean Error [cm]	0.022	2.096
Relative Mean Error [%]	0.011	1.048
Mean Absolute Error [cm]	2.244	0.915
Relative Mean Absolute Error [%]	1.122	0.457
Root Mean Squared Error [cm]	2.808	0.902

Figure 5.11: Measurements and their statistics for 10 measurements.

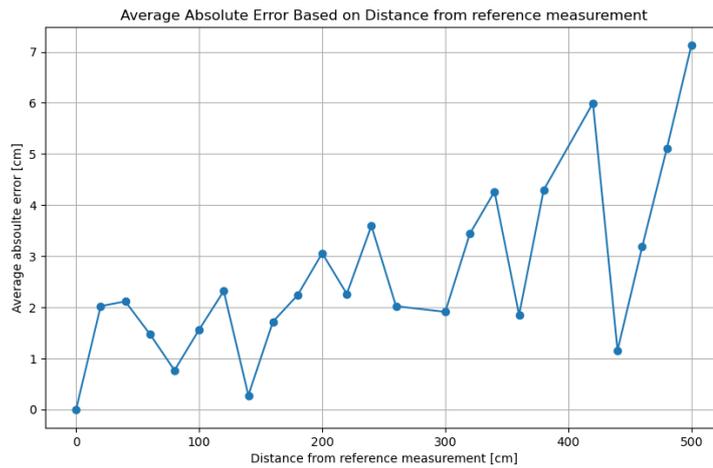
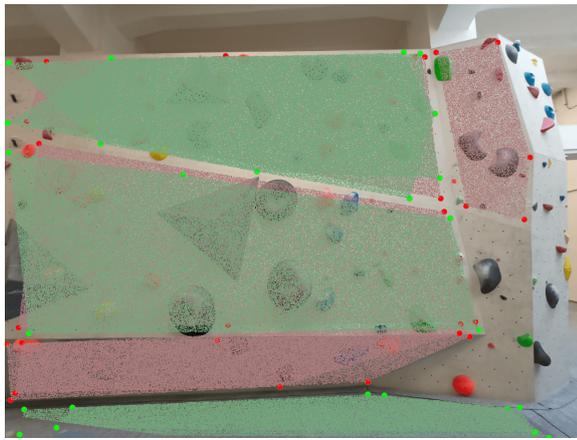
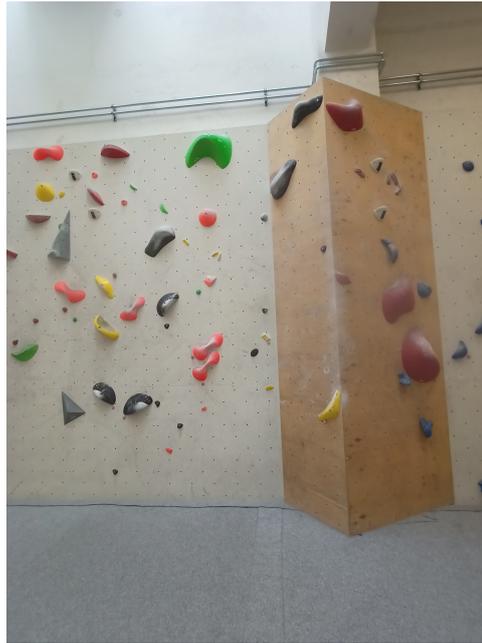


Figure 5.12: Graph of relative error based on distance from reference measurement, showing the increase in error the further the reference.



Statistic	Value
Mean Error [°]	7.031
Relative Mean Error [%]	-2.059
Mean Absolute Error [°]	9.984
Relative Mean Absolute Error [%]	22.616
Standard Deviation [°]	48.298
Root Mean Squared Error [°]	17.987

Figure 5.15: Angle measurement statistics for 5 measurements. The red and blue points mark the selected planes and the area is transparently colored.



Statistic	Mean value
Mean Error [°]	-11.997
Relative Mean Error [%]	-13.330
Mean Absolute Error [°]	11.997
Relative Mean Absolute Error [%]	13.330
Standard Deviation [°]	2.933
Root Mean Squared Error [°]	12.350

Figure 5.16: Angle measurement statistics for 2 measurements.



Statistic	Mean value
Mean Error [°]	-3.238
Relative Mean Error [%]	-3.949
Mean Absolute Error [°]	3.238
Relative Mean Absolute Error [%]	3.949
Root Mean Squared Error [°]	3.238

Figure 5.17: Angle measurement statistics for a single measurements (only angle in the reconstructed pointcloud).

