Interner Bericht

DLR-IB-FT-BS-2018-256

Fusion Of Lidar And Camera Data For Distance Determination Of Image Features

Hochschulschrift

Neumann, Tobias

Deutsches Zentrum für Luft- und Raumfahrt

Institut für Flugsystemtechnik
Braunschweig

Deutsches Zentrum
DLR für Luft- und Raumfahrt

Institutsbericht
DLR-IB-FT-BS-2018-256

Fusion Of Lidar And Camera Data For Distance Determination Of Image Features

Neumann, Tobias

Institut für Flugsystemtechnik
Braunschweig

060 Seiten
121 Abbildungen
002 Tabellen
041 Referenzen

Deutsches Zentrum für Luft- und Raumfahrt e.V.
Institut für Flugsystemtechnik
Abteilung Unbemannte Luftfahrzeuge

Braunschweig, den 02.02.2024

[redacted]

Institutsdirektor: Prof. Dr.-Ing. S. Levedag

Abteilungsleitung: Johann Dauer

Betreuer:in: Nikolaus Ammann

Verfasser:in: Tobias Neumann

FUSION OF LIDAR AND CAMERA DATA FOR DISTANCE DETERMINATION OF IMAGE FEATURES

FUSION VON LIDAR- UND KAMERADATEN FÜR DIE ENTFERNUNGSBESTIMMUNG VON BILDMERKMALEN

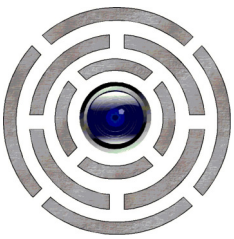
Tobias Neumann
tobias.neumann@stud-mail.uni-wuerzburg.de

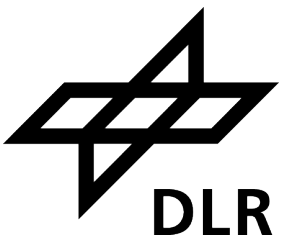
Supervisors:
Prof. Sergio Montenegro[a]
Nikolaus Alexander Ammann, M.Sc.[b]

[a]Julius-Maximinlians-University of Wuerzburg
Chair of Computer Science VIII - Aerospace Information Technology
Am Hubland, D-97074 Wuerzburg

[b]German Aerospace Center (DLR)
Institute of Flight Systems
Lilienthalplatz 7, D-38108 Braunschweig

September 18, 2018





Acknowledgment

This work was done at the DLR Institute of Flight Systems, which provided me with equipment and expertise needed to work up this thesis in that scope. I thank my supervisor, Nikolaus Alexander Ammann, M.Sc., and all the people at the DLR making this collaboration possible. Furthermore, I thank Prof. Sergio Montenegro of the Chair of Computer Science VIII at the Julius-Maximilians-University of Wuerzburg to give me the support for creating this work.

Abstract

The fusion of sensors is a major need in order to create a wider spectrum for environmental perception. Using laser range scanners and cameras, the distance of detected objects of interest within the image can be determined by merging the sensor data properly. On real-time systems like unmanned aerial vehicles, it is important to make simple calculations and keep the merging process cheap in terms of computational time. This paper develops an appropriate fusion method based on ray tracing techniques. The proposed method is rather simple, making it possible to create slim implementations. In order to widen the area of application in which this method can be used, variations of it on different parts are introduced. To verify this method, but also to possibly reveal weaknesses, it is tested in a simulation environment and through experimental tests and analyzes.

Zusammenfassung

Die Fusion von Sensoren ist ein wichtiger Bestandteil für die Erweiterung der Umgebungswahrnehmung. Durch die Nutzung von Laserscanner und Kamera kann die Entfernung eines Objektes im Bild nur bestimmt werden, wenn die Sensordaten richtig zusammengeführt wurden. Auf Echtzeitsystemen, wie beispielsweise unbemannte Luftfahrzeuge, ist es wichtig, die Berechnungen einfach zu halten. Diese Arbeit entwickelt ein entsprechendes Fusionsmodell basierend auf Raytracing. Das vorgestellte Methode ist simpel gehalten, was eine effiziente Implementation ermöglicht. Zusätzlich werden Anpassungsmöglichkeiten an verschiedenen Stellen der Methode für die Nutzung in mehreren Bereichen aufgezeigt. Es werden in Simulationsumgebungen und Testversuchen auf einem Drohnentestbett sowohl die Funktionsweise überprüft, als auch auf mögliche Schwachstellen untersucht.

Contents

1 Introduction

Evaluating sensors of a system separately would provide information only derivable from the physical properties the measurements of a sensor represents. Combining different sensors, or similar sensor which are for example placed on different positions, makes it possible to derive other physical quantities and properties from the dataset created by the sensor data fusion. This process of sensor fusion is done to combine the advantages and different representation of an environment of single but also similar sensors. Especially in the field of range and visual measurements, the combination of these two sectors can lead to much bigger and more valuable environment representations. The laser scanner and camera, which are representing these fields, are used to gather distances for a specific direction and to gain intensive images of one or more spectra of electromagnetic radiation, respectively. The correlation of these two datasets has many application areas. Depending on the actual use case and available resources, a variety of approaches are already made by different people in different fields. This work starts by presenting some of these approaches in section 2. Using a laser scanner and a gray scale camera on a drone, presented in section 3.1, a fitting fusion method is being developed. Since this setup platform is a real-time environment and thus limited computation resources, a theoretical model built on ray tracing is developed in detail. Described in section 3.2, this model accepts an image coordinate, i.e. of a detected feature on the image, and calculates a corresponding distance from the camera to this point in the environment by only having one dataset of laser scanner and camera each. Since the model expects an already known relative position and rotation between the sensors, the determination of these is discussed in section 3.3. After that, in section 3.4 this model is implemented in a simulation environment and is validated with synthetic sensor data. Verified in the simulation, the fusion model is ready to be implemented in the software framework on a drone platform. After presenting the actual platform software environment, a performance-error analysis of this implementation is done through various scenarios with synthetic data. In the section 4, this analysis shows some cases this model is limited by the resolution of the laser scanner and can return misleading distances, but also due the structure of the model. Also, to increase the performance by simplifying the model in different ways, benchmarks are performed in connection with the error rate the simplifications increase. Finally, experimental tests and their results are presented in section 5 in order to verify the model, but also to uncover possible weak points. Some specific constellations producing wrong distances are successfully reconstructed and evaluated, which were already shown in the simulation. The appendix is showing data in form of visualizations and tables from simulation, analysis and experimental tests for all setups and configurations used in this work, which are discussed in the corresponding sections.

2 State of the Art

Previous works, specifically on camera-LiDAR (Light Detection and Ranging) fusion, show many different approaches to the problem to merge sensor data, depending on what is going to be the desired achievement.

The data of both sensors represent the scanned or captured environment but in a different context. The camera gains intensive images of one or more spectra of electromagnetic radiation. Most of them, as they are built into smartphones, capture light with the wavelengths of red, green and blue color. Since this is a passive sensor, no sources of energy are built-in. This is making the sensor more simple and it is possible to easily implement cameras with higher resolutions. The laser scanner, however, is an active sensor. This means, it sends out laser rays as a form of energy and captures the reflections of the environment within the wavelength of the sent out laser. Although

(a) View of scanner.

(b) 3D view.

Figure 1: Colorized point cloud by Klimentjew et al.[1].

this makes the sensor relatively more complex, its data describe depth information for specific directions being a completely different context of the same environment.

In the fusion process, it is important to map objects from the image data to objects of the laser scan data. These could be represented by features, segments or single data points. In a static case, in which the sensors are placed at fixed positions in the environment over time, assigning color values from the camera image to single data points of the point cloud is a good approach. In this way, 2D laser scanners can be used for 3D laser scanning by e.g. changing the angle of the 2D laser plane as for example done by Klimentjew et al.[1]. Forkuo and King[2] match image and laser scanning data by generating a "Synthetic Camera Image" out of the 3D point cloud from a laser scanner. This image represents the view of the camera within the 3D point cloud, which makes it possible to find correlating objects or features by various matching methods.

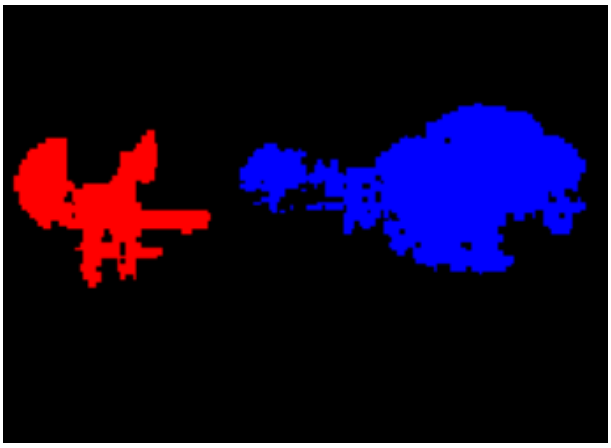
In dynamic configurations, i.e. moving vehicles which have to avoid obstacles, other approaches have to be considered. One way to do this is to generate states of the environment out of prior scan iterations (multiple time epochs) and migrate the new scan into the last known state. Stiller et al.[3] present an obstacle detection by building an obstacle map through the fusion of multiple different sensors with overlapping fields of view. This map is a good representation of the environment. With a Kalman filter, the current state of the vehicle within the obstacle map is being determined. The paper of Stiller et al.[3] also describes a high reliability of the detection system because of the

(a) Image captured by the camera.

(b) Synthetic view of the 3D point cloud in position of the camera.

Figure 2: Images by Forkuo and King[2].

(a) Image captured by the camera with scan line of the laser scanner.



(b) Segmentation of the parts of interest from the laser scan.

Figure 3: Images by Garcia-Alegre et al.[6].

high redundancy through the use of many different sensors. Becker et al.[4] achive this multiple epoch obstacle determination by detecting and tracking an object with each sensor individually. This way, the data is filtered prior to the actual sensor fusion. Also, before the fusion process starts, the different objects and their states are compared with each other and it will be detected if an object seen from one sensor corresponds to an object seen from the other sensor. Now, the data of each object can be merged into a whole state model, e.g. with the help of Kalman filters. Also, presented by Baltzakis et al.[5], two different visual datasets with a single data set of a 2D laser scanner can be used for approximately determine obstacles. This is possible due region detection within the images and extrapolating of the registered depth information.

If the resources, e.g. processing power or memory capacity, are limited, the maintaining of states of single objects or a whole obstacle map is not always possible. In this case, the distance determination of an object has to be acquired within a single time epoch. To classify objects in outdoor environments, Garcia-Alegre et al.[6] use a segmentation algorithm on the visual color image and merge them on a horizontal line of the laser data. With this method, the segments can be assigned with depth information and thus dynamic objects can be classified. The segmentation method reduces the processing time by minimizing the area-of-interest. But this technique is only available for sensors being positioned on the ground because there is only one 2D laser data set evaluated per epoch, which limits the sensor positioning and orientation within the environment.

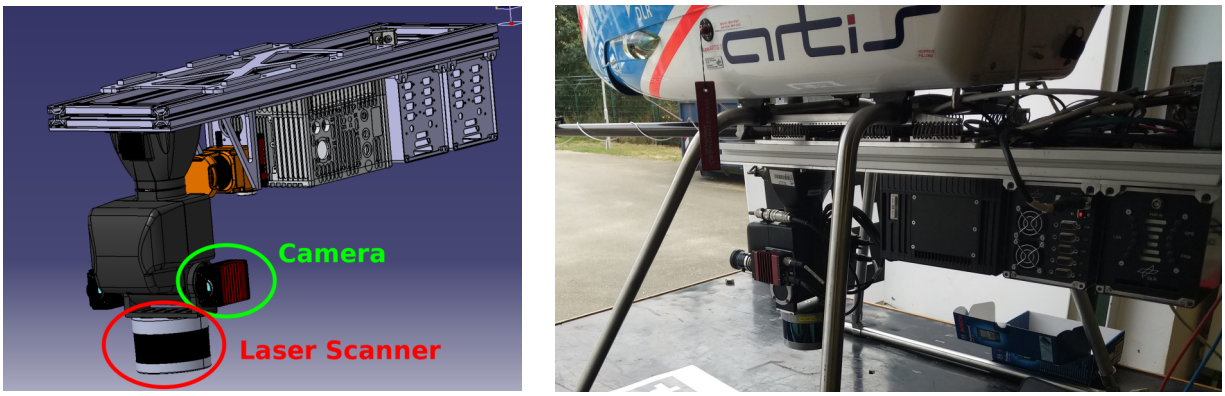
To acquire depth information e.g. from the air and detect how far objects are on the ground, another way has to be approached. This thesis solves this problem using a multi-line laser scanner. In this way, there are almost no constraints regarding the vehicle-environment constellation.

3 Theory on Sensor Fusion

The following sections develop a fusion method merging data of the laser scanner and camera. Based on spatial ray tracing, it only needs a few mathematical operations to determine distances from a single data set of each sensor.

3.1 Sensor Setup

As sensors, an AVT Prosilica GT 1380 camera[7] and Velodyne VLP-16 multi-line laser scanner[8] are used. They are rigidly mounted on the same rack with their view pointed in the same direction



(a) CAD model. **(b)** Mounted on superARTIS drone.

Figure 4: Setup of the Camera and Laser Scanner.

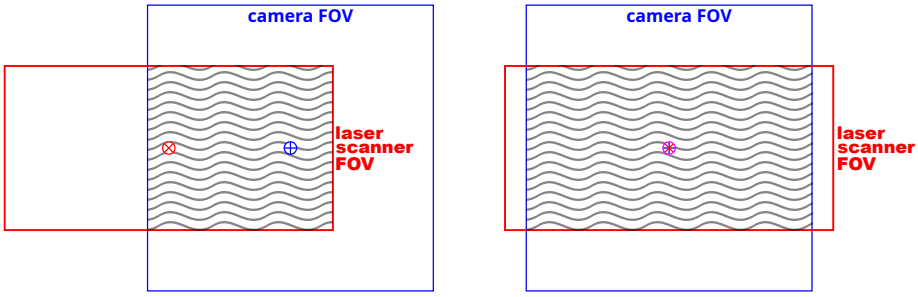
as it is displayed in figure 4. In this way, also shown in figure 5, the most area of the image field of view (FOV) is covered by the one of the laser scanner, which is needed to acquire distances for as many pixels of the image as possible.

With this configuration, the translation between the two sensors has to be considered in the model. But more importantly, the relative rotation has to be included too. Otherwise, both sensors have to be *exactly* oriented to the same direction. In addition, this makes it possible to apply the resulting model to a more complex sensor configuration, such as mounting the sensors on different independent racks. This would also give an option to dynamically rotate the camera can be with the use of the 360° width of the FOV of the used VLP-16 laser scanner. Both, the translation and rotation values have to be known and can be determined for example through calibration, which is discussed in section 3.3.

3.2 Ray Tracing Model

To assign a distance to each pixel of the camera image (or at least within the overlapping FOV's), a spatial solution is approached. The laser scanner radiates the laser rays as shown in figure 6.

With each scanning line having a different Φ angle, a mirror in the device is rotating, so that with each laser burst the Θ angle is changing. In this way, the distances are gathered with their elevation and azimuth coordinates within a spherical coordinate system. For simplicity, this system is used to represent the laser pixels in the three-dimensional room of the model. With the values of azimuth Θ and elevation Φ as the center points at the measured distance R, each pixel is projected



(a) Sensors pointing in different directions. **(b)** Both sensors pointing in the same direction.

Figure 5: Overlapping FOV's of camera (blue) and laser scanner (red).

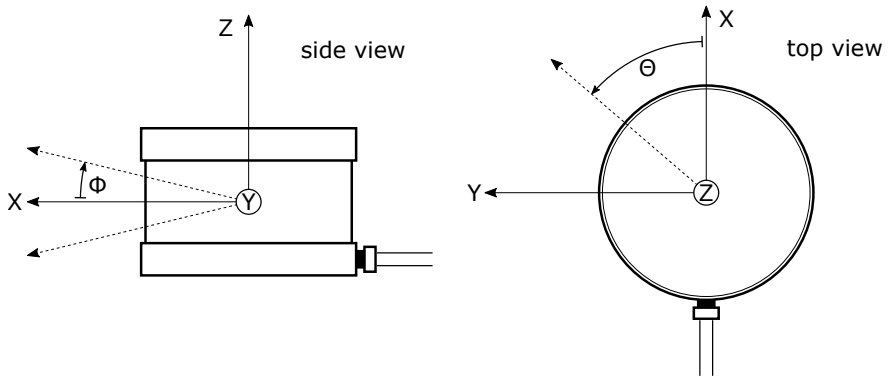


Figure 6: Laser scanner angle coordinates.

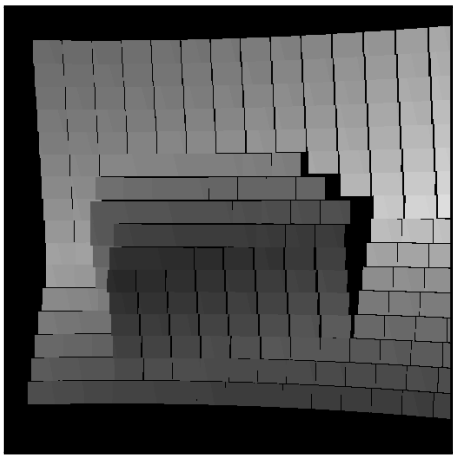
in a spherical coordinate system. This point gets stretched into a 2D plane which lies as a concave surface in the 3D model room described through a minimum and maximum set of Θ and Φ values, which are defined as follows:

$$\begin{aligned} \Theta_{\text{min/max}} &= \Theta_{\text{measurement}} \mp 0.5 \cdot \Theta_{\text{resolution}} \\ \Phi_{\text{min/max}} &= \Phi_{\text{measurement}} \mp 0.5 \cdot \Phi_{\text{resolution}} \end{aligned} \tag{1}$$

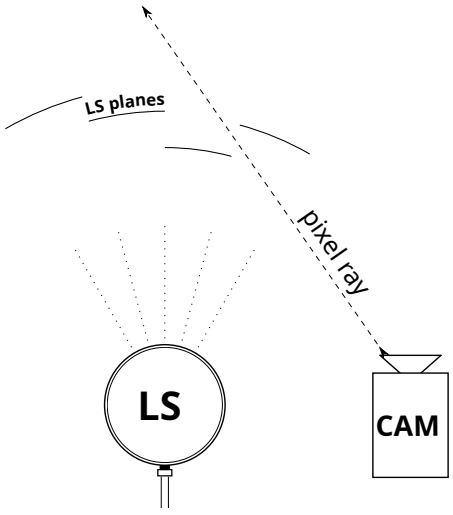
Where $X_{\text{measurement}}$ is the actual measured value and $X_{\text{resolution}}$ is the resolution of the laser scanner at this specific coordinate axis. The resulting laser pixel plane now represents each distance measurement within its Θ and Φ limits.

The camera is placed in the model room with the same relative translation and orientation to the laser scanner, as in the real world. Since the laser scanner lies in the origin of the models coordinate system and its orientation towards the positive x-axis, the camera can be positioned with the same (X, Y, Z) and (R, Θ, Φ) values[1] for the relative translation and rotation, respectively. The mathematical relation between the Cartesian and the spherical coordinate system used in the model room will be derived later over the course of the following paragraphs.

[1]These values have to be obtained e.g. through calibration. Different ways to archive this are discussed in section 3.3.



(a) Simulation result with laser pixel planes.



(b) Image coordinate ray hits a plane gap (2D visualization).

Figure 7: Example with laser pixel as planes. The camera position at the top right of the laser scanner.

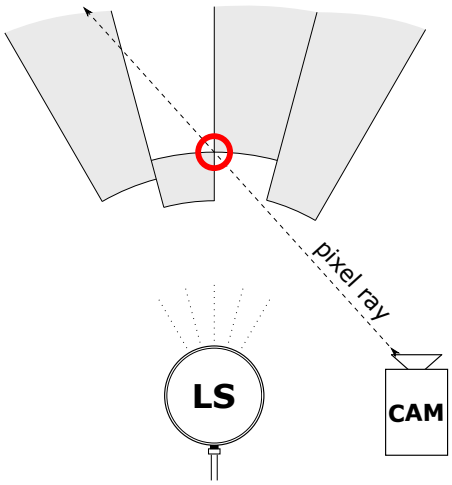


Figure 8: Case with an image pixel ray going between the corners of two laser pixel cubes.

As an example, the camera was positioned slightly at the top right of the laser scanner, which results in a view within the model room as shown in figure 7. The example scan data originates from a simulation environment, discussed in section 3.4.

As can be seen in figure 7a, there are gaps between the laser pixels from the view of the camera. If a ray from a camera pixel coordinate lies just in this gap as shown in figure 7b, no distance can be acquired. To counteract this, the laser pixel plane will be stretched into a volume by defining an additional minimum and maximum value of the distance R coordinate:

$$\begin{aligned} R_{\text{min}} &= R_{\text{measure}} \\ R_{\text{max}} &= R_{\text{measure}} + R_{\text{MDN}} + R_{\text{margin}} \end{aligned} \tag{2}$$

Whereas R_{MDN} is the distance between the R_{min} values of the current laser and the most distant direct neighbor pixel. If all direct neighbor pixels are closer to the laser scanner (smaller R_{min}) than the current pixel, then R_{MDN} equals zero. The R_{margin} value acts as small additional depth to be sure to get a ray hit in case the camera pixel ray just hits exactly the corners between the front of the more distant pixel and the back of the closer one as shown in figure 8. The ray tracing actually used is discussed in the next few paragraphs.

With each laser pixel volume having a minimum and a maximum value for each coordinate representing a "cube" in the spherical coordinate system, ray tracing is used as the method to get a distance of each camera image pixel. A ray is defined by the camera position as origin $\vec{O}$ and the direction $\vec{D}$, which results from the current pixel in the image. $\vec{D}$, however, is defined as:

$$\vec{D} = \mathbf{R} \times \vec{D^P} \tag{3}$$

or

$$\begin{aligned} \begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} &= \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \times \begin{pmatrix} D_x^P \\ D_y^P \\ D_z^P \end{pmatrix} \\ \text{whereas } \begin{pmatrix} D_x^P \\ D_y^P \\ D_z^P \end{pmatrix} &= \begin{pmatrix} FocalLength \\ (I_{\text{width}} \cdot 0.5) - PX_X - 0.5 \\ (I_{\text{height}} \cdot 0.5) - PX_Y - 0.5 \end{pmatrix} \end{aligned} \tag{4}$$

with $\mathbf{R}$ being the rotation matrix from the center view of the laser scanner to the one of the camera, $\vec{D^P}$ the direction vector from the pixel within the image, $PX_{X/Y}$ the pixel coordinates with the

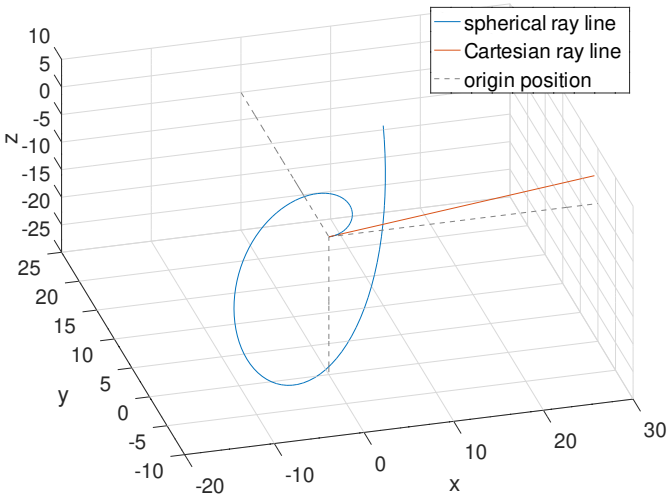


Figure 9: Ray lines in Cartesian and spherical coordinate system.

origin in the top left corner of the image, and $I_{\text{width/height}}$ the image dimensions. Note that the $FocalLength$ is given in 1·[pixel] instead of 1·[mm]. Also, the $\vec{D}$ vector has to be normalized since it represents a direction only.

With the normalized direction vector $\vec{D}$, a point $\vec{P}$ on the pixel ray can be described as follows:

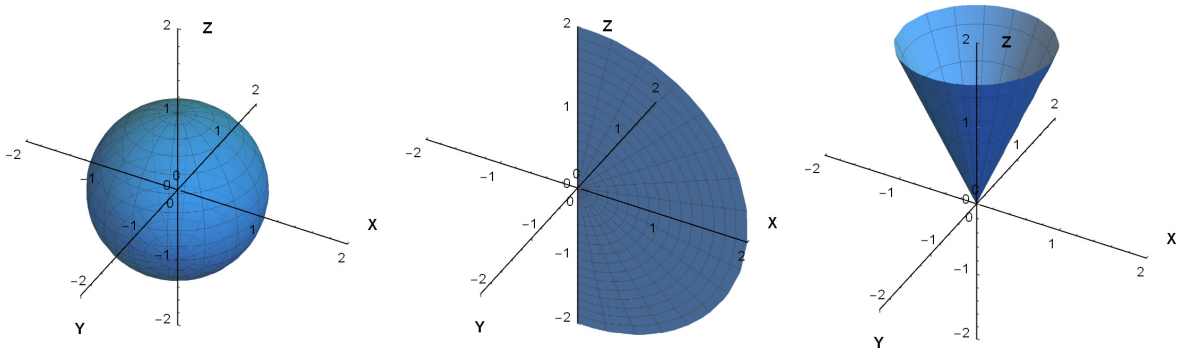
$$\vec{P} = \vec{O} + t \cdot \vec{D} \tag{5}$$

or

$$\begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} = \begin{pmatrix} O_x \\ O_y \\ O_z \end{pmatrix} + t \cdot \begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} \tag{6}$$

To check whether a ray hits a laser pixel cube or not, the pass through each cube side is calculated, which are two calculations for each coordinate axis. The closest pass, if any, is considered as a ray hit. This is done by calculating the t values of the corresponding passes, which end up being in the limit values of each coordinate axis as the cube sides. But since these limits are defined in spherical coordinates and the direction of the rays is a straight line defined in Cartesian coordinates (see ray line comparison in figure 9), the spherical coordinates for a point of the ray depending on the t value has to be determined. It is known how to get the spherical coordinates of a point defined with (P_x, P_y, P_z):

$$\begin{aligned} R &= \sqrt{P_x^2 + P_y^2 + P_z^2} \\ \Theta &= \arctan\left(\frac{P_y}{P_x}\right) \\ \Phi &= \arctan\left(\frac{P_z}{\sqrt{P_x^2 + P_y^2}}\right) \end{aligned} \tag{7}$$



(a) Constant R-coordinate. **(b)** Constant Θ-coordinate. **(c)** Constant Φ-coordinate.

Figure 10: 3D surfaces at constant spherical coordinates.

Inserting equation 6 into eq. 7 results in:

$$\begin{aligned} R &= \sqrt{(O_x + t \cdot D_x)^2 + (O_y + t \cdot D_y)^2 + (O_z + t \cdot D_z)^2} \\ \Theta &= \arctan\left(\frac{O_y + t \cdot D_y}{O_x + t \cdot D_x}\right) \\ \Phi &= \arctan\left(\frac{O_z + t \cdot D_z}{\sqrt{(O_x + t \cdot D_x)^2 + (O_y + t \cdot D_y)^2}}\right) \end{aligned} \tag{8}$$

As the sides of the laser pixel cubes are defined as spherical coordinate values (left-hand side of the equations 8), it is only needed to isolate the t values for each spherical coordinate.

$$\begin{aligned} t_R = \frac{1}{D_x^2 + D_y^2 + D_z^2} \cdot \Big(&-D_x \cdot O_x - D_y \cdot O_y - D_z \cdot O_z \pm \\ &\sqrt{(D_x \cdot O_x + D_y \cdot O_y + D_z \cdot O_z)^2 - (D_x^2 + D_y^2 + D_z^2) \cdot (O_x^2 + O_y^2 + O_z^2 - R)}\Big) \end{aligned} \tag{9}$$

$$t_\Theta = \frac{O_x \cdot \tan(\Theta) - O_y}{D_y - D_x \cdot \tan(\Theta)} \quad \text{whereas} \quad O_x \cdot D_y \neq O_y \cdot D_x \tag{10}$$

$$\begin{aligned} t_\Phi = \frac{1}{(D_x^2 + D_y^2) \cdot G - D_z^2} \cdot \Big(&D_z \cdot O_z - H \cdot G \pm \\ &\sqrt{G \cdot \left(D_z^2 \cdot (O_x^2 + O_y^2) - 2 \cdot D_z \cdot H \cdot O_z + (D_x^2 + D_y^2) \cdot O_z^2 - (D_y \cdot O_x - D_x \cdot O_y)^2 \cdot G\right)}\Big) \\ \text{with} \quad G = \tan^2(\Phi) \quad &\text{and} \quad H = D_x \cdot O_x + D_y \cdot O_y \end{aligned} \tag{11}$$

Both the t equations of the R and Φ coordinates (eq. 9 and 11) have two solutions, since a straight ray line can hit a sphere (constant R coordinate, figure 10a) and a cone (constant Φ coordinate, 10c) twice. If the square root discriminant is negative, the ray misses the sphere or cone. Otherwise, the ray only touches them at one point on the surface with the discriminant being zero.

To check for hits, the t value(s) of each side of the laser pixel volume have to be calculated. However, these values only represent hits of the current minimum or maximum R, Θ or Φ coordinate of the pixel cube, this t value was calculated for. These surfaces are a sphere, a vertical plane or a cone (or horizontal plane), respectively (figure 10). The equations 8 calculate the hit point coordinates. With the minimum and maximum values of the other coordinate axes of the pixel cube (eq. 1 and 2) the current t value is not calculated from, it can be checked if the coordinates of this t value are in the range of the laser pixel volume. Notice that a very small margin value[2] is added to the maximum and subtracted from the minimum values of each coordinate in the

[2]The magnitude of this margin value depends on the floating-point precision the implementation uses.

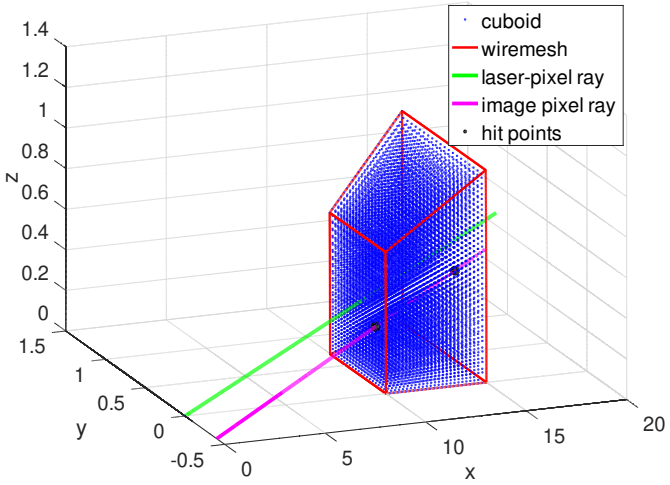


Figure 11: Hit by ray tracing applied to a spherical cuboid. X-axis scaled.

range checking to prevent floating-point precision errors in the algorithm implementation later on. Without doing so, some hit points would be wrongly considered as not lying within the ranges. After this, the smallest t value of the remaining ones is chosen to be the hit, while negative t values are not considered since the ray is radiated forwards. There are cases where negative t values can occur, e.g. if the ray origin lies in the laser pixel volume itself. If no t value remains after the range check, the ray does not hit the laser cube. Figure 11 shows a spherical cuboid with the ray of the measurement and the ray of the image pixel and its two ray hits in the front and the back.