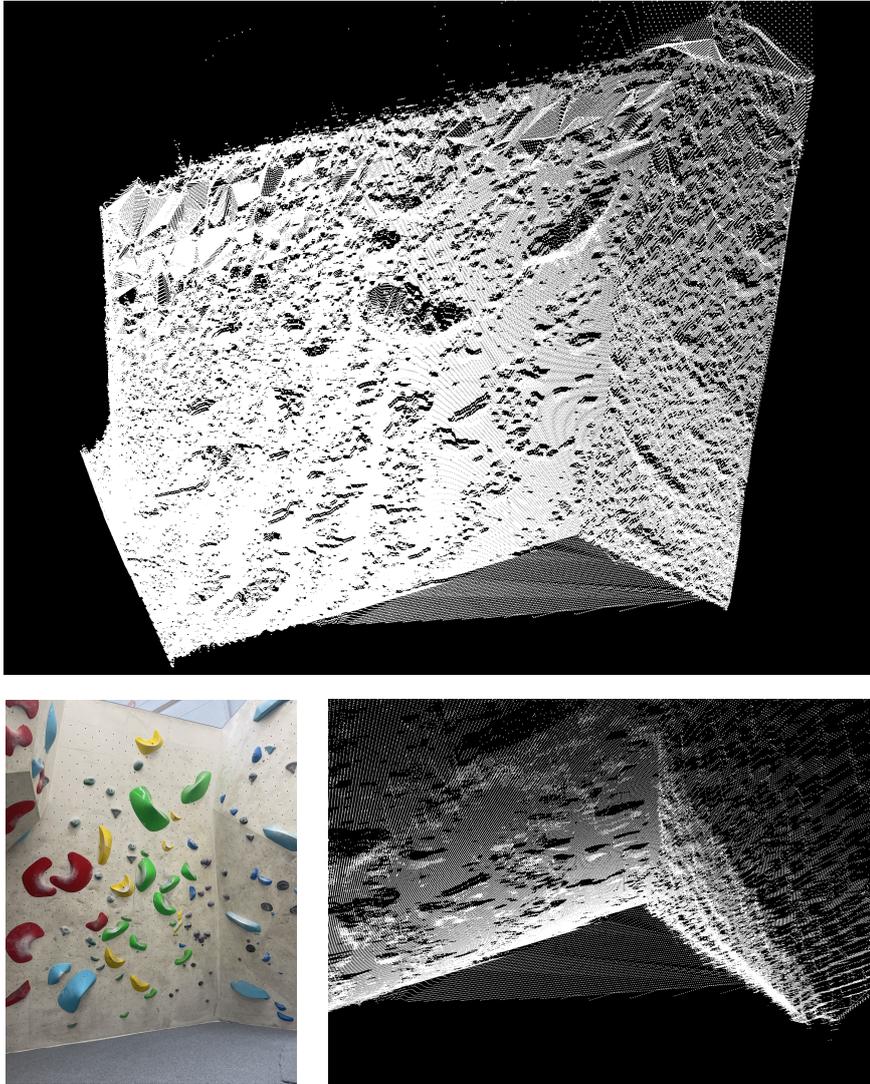


Figure 5.18: A untextured 3D reconstruction and its detail showing flat and uneven regions depending on angle towards camera. The original image is also shown for comparison.

Chapter 6

Limitations and future Work

The primary goal of this thesis was to develop an effective 3D reconstruction pipeline for indoor boulders. While the results achieved demonstrate the feasibility of the approach, several limitations were identified that suggest avenues for future improvement.

Limitations of this work mainly lie in accuracy, where errors of above 10% were recorded for some test cases, pointing towards a generally inconsistent tool (some boulders get reconstructed with a small error while in a couple of cases, a big error was recorded). This is partly due to the limitation of the method (only two images being used, handheld cameras etc.), however with more time and effort performance can probably be improved.

Possible future work includes either trying to address some of the mentioned limitations of this work or extending functionality in other ways.

6.1 Improving accuracy

At this stage, the thesis shows that to some extent, reconstructing from two images is possible, but to improve results, new methods would need to be tried, to end up with a more accurate dense 3D point-cloud. Many alternatives present themselves as possible paths towards potential improvement of the results.

Segmenting the images for example, could help find better correspondences and segments could then be used for improved interpolation. Monodepth algorithms could also be combined with the existing process, providing additional data for improved computation of results.

Another possible approach would be to train a model that would try to replace most of the pipeline, however, here we face the issue of no datasets being available for training (pairs of pictures with a 3D model would be needed). This may be partially resolved by generating a synthetic dataset.

To enable evaluation of these and other methods, better evaluation of the results would be helpful, to help with isolating the source of the error. Our current testing methods evaluated that the reconstruction finishes with some error and we can quantify this error, but we can't find its source from this information. Better ground-truth data would help with this.



Figure 6.1: A Lidar 3D scan of a boulder captured by a iPhone 14 Pro lidar scanner, using the application 3D scanner. The same image is shown with (R) and without (L) texture, to better display the accuracy and spatial textures captured.