To check a ray hit of an image pixel in the whole model world, it is only needed to iterate through all cubes and choose the smallest t value as a hit, if any value is available. Figure 12 shows a visualization of the ray hits in a 2D model room.

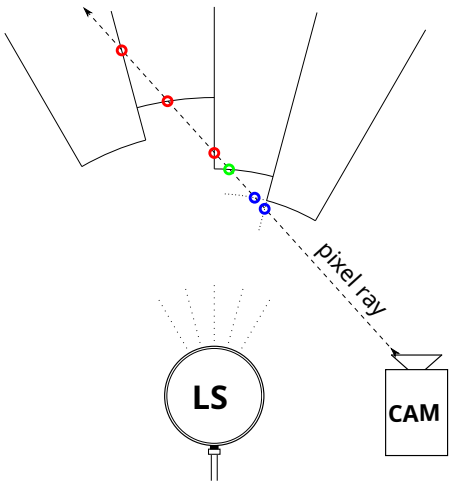


Figure 12: Ray tracing model as 2D visualization.
blue: ray hits of the coordinate axis surfaces of the most right cube, but invalid due range check
green: first ray hit within the cube-range
red: ray hits with greater t values than the first valid hit

3.3 Relative Sensor Translation and Rotation

The translation between the laser scanner and the camera will be measured by hand in each axis of the Cartesian coordinate system they lie in. In the use case of a drone, this should be precise enough since a misalignment in the millimeter level should not affect measurements which are 20 meters and upwards away. The relative rotation between these two sensors, however, has to be determined much more precisely, since every degree of error will be quadratically amplified by the radius, or the distance, the measurement points are away.

3.3.1 Direct Calibration

If the two sensors, laser scanner and camera, are considered without any context, a direct calibration between these can be applied. The only constraint is, that these sensors are rigidly connected to each other; or if not, every change of their relative translation and rotation has to be known or determined somehow.

There are different ways to calibrate this constellation. The most common way is, to use a checkerboard, or at least a plane with a checkerboard-like texture. This way, the camera can determine its position to the board through its scale and distortion within the image, while the laser scanner position can detect the plane and its relative position through several sensor-board constellations. Although this is common, many different ways to improve the accuracy or to simplify the calibration procedure itself have been already developed. See Chai et al.[20], Kassir and Peynot[21], Zhang and Pless[22], Krause and Evert[23], Bi and Lu[24] for reference.

Besides this way, there are also techniques of having no calibration object from Scaramuzza et al.[25] or having other sensor constellations, like a stereo camera system which have to be calibrated with a laser scanner by Aliakbarpour et al.[26] . It is also possible to have some rough parameters from a less precise calibration already and improve these in another step afterwards. The technique of Peynot and Kassir[27] improves these parameters iteratively over time, so that the values diverge to the true relative rotation of the sensors. Rönnholm and Haggrén[28] are doing this too, but instead, they interactively change the exterior orientation parameters of one sensor. The final relative rotation is determined by calculating the misalignments through these small parameter changes.

3.3.2 Calibration with an Inertial Measurement Unit

The most applications with laser scanner and camera merged into one system are robotic systems or drones. But these applications very likely have a built-in inertial measurement unit (IMU). Therefore, besides the relative rotation between the laser scanner and camera, also these between

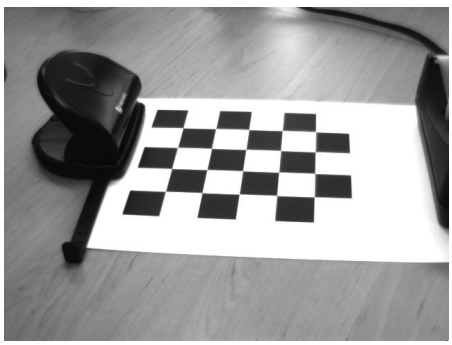


Figure 13: Checkerboard calibration pattern with its plane normal oriented to the gravitational vector. Picture from Hol et al.[29].

the IMU and the other two sensors have to be known to correctly relate and fusion their data. An indirect approach for the calibration *between* the visual and range sensor is to separately calibrate the camera-IMU and the laser scanner-IMU constellation, and combine the results to the final relative rotation between them.

3.3.2.1 Relative Rotation between Camera and IMU

As it is common to use a checkerboard for calibrating the orientation of laser scanner and camera, it is for calibrating the orientation between camera and IMU too. There are different approaches: either the checkerboard is oriented to the gravitational vector of the IMU as shown in figure 13, or the delta positions and delta IMU measurements between short time points are used to determine the relative rotation of these sensor frames. See Hol et al.[29], Kleinert and Stilla[30] for references.

There are also more stable methods of calibrations, as Rehder and Siegwart[31] are also considering a possible misalignment between individually placed accelerator axis sensors within the IMU. On another side, Li and Mourikis[32] take the variation of the time of arrival for each sensor into account, since either the calibration process or the online fusion of the sensors need accurately aligned timestamps to function properly.

The method used in this thesis is a toolbox from Lobo and Dias[33]. With several different poses of the rigidly mounted sensor configuration and therefore different views of a size-defined checkerboard, this toolbox written in MATLAB can determine the relative orientation between the camera and the IMU. It uses the advantage of having the checkerboard placed along the gravitational vector, but needs human interaction of the interface to select corners of the checkerboard for each captured pose. This interaction is part of the Camera Calibration Toolbox[34] used by Lobo and Dias[33].

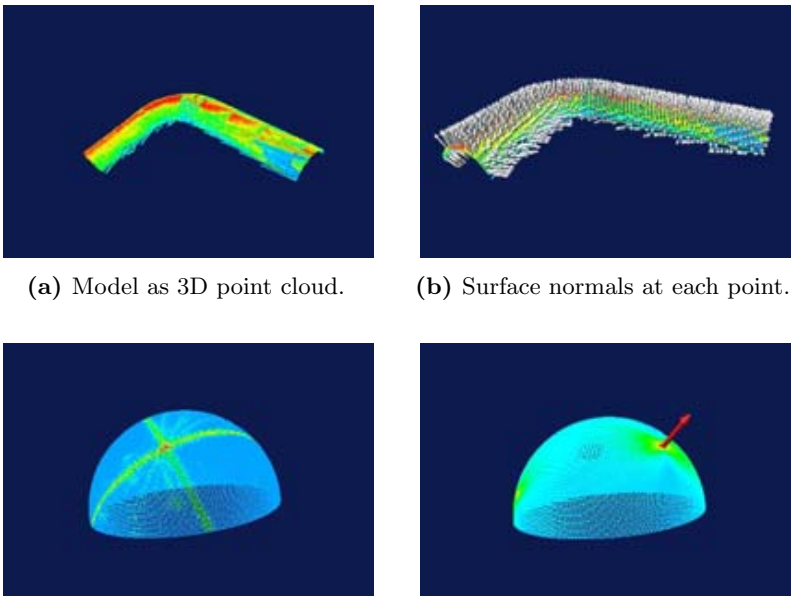
3.3.2.2 Relative Rotation between Laser Scanner and IMU

The relation of the laser scanner and the IMU is difficult because the range of the laser scanner is only a 2D plane in a 3D world, where the distance measurements are located for each horizontal angle. Even, a multi-line scanner like the Velodyne VLP-16[8] used in this thesis has only a very low vertical scanning resolution. This makes it difficult to actually create enough data points needed for the calibration. Consequently, there is not much work done yet related to this topic.

There are related papers, e.g. from Talaya et al.[35] who include the known position over time, or trajectory, to perform a proper calibration. Baglietto et al.[36] for example use an extended Kalman filter to calibrate the relative rotation between the laser scanner and the IMU, but also to evaluate and assign data points in the model world.

With the laser scanner used in this thesis, the point cloud of one scan contains only around 7200 data points. If large objects are scanned, their surfaces could be reconstructed. In this way, the gravitational vector can be determined for each scan and the same continuing principle as with the oriented checkerboard can be applied to gain the relative orientation. A very simple way is to use walls or their edges. Ensured the walls of a considered edge are oriented along the gravitational vector, a similar method as by Rabbani and Van Den Heuvel[37] can be applied. Reconstructing the surface normals of the point cloud by e.g. k-nearest neighbor search, they can vote through Gaussian spheres as Hough transform spaces for all directions which are orthogonal to their surface vector. In this way, in the Hough space is a peak, which should be a cross point of the two walls, that can be considered as a direction of interest for the subsequent calibration procedure. Visualizations of an example can be seen in figure 14 as presented in their original paper.

Analogue to this idea, and more close to the paper of Rabbani and Van Den Heuvel[37], a cylinder can be hanged up, which then points along the gravitational vector. Using the method in the named paper, the direction can be directly extracted. This method is stable to outliers, like any



(a) Model as 3D point cloud.

(b) Surface normals at each point.

(c) Surface normals plotted at Gaussian sphere.

(d) Each normal vector votes every orthogonal direction on another Gaussian sphere. Peaks to be considered as possible cylinder directions.

Figure 14: Created Gaussian sphere as Hough transform space by Rabbani and Van Den Heuvel[37].

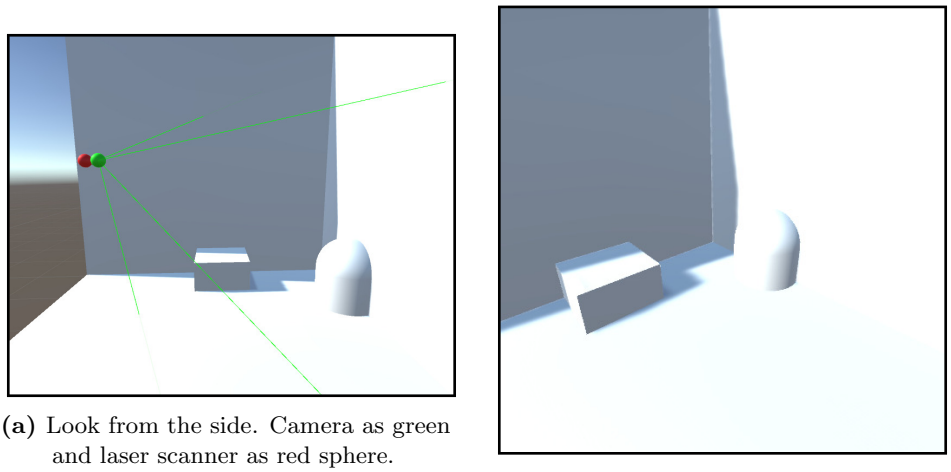
other object in the scene would produce. However, no other cylinder-like object can be in the scanned scene, because otherwise a wrong direction vector would be determined.

An alternative method from Eberly[38] fits one cylinder to all given points in the point cloud. Therefore all points not representing the cylinder have to be excluded though e.g. classification algorithms applied to the raw point cloud. Once fitted, the direction of the center axis of the found cylinder can be extracted as the vector which is searched.

Having several poses, in each of them the gravitational vector is measured by the IMU and the laser scanner. In other words: in each frame there are several pairwise-related known vectors, describing a Wahba's problem[39]. This can be solved either by the q-method[40] or in time-critical systems through QUEST[41].

3.4 Simulation

To easily verify the ray tracing model discussed in the prior section, the Unity game engine version 2018.2.3f1[9] is used as simulation environment. With virtually created objects shown in figure 15 representing a real environment and the built-in ray casting function of the Unity game engine, the laser scans can be simulated. The same laser scanner and camera setup as described in section 3.1 is used. Thus it is simulated a 16 line laser scanner with an adjustable horizontal resolution and a camera with adjustable pixel resolution and FOV.

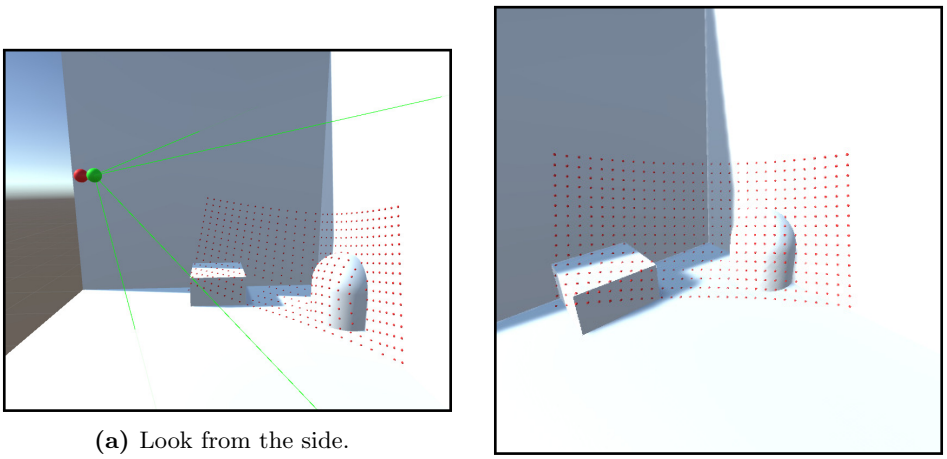


(a) Look from the side. Camera as green and laser scanner as red sphere.

(b) Look from camera view.

Figure 15: View of the simulation environment.

Figure 16 shows the laser scanner hits in the simulation environment as small red spheres. The distance and angle information of the ray hits are transformed into the model coordinate system as it would be in the real setup later on.

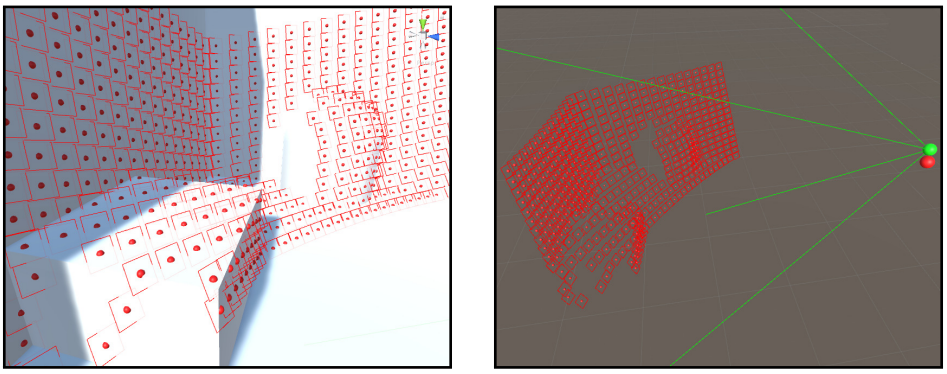


(a) Look from the side.

(b) Look from camera view.

Figure 16: Hit points of the laser scanner.

For visualization purposes, the edges of the front surface of the laser pixel cubes are shown in figure 17. Notice that for simplicity only the corners got connected instead of redrawing the actual edges.

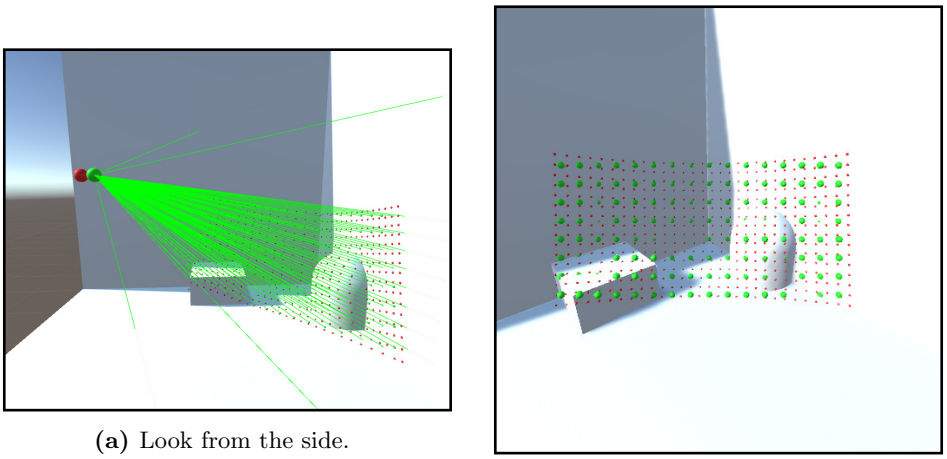


(a) Fronts of the laser scanner cubes dive into the surface of the actual objects.

(b) Fronts of the laser scanner cubes only, forming approximately the simulated world.

Figure 17: Visualization of the front surfaces of the laser pixel cubes.

Now the model algorithms are executed with each pixel of the simulated camera view (here 24x24 [pixel]) as a ray for the ray tracing model. The resulting pictures in figure 18 show the pixel rays and their hits as green, while keeping the laser scanning hit points as red spheres.

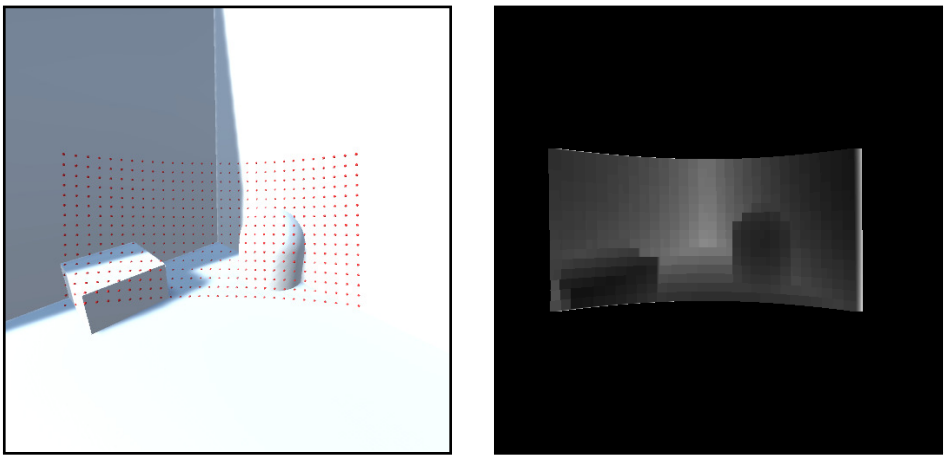


(a) Look from the side.

(b) Look from camera view.

Figure 18: Executed ray tracing model.

With a 512x512 [pixel] camera view, the corresponding depth or distance map is generated, as shown in figure 19. The camera FOV was 80x80° whereas the one of the laser scanner was 60x30°. More distant ray hits are represented as brighter pixel. The black areas around the laser scanned field are in the FOV of the camera – but without laser pixel cubes of the laser scanner available to get distance information from.



(a) Camera view image. **(b)** Depth map.

Figure 19: Simulation results.

As noticeable on the top, bottom and right side of the non-black area, there are white fades. These are ray hits of the camera pixel, which hit laser cubes positioned at the edges of the cube field. This results in hitting the side surfaces way beyond the fronts of the cubes and thus a greater distance. This effect is reduced by make a laser pixel only as depth as the most distant direct neighbor pixel as described earlier, but it can not completely avoided. Its occurrence is further discussed in section 4.3.

4 Implementation

To prove the functionality of the fusion model developed in the sections before, it is implemented into a software framework, which runs on the drone testbed ARTIS.

4.1 Software Framework

The framework the ray tracing model is implemented in is a Tasking Framework developed by the German Aerospace Center (DLR), which was originally developed on a management system based on RODOS[10].[11] It is designed to easily model e.g. sensor inputs and processing algorithms as tasks. This way, for instance, one task can wait for input updates before running. To trigger tasks, not only by ready inputs, events are used to provide a processing-on-demand behavior. By not simply clock these tasks at a fixed rate the real-time processing requirements are going to be fulfilled. The used operating system is CentOS release version 7.5.1804[12], where the Tasking Framework got migrated to. Figure 20 shows this as a block diagram.

First used in the DLR's Autonomous Terrain-based Optical Navigation (ATON) project[13], the Tasking Framework proved itself as real-time capable. But the module architecture of single

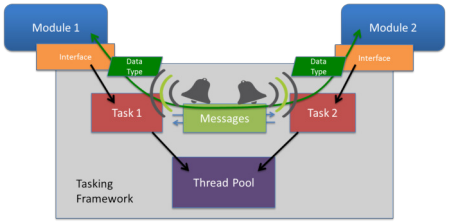


Figure 20: Tasking Framework. Figure from Rittweger et al.[14].

(a) MDSD overview

(b) Code generation from a template

Figure 21: Model-Driven Software Development. Figures from Rittweger et al.[14].

software parts brought the problem of the interfaces between them, which are hard to maintain especially when changes in existing source code have to be made or incompatibilities occur. Because of this, Model-Driven Software Development (MDSD) was introduced (figure 21a). This makes it possible to build up the whole software architecture as a model, for instance, with a visual diagram editor. UML and SysML are used as a combination to build up the program in models, which can be connected through ports among themselves. Source code can be generated out of these models with exact interface definitions. These ensure a global compatibility between the interface variables and the conversions between them and individual model variables. Figure 22 shows the whole model structure of the framework used in this thesis. Small changes can be adjusted in the models, and to project these changes into the source code only a regeneration is needed instead of manually reviewing the source code. For individual algorithm implementations *generation gap patterns* are used. A subclass of the corresponding module is generated one-time and its generated base class accesses this subclass (which derives from the base class) running the individual algorithms. All this improves the collaboration between developers, which work in very different fields especially in large projects. This is because interfaces between the modules are well defined, and incompatibilities are uncovered immediately. [14][15]

The used MDSD technique has also some other features. One advantage is, that the model can be generated in every programming language desired with the help of generation patterns (figure 21b). They completely disconnect developers from the need to stick with a specific language and thus possible system limitations. Besides the generation of the source code itself, MDSD also generates documentation files and unit tests. These documentations, created as LaTeX files,

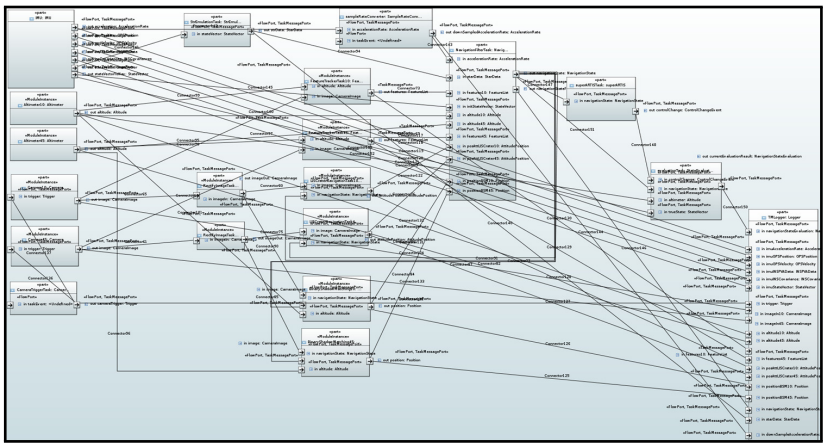


Figure 22: Overview of the model structure of the used software framework. Figure from Rittweger et al.[14].

contain descriptions of the used elements and type definitions of input and output parameters. Unit tests are added in order to check for unsupported UML/SysML elements, but also to check the compatibility between interfaces of the modules. An additional generation of a configuration manager makes it possible to read in pre-configuration files the modules get initialized with. It is easier to just adjust the configuration files instead of working with the whole module every time. The MDSD is doing this job by adding the parsing source code into the software project. In the end, appropriate building scripts will be generated too. Each module gets its own building script containing corresponding include- and source files. [14][15]

One important feature is the possibility of using recorded sensor logs as sensors itself. This way real flight data can be replayed on a stationary system, or even without hardware sensors, and perform tests as a real flight would be performed.[14][15] Before the ray tracing model is executed in a real event, sensor logs of a test flight are going to be fed into the model software framework and, subsequently, its behavior verified. Also, simulation data will be prepared in the same format as the sensor logs for the performance-error analysis later on. In this way distance deviations because of the model or its simplifications, and at which level, can be verified.

4.2 Image Feature Detection

The ray tracing model needs X and Y pixel coordinates of the camera image in order to create a ray (equation 4) and determine to the depth of this particular pixel, if covered by the laser scanner. These coordinates are the image position of an AprilTag, which are used in the software implementation for feature detection. These tags are two-dimensional binary tags, and the determination system allows a fast and robust six degree of freedom localization of these with a single image.[16][17] The AprilTag software module will be fed with rectified images of the camera and returns the position of the center of the tag features as image coordinates to the data fusion module. In there the ray tracing module determines the distance from the camera to this feature tag. See an example in figure 23.

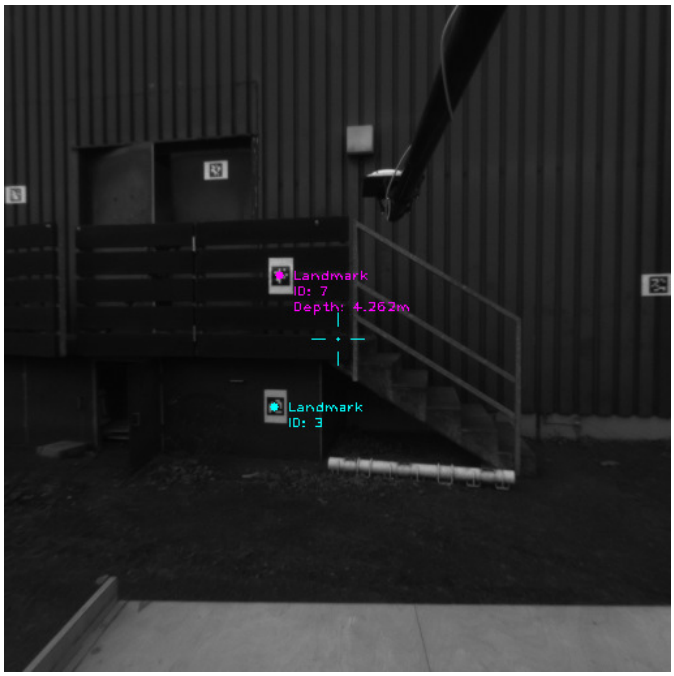


Figure 23: AprilTags in a real environment.
One tag with determined distance.

4.3 Performance-Error Analysis

In the following sections the time performance of the implementation is analyzed. To further increase this performance, different model simplifications are done which, however, increase the error rate of the model results.

4.3.1 Model Simplifications

To get an idea of the accuracy and the performance of the model algorithm, evaluations of these parameters in different configurations have to be made. Also, problems occurring by applying this model to different environment constellations, which are scanned by the sensors, will be explained. These can cause minor, but also major misinterpretations of the digitalized environment.

Starting with the correlation of the performance with the inaccuracy level of the model, it gets simplified in different ways. This is to get better performances at the costs of a higher inaccuracy of the distance measurements. For comparable results, the tests are performed on an ADLQM87PC system[18], which is used throughout this thesis. It consists of an Intel i7-4700EQ processor running at 2.40GHz and a random access memory of 8GB. The whole software framework is multi-threaded, but the algorithm for the sensor data fusion run only in a single thread.

Instead of acquiring the distances of several AprilTags, all pixels of the camera image are used, creating a whole depth map. In this way, if the true distances from the camera are known, an error map can be created which represents the difference between the resulting depth map and the true distances. However, the true distances of a real environment are hard to obtain. Even if obtained, they are afflicted with errors of a minor level at least. To counteract this problem, data of the simulated environment is used. The same camera and laser scanner have properties as the real sensors. These are the fields of view and the resolutions. This means, exact true distances can be acquired to create the error map. In turn, because of the performance analysis, the model cannot be run at the simulation itself, but have to be executed on the hardware system with the software framework. Since this framework provides a replay feature (section 4.1) it is possible to feed it with sensor data generated out of the simulation and run the ray tracing model on the ADL hardware system. With this method, the correlation between the performance and the inaccuracy level of the model will not be disconnected.

The camera and the laser scanner were placed in the simulation environment in four different configurations. The laser scanner is on the same position rotated the same way in each configuration. Table 1 shows the relative position and the direction of view of the camera as normalized vector. $x = 1$ equals to the front of the laser scanners view, $y = 1$ to the left and $z = 1$ to the top. So if both sensors are on the same position, the relative position vector of the camera would be $(0, 0, 0)^T$. The laser scanner points always to the front, so it has a direction vector of $(1.0, 0, 0)^T$. If they are

C	rel. position	direction of view	note
1	$(0, 0, 0)^T$	$(1.0, 0, 0)^T$	same position and direction
2	$(0, -0.7, 0.15)^T$	$(1.0, 0, 0)^T$	changed position, but same direction
3	$(0.2, 0.9, -0.1)^T$	$(0.866, -0.5, 0)^T$	changed position and direction (30°to the right)
4	$(0.6, -1.6, 0.4)^T$	$(0.951, 0.309, 0)^T$	changed position and direction (18°to the left)

Table 1: Sensor configurations used for analysis. Relative position and direction of view of the camera only, since they are relative to the laser scanner. Values represented as vector $(x, y, z)^T$.

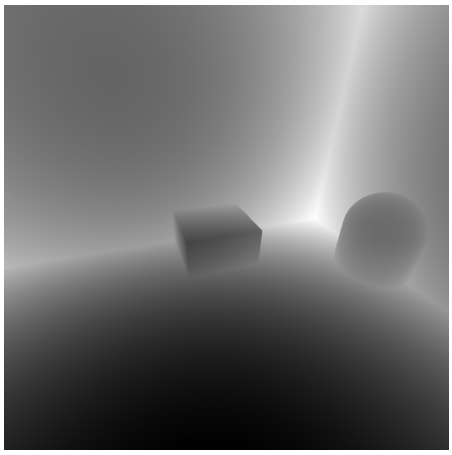


Figure 24: True distances from the camera of configuration 1.

pointing in the same direction, the vector of the camera would be $(1.0, 0, 0)^T$ too. The unit of the values is 1 meter.