■ 6.1.1 Creating datasets

A notable action that could help in improvement of this work and others would be creating a dataset of boulder images ideally paired with their accurate 3D model (sourced from the manufacturer or gathered by a very precise 3D scanner).

One example of better ground truth data can be seen in fig. 6.1, captured with a lidar scanner equipped phone (for example ¹).

■ 6.1.2 Self-calibration

Right now, the program created for reconstruction takes a camera matrix of the capturing device as input. This however could be avoided, in the simplest case just by correctly estimating the matrix based on some image information. Contrastly, a calibration utility could be built into the app, to let users calculate the camera matrix of their device. Methods exist to self-calibrate the camera using the image pair itself (found for example in [14]). The regular mounting holes or detectable lines in the image could be used, however these methods may come with a decrease in accuracy compared to standalone calibration.

■ 6.2 Building a mobile app

To provide better functionality for users, a mobile app could be developed with the pipeline implemented to reconstruct the 3D model directly on the device capturing images.

Right now, the desktop app allows for the examination and measurement of

¹<https://apps.apple.com/us/app/3d-scanner-app/id1419913995>

boulders in between climbing sessions (with previously captured data, the user can reconstruct the boulder he climbed in the gym at home).

However, since one of the potential uses of the measurement application is direct utilization by climbers in a climbing gym, this would greatly improve the user experience for this use case (running a desktop application in a climbing gym is not really feasible).

■ 6.3 Automatic scale detection

Although not a easy task in general, not requiring a user to input scale could be a big factor in improving this work. As we discussed, some of the boulder walls have regularities of known size, so a detection algorithm could be designed to automatically scale measurements on boulders on which such regularities exist.

Alternatively, taking inspiration from the default measuring app on IOS, a mix of gyroscope information about the phone movement and rotation coupled with image data could be used to measure without the need for knowing distance [2]. This method however is prone to huge errors, especially in the indoor use case, where the movement is minimal so the sensors provide low accuracy.

■ 6.4 Inventing new use cases and adding features

The other possible direction that this work can be used as a base for is using 3D reconstructed data for a very different use case than measuring. This might require changing the pipeline or adding extra processing, depending on the use. Some use cases that come to mind are 3D printing and use of the model in virtual reality.



Chapter 7

Conclusion

In this thesis, a 3D reconstruction pipeline was designed, tested and evaluated, to create 3D models (point clouds) of climbing boulders. The input information needed for this process is a pair of images of a indoor boulder and a camera matrix. If the pair of images fulfills some basic requirements discussed in the thesis, it is possible to reconstruct the 3D data. For testing purposes a dataset of such images was captured by the author.

A GUI desktop application was developed for the processing of such images, and it produces 3D point clouds as well as providing an interface to measure in the scene and estimate angles as well. The results of such measurements carry some error but we observed that in most cases the tool is usable even with these errors. However increasing accuracy is a target in the future.

In the process of creating the application, a python wrapper was created for the dense stereo library GCS [5], which can be used in future projects.

Promising results were achieved, and limitations and possible future work was identified. This tool is a good test ground of two image reconstruction and could be improved and extended to become even better.



Bibliography

- [1] Capturing reality: Reality scan.
- [2] Apple Inc. Use the measure app on your iphone, ipad, or ipod touch, 2024. Accessed: 2024-05-22.
- [3] Daniel Barath, Jiri Matas, and Jana Noskova. Magsac: Marginalizing sample consensus. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [4] Luke Campagnola et al. *PyQtGraph: Scientific Graphics and GUI Library for Python*. MIT License, 2009–.
- [5] Jan Čech and Radim Šára. Efficient sampling of disparity space for fast and accurate matching. In *BenCOS 2007: CVPR Workshop Towards Benchmarking Automated Calibration, Orientation and Surface Reconstruction from Images*. IEEE, 2007. Software GCS 2.0.
- [6] Dimo Chotrov. Epipolar geometry of stereo vision, 2018.
- [7] Riverbank Computing. *PyQt*, 1998–.
- [8] D. Ponsa E. Rublee E. Riba, D. Mishkin and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.
- [9] Clément Godard, Oisín Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency, 2017.
- [10] Richard Hartley and Andrew Zisserman. *Structure Computation*, page 310–324. Cambridge University Press, 2 edition.
- [11] Richard I. Hartley. Theory and practice of projective rectification. *International Journal of Computer Vision*, 35(2):115–127, 1999.
- [12] Kornia contributors. *Documentation of: kornia.geometry.epipolar*, 2024.
- [13] Eleni Kouti. Geometric documentation of climbing routes: 3d maps of sport climbing fields. the case study of villanueva de valdegoxia. Master’s thesis, National Technical University of Athens, School of Rural