Each configuration is ray traced with six different steps of model simplification, which are the following:

0. step (M0): no simplification / full model
1. step (M1): only use the front planes of each laser pixel cuboid / no stretching into a volume (see figure 7)
2. step (M2): laser pixel volume, but 5 horizontal neighbor cuboids got merged into one (depth got averaged)
3. step (M3): laser pixel plane, with 5 horizontal merged cuboids
4. step (M4): laser pixel volume, with 20 horizontal merged cuboids
5. step (M5): laser pixel plane, with 20 horizontal merged cuboids

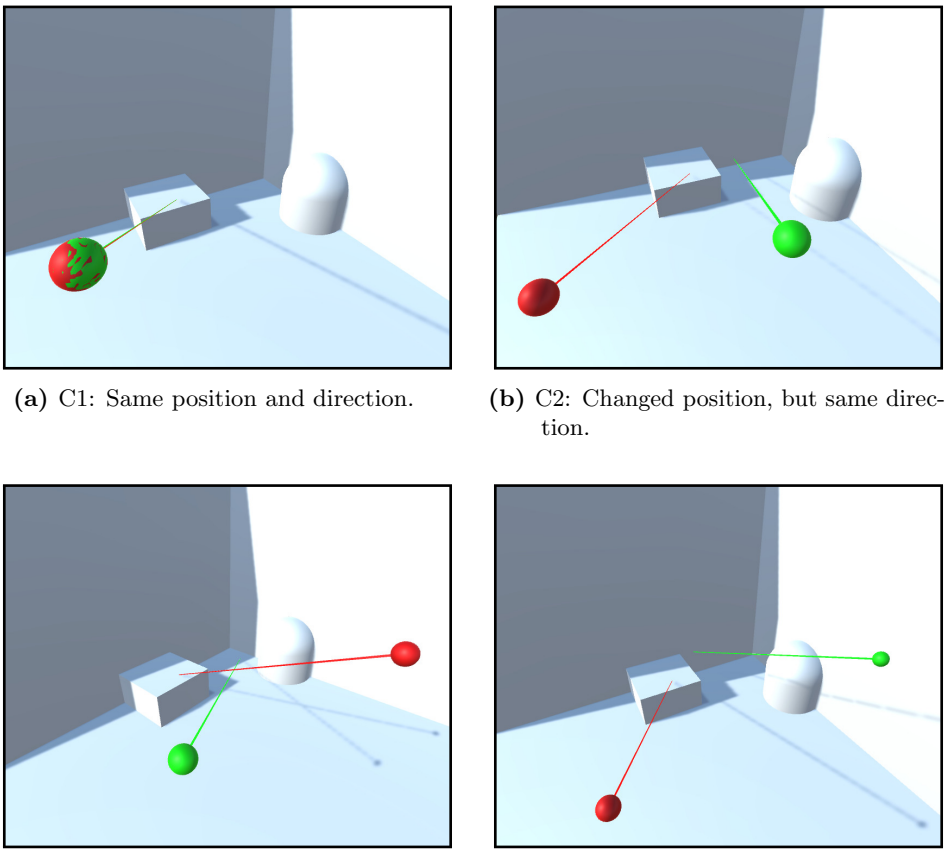
The goal of merging horizontal cubes into one in step 3 to 6 is to greatly increase the performance of the model algorithm – but for the cost of increased inaccuracy of the depth determination.

For the calculation time needed, each ray trace of a pixel gets measured and averaged. To also give an uncertainty of this averaged value, the Welford algorithm is applied, which calculates the variance of a running stream of values.[19] This online variant of variance calculation removes the need of saving all values in the memory, which is a lot with 1024x1024 [pixel] at 64bit (8 bytes) each.

The inaccuracy of the current model and configuration setup is classified in two different directions: one direction in which the model determined a depth too far (positive values), the other in which it is determined too close (negative values). For each setup an error map is produced, indicating the amplitude of each wrong determined depth of a pixel. All error amplitudes greater than 1 meter are colored in the maximum color value, making it possible to look closer to the errors between -1 and +1 meter. Negative values are shown in red, whereas positive ones in the green color. Blue indicates that the model couldn't determine a distance at this pixel, because either the field of view of the laser scanner is too small or it just did hit a gap between two laser pixel planes, shown in figure 7b.

Following discussions regarding visualized maps are only for the specifically named and shown model-configuration setups. See Appendix B: Error Maps for the visualizations of all other setups.

As all error maps, figure 26 is generated by subtracting the true depth values, which are generated from the simulation, from the depth map determined by the model. This makes depth values,



(a) C1: Same position and direction.

(b) C2: Changed position, but same direction.

(c) C3: Changed position and direction (30°to the right).

(d) C4: Changed position and direction (18°to the left).

Figure 25: Visualized sensor configurations. Laser scanner as red and camera as green sphere. Colored lines represent directions of view.

which are determined too close, negative – thus values determined too far are positive. The error map of M0C1 (**C**onfiguration **1** in the **M**odel simplification step **0**) has high variations at the edges of the environment objects in front of the walls. The reason for this are the laser scanner pixels (volumes, particular in M0) being bigger than an image pixel would be in the environment. This means these laser pixel volumes stick out of the actual object shape from the point of view of the camera, indicating the negative areas around the objects. On the other hand, there are also laser pixels sticking *in*, which are depth determinations from the walls behind the objects. On the ground and the top side of the cube object there are stripes of small negative and positive delta values observable. Since the center points of the laser pixels mapped in the environment are the actual depth measurements — meaning at these points the delta of depth is zero — and the front of each laser pixel looking to the position of the laser scanner, the edges of them are somewhat rotated away from the plane actually measured in the environment. Especially the top and bottom edges of the pixel of the ground plane are showing this constellation. If a vertical plane would be measured, of which the surface is not pointing to the laser scanner, vertical stripes of delta values would be observable. Also, if a measured surface is *exactly* oriented to the laser scanners position, the laser pixel have the least error rate integrated over the whole laser pixel front, but the corners still would indicate only negative values, since the front of the pixel in the ray tracing model is concave.

Figure 27 shows the error map of the M0C4 setup. The same variations occur at the edges of the environment objects as in the M0C1 setup. Additionally to this, there are red steps at the

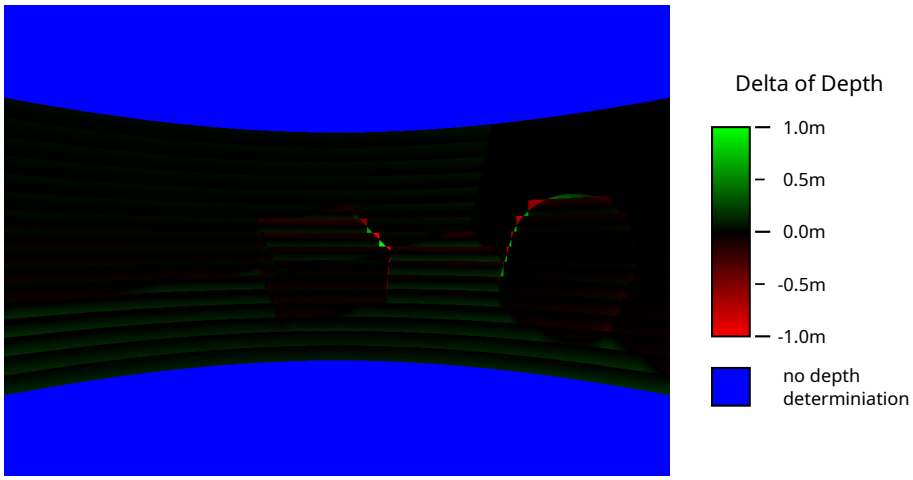


Figure 26: Error map of configuration 1 in model simplification step 0.

right side of the environment cube object, which are fading out to zero (black) to the right. These steps are laser pixel volumes. Their depth measurements of the laser scan are at the front of the cube object. But since the laser scanner is placed more to the left, the laser pixel volumes are scaled in depth to the right side direction. This big incorrect depth determination emerges from the idea and setup of the actual ray tracing model. More problems occurring in the same ways are discussed later in section 4.3.2.

Going on with the first model simplification step, figure 28 clearly shows gaps within the laser scanned area of the environment. For instance, in place of the red steps described in the M0C4 setup, there is now only a no-depth determined area, which derives from the simplification of the laser pixel being planes instead of scaled volumes. Also observable, there are positive delta depth values between the front laser pixel planes of the environment cube object. At these points, the depth of the laser pixels behind the the front of the cubes is determined, whereas the true depth values are these of the font itself. This means, that there is a chance of mistakenly determining a distance from an object behind the one, which actually is wanted to be determined.

If the first model simplification step is simplified furthermore by averaging horizontal neighbor laser pixels, another effect occurs. The M3C4 setup shows emerging laser pixels in the no-depth

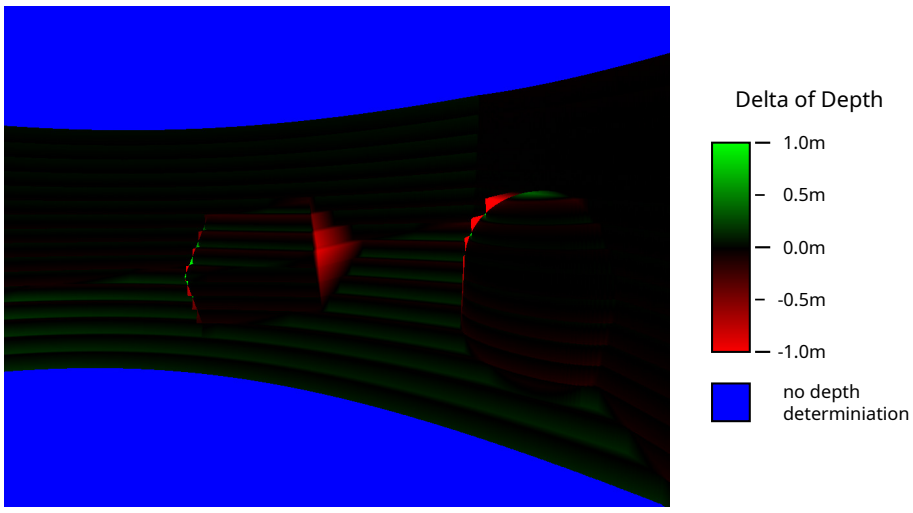


Figure 27: Error map of configuration 4 in model simplification step 0.

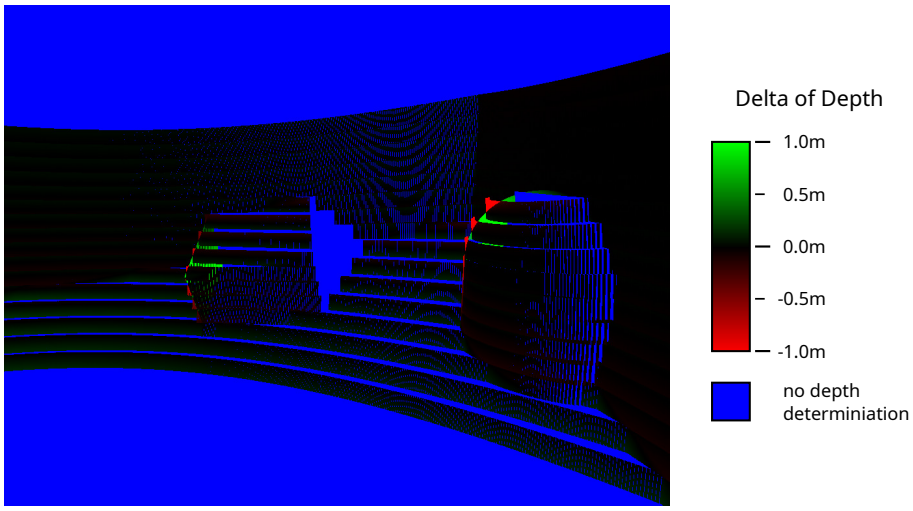


Figure 28: Error map of configuration 4 in model simplification step 1.

area to the right of the environment cube object, as it can be seen in figure 29. This happens, because pixels of this particular cube and pixels of the objects behind the cube get averaged together to a depth value. This results in the averaged laser pixels lying in between this two pixel groups. This also happens with a bigger laser pixel averaging, as applied in the two last model simplification steps.

Putting these error maps in numbers, table 5 in Appendix E: Analysis Results provides four values for each M_C_ setup. The first two values are the separate averages of the positive and negative error values. For the third value, the positive and negative values got combined with their signs, of which the average is calculated. The fourth value represents in percent, how much pixels have no depth assigned because of the current model simplification step (blue pixels in the error maps). Notice, that the standard deviation of the average values is always much bigger because of some few error values which are relatively huge regarding the average itself.

There are some other relations in the table. Looking at the first configuration, the equality of each value within the M0-M1, M2-M3 and M4-M5 pairs can be seen. This is because the camera and the laser scanner are placed at the same position, thus making the model simplification step from laser pixel volumes to planes (as within these M_ pairs) indifferent. For the other configurations, the M_ with laser pixel planes have always a higher percent of pixels with no determined depth within the named pairs, which come from the resulting gaps between these planes. Also, the amplitudes of the positive and negative errors of the M_ with laser pixel volumes are always higher than these from M_ with laser pixel planes, since there are more pixels with distances determined. The rays of these pixels are hitting the sides of these volumes, which have at the end in average a higher error value. That making this error values describing more *content* (cuboid fronts *and* sides) than the error values describing only laser pixel planes.

Also presented in Appendix E: Analysis Results, table 3 lists the average time per pixel needed for the depth calculation plus the minimum and maximum time (all in [μs]). The latter time, however, has no specific meaning in this analysis, since the ray tracing software run at a Linux operating system, where other tasks are running at the same time and thus elevating the time measurements for some short periods, which in turn is reflected in the maximum time measured. Also, the CPU clock speed plays a role here, since it can vary due thermal throttling. See section 4.3.3 for a visualization of this effect.

Also, within the time values some relations can be found. First of all, the time values for each M1, M3 and M5 simplification step are very similar for all configurations. Since all configurations for each M_ have no different setups with the laser pixel planes, the time values have to be in

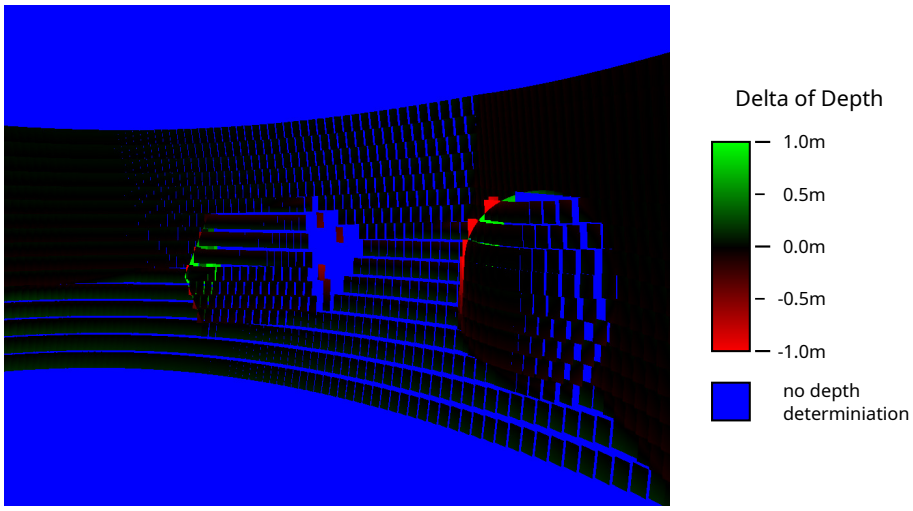


Figure 29: Error map of configuration 4 in model simplification step 3.

the same area. In the other three M_, this is only the case for the last three configurations. The reason is the in the models used laser pixel volumes. Whereas the camera of C1 sees these volumes exclusive only from the front, which triggers some conditions to skip calculations for the sides of the volumes (see section 3.2), the cameras of the other three configurations have a full view of the laser pixel volumes including their sides.

Regarding the actual model simplification steps, the significant time reductions cannot be missed. Ignoring the special camera position of the first configuration, the time reduction from the full model (M0) to the simplification with laser pixel planes inclusive merging of 20 laser pixels (M5) is nearly one hundred fold. Comparing the error values between M0 and M4[3], the average errors of measured distance are not that significantly bigger as the corresponding calculation time is smaller.

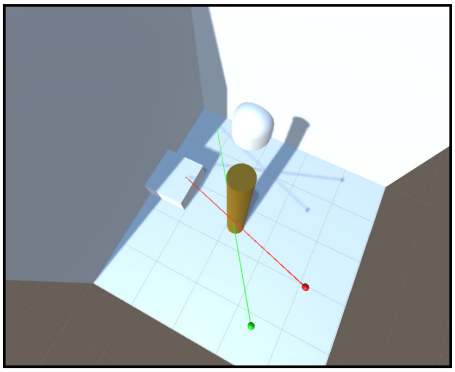
With this being said, a model simplification can be considered already if the performance is only slightly more important than the depth accuracy in the horizontal axis (which is reduced with the merging of horizontal laser pixels).

4.3.2 Covering Problem

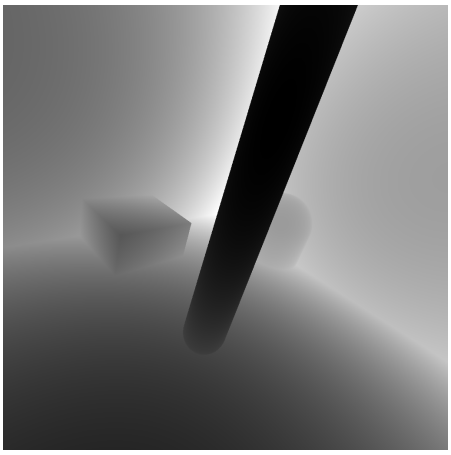
As stated earlier, a big misinterpretation of the measured environment can happen in the ray tracing model, when using laser pixel volumes. Suppose, there is a tree in the environment-sensor constellation as shown in figure 30a. The laser scanner will capture this tree, and the ray tracing model generates laser pixel cuboids out of it. Figure 30c shows the resulting model distances as a map from the view of the camera. As it can be seen, cuboids of the scanned tree get stretched into depth as defined (100 [meter]). However, these stretched cuboids cover empty space behind the tree, which does not represent the actual scanned environment. As a result, also indicated by the error map (figure 30d), there is a big area of very wrongly determined distances in the named empty space.

A possibility to counteract this is to mount an additional laser scanner to the other side of the camera, and only consider overlapping laser pixel volumes (considered with their depths). Thus if there is space from the view of one laser scanner, but a cuboid stretched into th space from the view of the other one, this cuboid's depth has to be reduced. However, this could lead to false (or

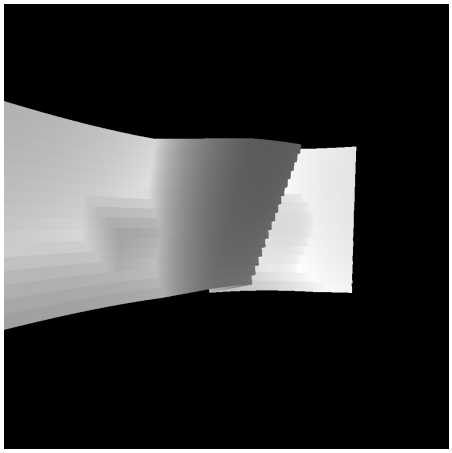
[3]Comparisons of error values between model simplification steps with laser pixel volumes and laser pixel planes have no significance – since, in place of the gaps (missing depth determination) in the plane's model, there are big error values in the volume's model. Thus making the average of the error values very different regarding the actual representing content.



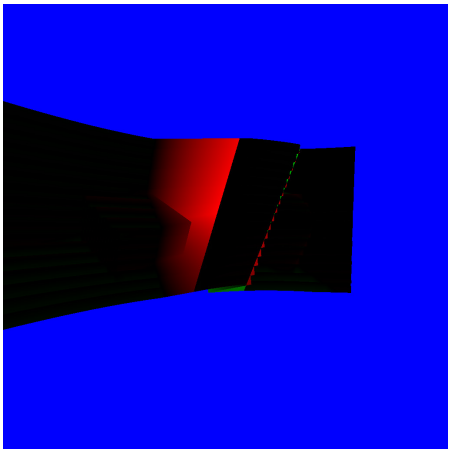
(a) Sensor-Tree constellation.



(b) True distances from the view of the camera.



(c) Depth map acquired with the ray tracing model.



(d) Error map between true distances and model depth map.

Figure 30: The covering problem. Sensor configuration C3 is used with a tree like object in the simulation environment.

oversized) space-detections and thus laser pixel depth reductions if a laser measurement point hits a small gap in the environment.

On the other hand, with realistic sensor configurations, this covering problem will not occur very often as with abnormal configurations like C3 or C4. If the sensors are placed far away like in these configurations, this ray tracing model has a high error rate anyway (see prior section 4.3.1).

4.3.3 Distribution of Calculation Time per Pixel

Table 3 in Appendix E: Analysis Results only lists the average time needed per pixel for the ray tracing model for each M_C_ setup. To give an idea of how much more or less time a pixel actually needed concerning the other pixels, a time map for each setup is generated. In this grayscale image, every pixel represents the same pixel position as in the final depth map of its setup. The brighter the pixel, the more time it needed to process the distance determination. To see structure more easily, the time values got mapped into the grayscale colors in a way, that the lowest and highest time value equals full black and full white respectively.

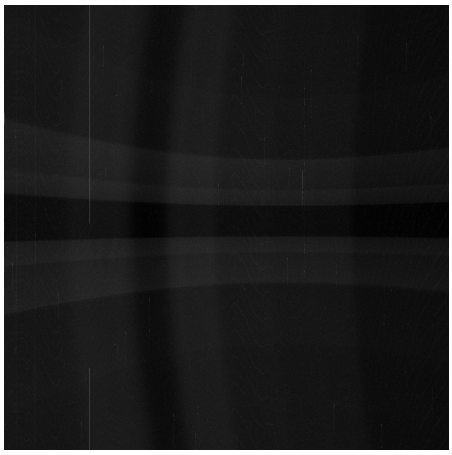
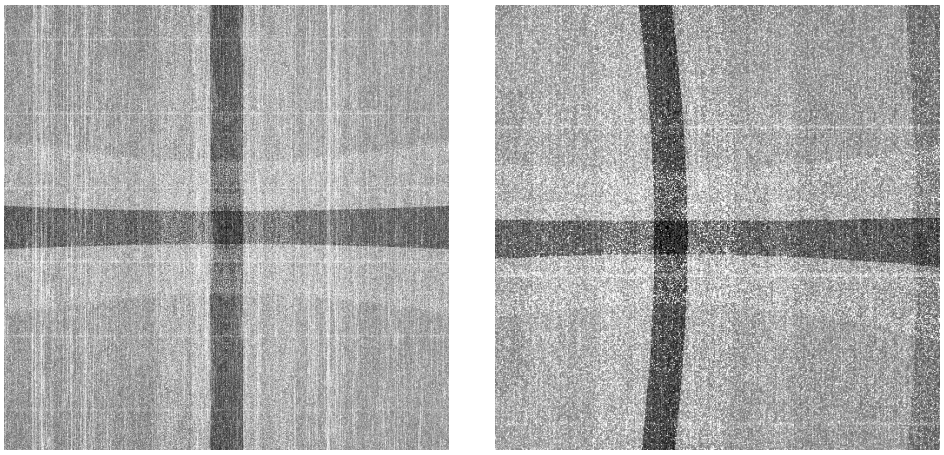


Figure 31: Time map of the M3C3 setup.

It is remarkable, that this crosslike texture as in figure 31 for the M3C3 setup can be seen in all the other ones, which have laser pixel planes included in their used model simplification step. Look in Appendix C: Time Maps for the time maps of all setups. This texture comes from the range check if a calculated t value is in the range of the laser pixel currently considered. This is done by calculating the point, which this t value is describing with its direction (equation 6), and transform these Cartesian coordinates in spherical ones, which in turn is needed for the range check of the R, Θ and Φ coordinates. The transformation to the spherical coordinate system produces this crosslike texture, since the square root and arcus tangens operations from the used C++ standard library need a characteristic time pattern to calculate the results. Note that the GNU libc version 2.17 is used on the CentOS release version 7.5.1804, since the mathematical function implementations vary by library version, processor architecture and operating system.

But also in the setups with laser pixel volumes, the C1 configuration has the same pattern. This is, because in this configuration the camera and the laser scanner are on the same position, thus triggering some breaking conditions which prevent to even produce t values for the sides of the cuboid. And this way, only positions of t values from the front of the laser pixel volume will be transformed into spherical coordinates, producing the same cross texture.

At the time maps of the other configurations with M_ having laser pixel volumes, other characteristic textures occur. This is because of the same principle mentioned above. But if looking close enough, the cross textures can still be seen, as there are always range checks of t values resulting from the fronts of the cuboids.



(a) Separately generated time map. Camera as in C1 configuration.

(b) Separately generated time map. Camera as in C2 configuration.

Figure 32: Execution of extracted code of the ray tracing model

For convincing, the detected part of code of the model got extracted to a separate independent C++ program, where only laser pixels with a distance of 4 [meters] (R coordinate) and no Θ or Φ constraints are given. Together with this, the used directions are used as the pixel coordinate as it would produce in the actual model software, and the time measured is then placed in a similar time map. The results can be seen in figure 32 with camera positions relative to the laser scanner as in the C1 and C2 configuration.

Another artifact in the time maps are the vertical white stripes. They indicate a slower calculation for some amount of time, and also do not occur systematically. The software framework with the model executed runs on a Linux distribution. With Linux being a multiuser and multiprocess operating system this software framework is plainly running at, there are other background tasks which have to be done. Therefore, these stripes within the time maps indicate a higher CPU usage at this time period, temporally degrading the available computation power and thus the performance for the ray tracing calculations.

4.4 Standard Deviation

As the software framework expects the fusion algorithm to provide a standard deviation for a returned distance, the following method is applied. The framework passes the image coordinate of a feature to the algorithm, whereas the coordinate lies in the center of a pixel. But since the pixel, which the feature relies on, is actually an area rather than a point, the whole pixel has to be considered. This can be approximated by applying the ray tracing model to each point on a, for example, 9 by 9 grid within the pixel. Out of this dataset of distances a weighted average and a weighted standard deviation are calculated, whereas the weight is defined as follows:

$$w_i = \max(0.0, 1.0 - d_i^c) \tag{12}$$

with d_i^c being the Euclidean distance of the point i to the pixel center. Both the weight definition and the size of the grid within the pixel can be adjusted as needed. The weighted average and the weighted standard deviation is then calculated as follows:

$$avg^w = \frac{\sum_i^N w_i * d_i^{\text{ray}}}{\sum_i^N w_i} \tag{13}$$

$$std^w = \sqrt{\frac{\sum_i^N w_i (d_i^{\text{ray}} - avg^w)}{(N-1)\frac{\sum_i^N w_i}{N}}} \tag{14}$$

with N being the amount of points of the pixel grid and d_i^{ray} the distance calculated through the ray tracing model for point i.

Another idea is to use the Euclidean distance of the actual hit of the pixel ray at a laser pixel cube to the center of the front of this cube within the model room. This definition of standard deviation should be more meaningful, since this center point represents a true measurement of the distance from the laser scanner in the model world. The surface of a laser pixel cube, however, is only an extrapolation trough parameters of the laser scanner, i.e. field of view and resolution. A big problem would be if the back of a stretched cuboid is hit by the camera ray, but just next to it is the front part of another cuboid. If this hit produces a close distance as the actual distance of this front part is, then the standard deviation would be too high.

But there is an additional inaccuracy to consider. The measurements of the laser scanner are not one hundred percent correct. The datasheet of the Velodyne VLP-16 multi-line laser scanner[8] names an uncertainty of 0.002 meters in each measurement. This has to be included in the calculations too. To get the likely error, if two errors are combined, the Gaussian formula states for this likely error ΔC:

$$\Delta C_{\text{likely}} = \sqrt{(\Delta A)^2 + (\Delta B)^2} \tag{15}$$

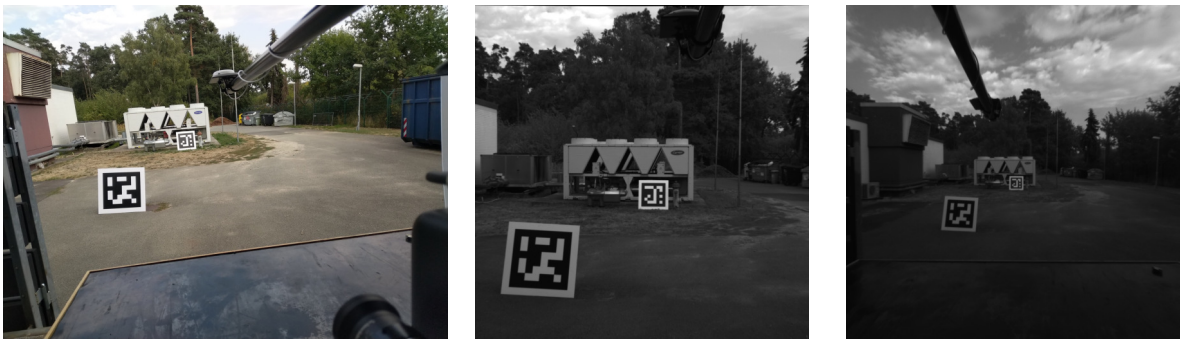
With ΔA and ΔB being the two errors or uncertainties, which are here the calculated standard deviation of the grid within the pixel and the measurement uncertainty of the laser scanner. If the maximum error for the combination of these two uncertainties is wanted instead, they just have to be summed up.

5 Experimental Results on Feature Distance Determination

Using a system consisting of the same computation hardware configuration as used in section 4.3 and the sensors as presented in section 3.1, some experiments in a real environment are made. Prior to this, the calibration of the sensor transitions is done by measuring the offsets at each Cartesian axis of the frames coordinate system. The relative rotation of each axis is presumed to be zero on each axis, which is a 3 by 3 identity matrix as a rotation matrix, as the directions of view of the sensors are exactly the same. This was done because there was no calibration available at the time; on the other hand, however, the rack is manufactured by computer numerical control (CNC) procedures and thus the sensors are mounted with high precision also by means of their orientations.

To make statistically more precise statements, several setups were tested. They consist of the test field and two AprilTags which are placed in different locations with each setup. For each tag, the software framework pass image coordinates in pixels, which represents the center of the tag within the image. Through the ray tracing model of this thesis, the distance from the camera is determined. These calculated distances are validated by a handheld laser rangefinder (LRF). Although several setups were tested, only a few are discussed. The measurements, statistical values and camera views for every setup can be found in Appendix D: Experiment Setups and Appendix F: Experiment Setups Statistics. Data is generated by running the software framework with the ray tracing algorithm for around 5 seconds at 10Hz and calculating the weighted average and weighted standard deviation of the datasets for each combination of left and right camera and tag one and tag two. The weights are defined as:

$$w_i = \frac{std^{\min}}{std_i^w} \tag{16}$$



(a) Test field with tags. **(b)** View of left sensor camera. **(c)** View of right sensor camera.

Figure 33: Experiment setup 1. Tag one on the front left and tag two on the back right.

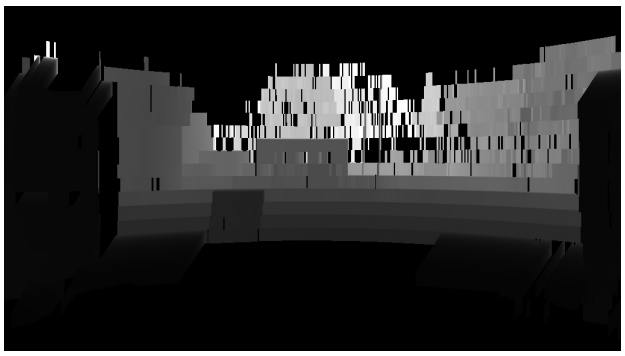


Figure 34: Depth map in the view of the right camera in the model world of experiment setup 1.

with std_i^w being the weighted standard deviation of the distance calculation i after equation 14 and $std^{\min}$ being the smallest possible standard deviation calculated in section 4.4, which is the uncertainty of the laser scanner. Because this value is fixed, whereas the standard deviation for each distance (equation 14) can be zero, the combination of std_i^w and the uncertainty value after equation 15 will bring the same value as the uncertainty value itself, consequently being the minimal standard deviation possible. The weighted average and the weighted standard deviation is then calculated as in the equations 13 and 14.

Figure 33 shows a photo of the first setup including the view of both sensor cameras. The left camera has a FOV of 40° by 40° and the right one a FOV of 80° by 80° with the closer tag being number one and the more distant tag number two. Figure 34 shows the depth map in view of of the right camera in the model world. Table 2 shows the distances of the two tags from both cameras with the difference (error) to the distance measured by the LRF and this measured value itself.

Note that the distance measured with the LRF is a very rough measurement (as captured in figure 35) since it is held with bare hands and also because of the unknown position of the actual

Tag	Camera	Distance Model	Standard Deviation	Delta from LRF	Distance LRF
T1	left	7.627	± 0.005	-0.011	7.638
	right	7.656	± 0.004	0.018	
T2	left	17.606	± 0.0106	-0.061	17.667
	right	17.602	± 0.008	-0.065	

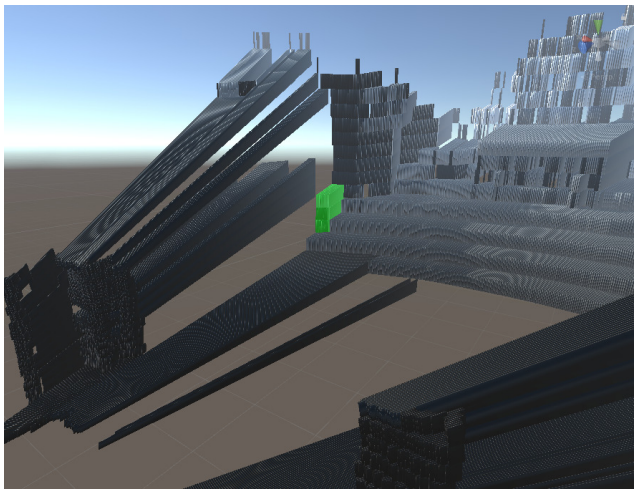
Table 2: Statistics for experiment setup 1. Values in [meter].

Figure 35: LRF next to the left camera sensor.

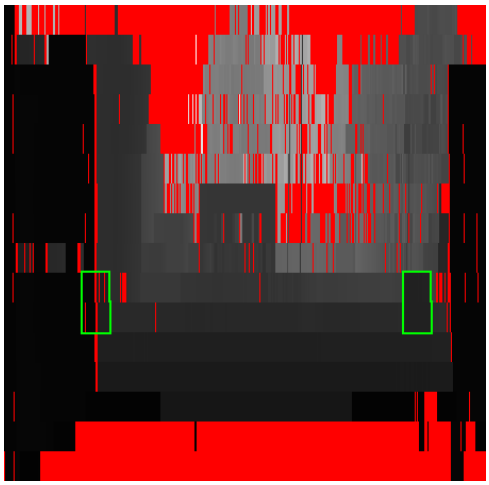
camera sensor. This hypothetic sensor position can vary through the used camera lens and the focus settings and it is difficult to estimate it.

In setup 6 and setup 7 the tags were placed at the edge of the view of the left and the right sensor camera, respectively, to check limits of the whole detection system. Since the left camera has a more narrow FOV than the right one, both tags were detected correctly in both cameras in setup 6. As it can be seen in table 6 in Appendix F: Experiment Setups Statistics, in setup 7 there were no values to process. Only the distances from the LRF are available. For the left, camera it was not possible to gain distance values from the ray tracing model, since the tags were placed at the edges of the FOV of the right camera and thus are out of view of the left one. The reason for the missing distance values of the right camera is, that the tag placed on the right was placed too far to the image border so that the feature detection algorithm could not detect the AprilTag. In figure 36a it is noticeable, that the right tag is placed too far to the right so that the right white border is not visible anymore and thus the detection of the AprilTag does not work properly. The missing distance values for the left tag have another reason, since it is also in the view of the feature tracker. Figure 36c shows the view of the laser scanner, where a pixel is representing one laser point. The laser pixels got horizontally stretched by the factor 28 to get an almost quadratic picture. Note that the FOV of the laser scanner is 90° by 30°, thus making quadratic objects not quadratic in the picture at all. The brighter the pixel the greater was the distant measured, whereas the red areas have no distances available. The left green marked area indicates all the measurement points for the left tag. As it can be seen, there are unavailable distances within this tag area, which is the reason for the unavailability of values in table 6. The darker left half of this tag is from another object standing in between the laser scanner and the tag itself. Since the measurement period for each setup was around 5 seconds, this is a systematic flaw. Reasons for this could be an internal failure at this angle or dirt on the protective glass of the laser scanner hull. But it could also because of the object in front of the tag, which creates this gap along its right side. This can happen if e.g. the edge of the object is made of metal, which reflects the laser beams and distorts the actual distance measurement. Figure 36b shows a 3D rendering of the model world of one scan of setup 7, with the look at the left tag. In this model world, the laser pixel cuboids are only as depth as the most distant direct nearest neighbor is away as described in section 3.2, which in other words consider only a 3 by 3 field of laser scans around the current cuboid. To go against such gaps as it occurs in setup 7, instead a 5 by 5 field of laser measurement could be considered to further extend the depth of the laser pixel cuboid. This would close one-pixel gaps, as long the camera pixel ray does not hit exactly the small gap in the back. The schematic in figure 36d shows the problem visually. The red line shows the camera pixel ray,

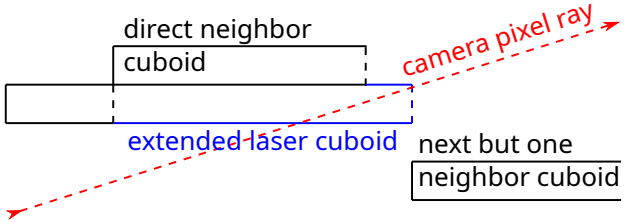
(a) View of right sensor camera.



(b) Rendering of the model world with look on the left tag marked in green.



(c) View of laser scanner.



(d) Simplified schematic of the gap problem.

Figure 36: Experiment setup 7.

which only hits a near cuboid if the blue marked cuboid got extended this way be derive its depth from the next but one neighbor cuboid instead of the direct one.

There are also missing values for setup 8 and setup 9, but this time it is about the distances of the LRF. All setup tests are done in daylight. Since the LRF has only a class 2 laser for its distance measurement, after tags standing about 20 meters away and without a tripod it is very difficult to hit the tag surface with the laser by hand. Also, at this distance, the laser is too weak at daylight the LRF to read the laser echo for its measurement process. The laser scanner, however, could measure these distances. In fact, the maximum distance measured by the laser scanner is close to 70 meters. This value is gathered from the raw laser scanner values, not by using tags.

In all setups, but especially in the last setup (S9), a noise of the distances from the ray tracing model over time was clearly observable. Also, table 6 in Appendix F: Experiment Setups Statistics states standard deviations of about 3 to 10 centimeters for all setups. One possible source of this could be the actual jitter of the laser scanner range measurements, which is fed directly in the ray tracing model and thus used by the tag distance determination. Figure 37 shows the view of the laser scanner of setup 9 with each pixel representing a laser point, but with the variance over time as value instead of an actual measured distance. The brighter the pixel, the higher the jitter was

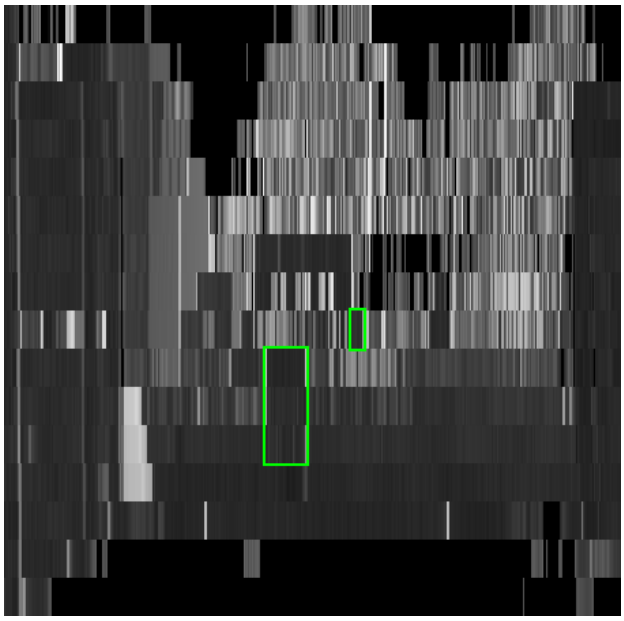
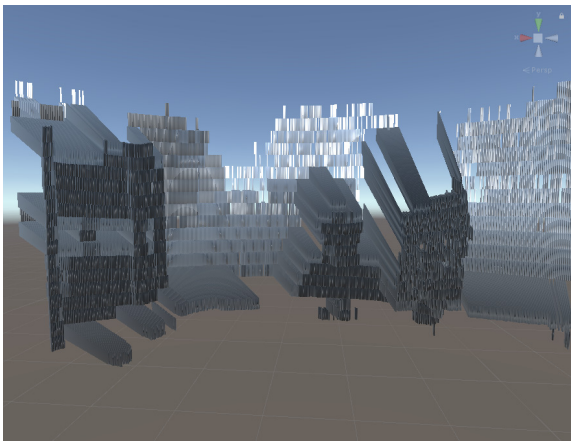


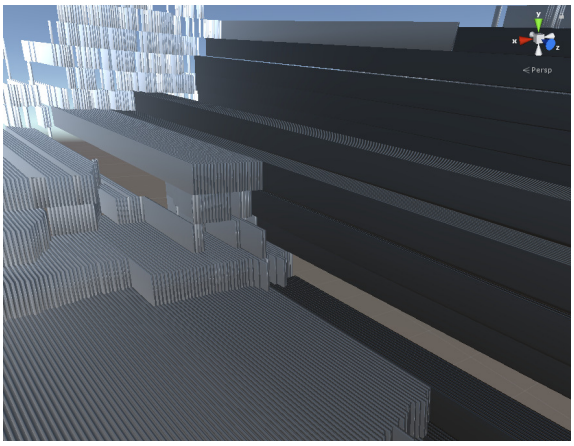
Figure 37: Jitter of experiment setup 9 within the 5 seconds measurement time period. Boxes represent the tags.

over the 5 seconds of the measurement. The tags are marked. The laser pixels got horizontally stretched by the factor 28 to get an almost quadratic picture. Note that the FOV of the laser scanner is 90° by 30°, thus making quadratic objects not quadratic in the picture at all. The brightest pixels equals a jitter, or standard deviation over time, of 29.448 [meter]. All other values between zero and this maximum jitter are logarithmically scaled. The front tag has an average jitter of around 58.112 [meter] and the back tag of around 50.428 [meter]. These values are from the laser scanner itself. Additionally to this, especially to for longer time periods, expansions and subtractions of the mounting rack because of temperature variations can occur, producing very small indifferences between the sensors orientations. These can also emerge from vibrations from e.g. rotors or other shock sources.

Figure 38: View of left sensor camera with occurring covering problem.



(a) Rendering of the model world with look from the left sensor camera.



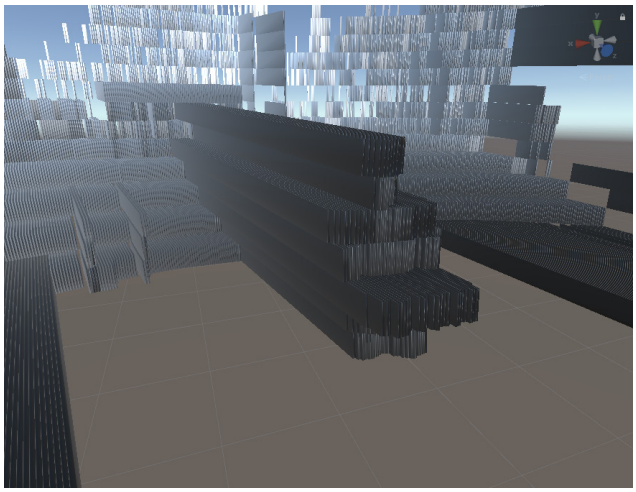
(b) Rendering of the model world with look on the tag, which is not visible due covering by the laser cuboids of the person.

Figure 39: Model world rendering of the occurred covering problem.

While testing the setups, the prior discussed covering problem (section 4.3.2) occurred. Figure 38 shows this occurrence in the view of the left sensor camera while a person standing in between the laser scanner and a tag, which is placed about 17.6 [meter] away from the sensor configuration (see 6 in the appendix at setup 1 tag 2). With the model world rendered at this state, figure 39b shows, that the person in front of the laser scanner, or its laser pixel cuboids, completely cover the tag. This is making the camera pixel ray of the center of the tag hit these cuboids of the person rather than (hypothetical) cuboids of the tag itself. The calculated distance from the ray tracing model is 2.255 ± 0.014 [meter]. Note, that the standard deviation is very small in terms of the big misinterpretation of the distance of the tag. This is because the calculation of this standard deviation does not consider the distance of the hit to the actual measurement point of the hit cuboid, as it is stated in section 4.4. Implementing this consideration, the standard deviation would be very high in this covering problem case.

Also, the "step problem" described in section 4.3 is tested. It states, that a cuboid is vertically too big due the low vertical resolution. Because of this, either a laser pixel can cover a point which lies behind its front in the real world, or this front lies too far away than a corresponding point in the real world would be. This is tested by using a relatively small tag. It has to be big enough, that the camera resolution allows the AprilTag recognition detect this tag, but also it has to be smaller than the high of a cuboid at the distance the tag is held from the laser scanner. This tag is then slowly moved from top to bottom. Figure 40a shows finally the occurrence of the step problem. At this time point, the tag was about 4.254 [meter] away from the right sensor camera and the point lying behind it, which is the body of the person holding the tag, 4.603 [meter] away from this camera. This image frame was captured by the right camera, from which the distance of the detected tag center is mistakenly calculated as the 4.603 [meter] by the ray tracing model. The model world rendering in figure 40b shows that only one row of laser scan cuboids has the tag detected (and hands of the person). In this case, the center of the tag in the image frame was either barely above or below this row, so that the camera pixel ray hit the cuboids of the body of the person.

(a) View of the right sensor camera.



(b) Rendering of the model world with look at the person holding the tag.

Figure 40: Forces occurring of the step problem.

6 Conclusions and Future Directions

A fusion model for visual image data and laser range data has been developed, of which with passing an image coordinate a distance can be returned as long the resulting ray hits the laser range data in the ray tracing model. This algorithm has been implemented into two environments, the simulation and the software framework. Several different setups in the simulation and later in the experimental tests have been considered and discussed. Also, analyzes for the performance of the algorithm in the software framework, as well as differences (errors) between true distances from the camera and estimated distances by the fusion model, have been done.

AprilTags are used in this thesis to detect as features. Of course, other methods can be used to e.g. detect corners of an object for avoidance. The only needs the fusion model has are the FOV of camera and laser scanner, their resolutions and the image coordinate of the image feature of interest.

Different ways to calibrate the relative transition and rotation between the sensor cameras and the laser scanner have been presented. Future research should contain the actual execution of such a calibration prior to the experimental tests to make more accurate statements, instead of assuming the relative rotations to be specific ones due CNC manufactured mounting racks.

Not only returning the distance alone, different ways to calculate the standard deviation regarding a specifically returned distance have been discussed. Each way has its simplicity, but also reasons to use it or not. Depending on how the actual usage of the fusion model is, the calculation of the standard deviation has to be adjusted.

Since a laser ray of the laser scanner itself will expand in diameter over distance, the actual first hit of this ray on a surface of an object is not compelled to be the center of the ray. In the ray tracing model, however, the measured distance of by a laser ray is considered to be such a center point. This issue should be integrated into the standard deviation too.

Either way, the fusion model presented in this thesis is a raw fusion itself. This means, there are no error correction or data filtering done. By using raw data and blindly using the resulting estimated distances air dust or reflections of windows can bring great problems, just to name a few. Also, such error cases as the “step problem” (section 4.3 and 5) or the “covering problem” (section 4.3.2) can still bring misleading distances from the fusion model.

Abbreviations

ARTIS Autonomous Rotorcraft Test bed for Intelligent Systems. 8, 19

ATON Autonomous Terrain-based Optical Navigation. 19

CAD computer-aided design. 8

CNC computer numerical control. 31, 37

DLR German Aerospace Center. 19

FOV field of view. 8, 16, 18, 32, 33, 35, 37

IMU inertial measurement unit. 14–16

LiDAR Light Detection and Ranging. 5

LRF laser rangefinder. 31–34, 59

MATLAB matrix laboratory. 15

MDSD Model-Driven Software Development. 20, 21

RODOS Realtime Onboard Dependable Operating System. 19

SysML Systems Modeling Language. 20, 21

UML Unified Modeling Language. 20, 21

References

[1] Denis Klimentjew, N Hendrich, and Jianwei Zhang. Multi sensor fusion of camera and 3d laser range finder for object recognition. pages 236–241, October 2010.

[2] Eric K Forkuo and Bruce King. Automatic fusion of photogrammetric imagery and laser scanner point clouds. *International Archives of Photogrammetry and Remote Sensing*, 35 (2004):921–926, 2004.

[3] Christoph Stiller, Jochen Hipp, C Rössig, and A Ewald. Multisensor obstacle detection and tracking. *Image and vision Computing*, 18(5):389–396, 2000.

[4] Jan C Becker, Andreas Simon, Ina Söhnitz, Harald Göllinger, and Walter Schumacher. A decentralized path planning and control structure for an autonomous vehicle. 1998.

[5] H Baltzakis, Antonis Argyros, and Panos Trahanias. Fusion of range and visual data for the extraction of scene structure information. In *Proceedings - International Conference on Pattern Recognition*, volume 4, pages 7–11 vol.4, 02 2002. ISBN 0-7695-1695-X.

[6] Maria C. Garcia-Alegre, David Martin Gomez, D. Miguel Guinea, and Domingo Guinea. Real-time fusion of visual images and laser data images for safe navigation in outdoor environments. In Ciza Thomas, editor, *Sensor Fusion*, chapter 7. InTech, Rijeka, 2011. doi: 10.5772/16690. URL `https://doi.org/10.5772/16690`.

[7] Allied Vision. Avt prosilica gt 1380 product page. URL `https://www.alliedvision.com/de/produkte/kameras/kameradetails/Prosilica%20GT/1380.html`. [accessed 04-May-2018].

[8] Velodyne LiDAR. Velodyne vlp-16 product page. URL `http://velodynelidar.com/vlp-16.html`. [accessed 04-May-2018].

[9] Unity Technologies. Unity game engine. URL `https://unity3d.com/`. [accessed 02-May-2018].

[10] S. Montenegro and F. Dannemann. Rodos - real time kernel design for dependability. In *DASIA 2009 - DAta Systems in Aerospace*, volume 669 of *ESA Special Publication*, page 66, May 2009.

[11] Olaf Maibaum, Daniel Lüdtke, and Andreas Gerndt. Tasking framework: Parallelization of computations in onboard control systems. In *ITG/GI Fachgruppentreffen Betriebssysteme*, November 2013. URL `http://elib.dlr.de/87505/`. http://www.betriebssysteme.org/Aktivitaeten/Treffen/2013-Berlin/Programm/.

[12] The CentOS Project. Centos linux operating system. URL `https://www.centos.org/`. [accessed 01-June-2018].

[13] Stephan Theil, Nikolaus Alexander Ammann, Franz Andert, Tobias Franz, Hans Krüger, Hannah Lehner, Martin Lingenauber, Daniel Lüdtke, Bolko Maass, Carsten Paproth, and Jürgen Wohlfeil. Aton (autonomous terrain-based optical navigation) for exploration missions: recent flight test results. *CEAS Space Journal*, March 2018. URL `http://elib.dlr.de/119557/`.

[14] Andreas Rittweger, Stephan Theil, Marco Scharringhausen, and René Schwarz. Dlr institute of space systems: Status report 2007–2016, part i. Technical report, DLR Institute of Space Systems, October 2016. URL `http://elib.dlr.de/105683/`.

[15] T. Franz, D. Lüdtke, O. Maibaum, and A. Gerndt. Model-based software engineering for an optical navigation system for spacecraft. *CEAS Space Journal*, 10(2):147–156, Jun 2018. ISSN 1868-2510. doi: 10.1007/s12567-017-0173-5. URL `https://doi.org/10.1007/s12567-017-0173-5`.

[16] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, May 2011.

[17] John Wang and Edwin Olson. AprilTag 2: Efficient and robust fiducial detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[18] ADL Embedded Solutions. Adlqm87pc product page. URL `https://www.adl-usa.com/product/adlqm87pc/`. [accessed 22-June-2018].

[19] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962. ISSN 00401706. URL `http://www.jstor.org/stable/1266577`.

[20] Ziqi Chai, Yuxin Sun, and Zhenhua Xiong. A novel method for lidar camera calibration by plane fitting. Jul 2018.

[21] Abdallah Kassir and Thierry Peynot. Reliable automatic camera-laser calibration. 09 2018.

[22] Qilong Zhang and R. Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2301–2306 vol.3, Sept 2004. doi: 10.1109/IROS.2004.1389752.

[23] S. Krause and R. Evert. Remission based improvement of extrinsic parameter calibration of camera and laser scanner. In *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 829–834, Dec 2012. doi: 10.1109/ICARCV.2012.6485265.

[24] D. Bi and Xiao Lu. A new flexible approach for single laser stripe profiler calibration. In *2008 International Conference on Information and Automation*, pages 76–80, June 2008. doi: 10.1109/ICINFA.2008.4607971.

[25] D. Scaramuzza, A. Harati, and R. Siegwart. Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4164–4169, Oct 2007. doi: 10.1109/IROS.2007.4399276.

[26] H. Aliakbarpour, P. Nunez, J. Prado, K. Khoshhal, and J. Dias. An efficient algorithm for extrinsic calibration between a 3d laser range finder and a stereo camera for surveillance. In *2009 International Conference on Advanced Robotics*, pages 1–6, June 2009.

[27] Thierry Peynot and Abdallah Kassir. Laser-camera data discrepancies and reliable perception in outdoor robotics. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2625–2632. IEEE, 2010.

[28] P Rönnholm and H Haggrén. Registration of laser scanning point clouds and aerial images using either artificial or natural tie features. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci*, pages 63–68, 2012.

[29] Jeroen D. Hol, Thomas B. Schön, and Fredrik Gustafsson. Modeling and calibration of inertial and vision sensors. *The International Journal of Robotics Research*, 29(2-3):231–244, 2010. doi: 10.1177/0278364909356812. URL https://doi.org/10.1177/0278364909356812.

[30] M. Kleinert and U. Stilla. Camera-imu calibration using a tilted calibration board. In *34. Wissenschaftlich-Technische Jahrestagung der DGPF, 62. Deutscher Kartographentag der DGfK und Geoinformatik 2014 der GfGI und des GIN, Hamburg, 26.-28. März 2014*, 2014.

[31] J. Rehder and R. Siegwart. Camera/imu calibration revisited. *IEEE Sensors Journal*, 17(11): 3257–3268, June 2017. ISSN 1530-437X. doi: 10.1109/JSEN.2017.2674307.

[32] Mingyang Li and Anastasios I. Mourikis. Online temporal calibration for camera–imu systems: Theory and algorithms. *The International Journal of Robotics Research*, 33(7):947–964, 2014. doi: 10.1177/0278364913515286. URL https://doi.org/10.1177/0278364913515286.

[33] Jorge Lobo and Jorge Dias. Relative pose calibration between visual and inertial sensors. *The International Journal of Robotics Research*, 26(6):561–575, 2007. doi: 10.1177/0278364907079276. URL https://doi.org/10.1177/0278364907079276.

[34] VJean-Yves Bouguet. Camera calibration toolbox for matlab. URL http://www.vision.caltech.edu/bouguetj/calib_doc/index.html. [accessed 02-September-2018].

[35] J Talaya, R Alamus, Ernest Bosch, Albert Serra Pagès, W Kornus, and Anna Baron. Integration of a terrestrial laser scanner with gps/imu orientation sensors. 35, 07 2004.

[36] Marco Baglietto, Antonio Sgorbissa, Damiano Verda, and Renato Zaccaria. Human navigation and mapping with a 6dof imu and a laser scanner. 59:1060–1069, 12 2011.

[37] Tahir Rabbani and Frank Van Den Heuvel. Efficient hough transform for automatic detection of cylinders in point clouds. 36, 01 2005.

[38] David Eberly. *Fitting 3D Data with a Cylinder*. Geometric Tools, August 2015. URL www.geometrictools.com.

[39] J. Farrell, J. Stuelpnagel, R. Wessner, J. Velman, and J. Brook. A least squares estimate of satellite attitude (grace wahba). *SIAM Review*, 8(3):384–386, 1966. doi: 10.1137/1008080. URL https://doi.org/10.1137/1008080.

[40] James R. Wertz. *Three-Axis Attitude Determination Methods*, pages 410–435. Springer Netherlands, Dordrecht, 1978. ISBN 978-94-009-9907-7. doi: 10.1007/978-94-009-9907-7_12. URL https://doi.org/10.1007/978-94-009-9907-7_12.

[41] M. D. SHUSTER and S. D. OH. Three-axis attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*, 4(1):70–77, Jan 1981. ISSN 0731-5090. doi: 10.2514/3.19717. URL https://doi.org/10.2514/3.19717.

Appendix A: Depth Maps

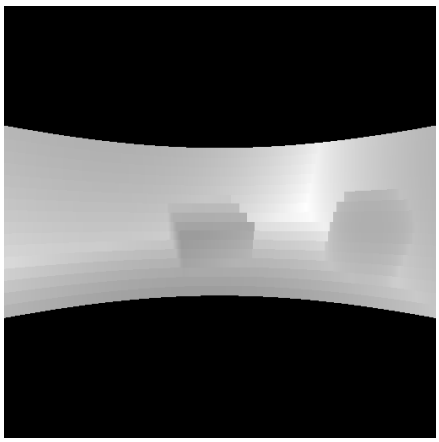


Figure 41: Depth map of M0C1

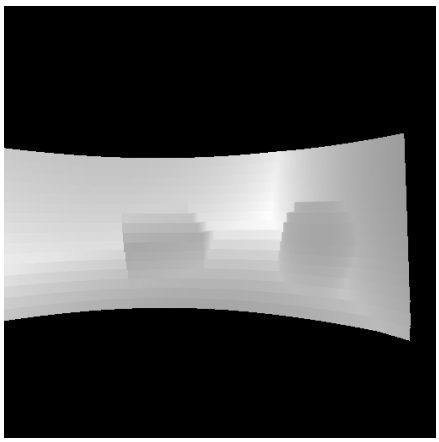


Figure 42: Depth map of M0C2

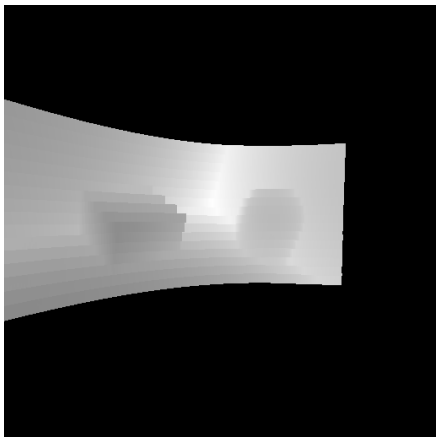


Figure 43: Depth map of M0C3

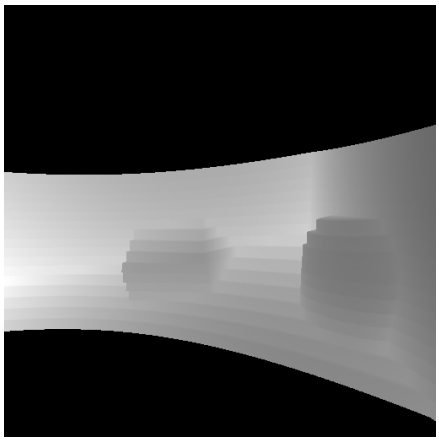


Figure 44: Depth map of M0C4

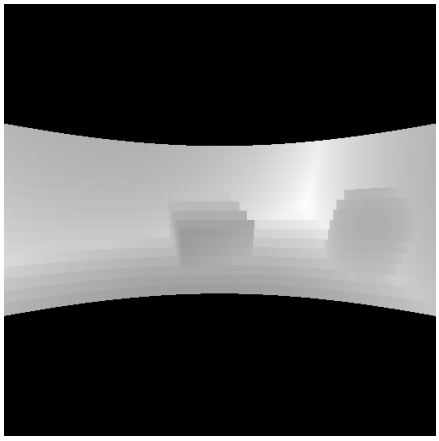


Figure 45: Depth map of M1C1

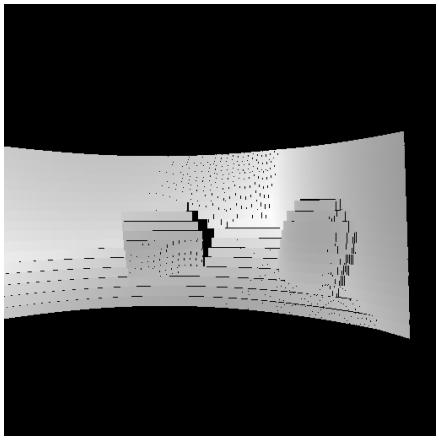


Figure 46: Depth map of M1C2

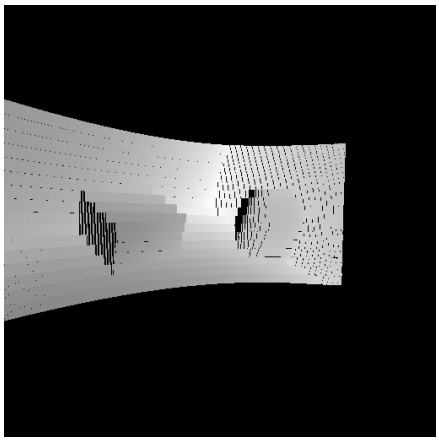


Figure 47: Depth map of M1C3

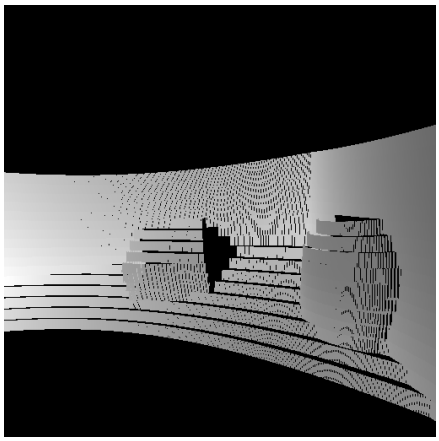


Figure 48: Depth map of M1C4

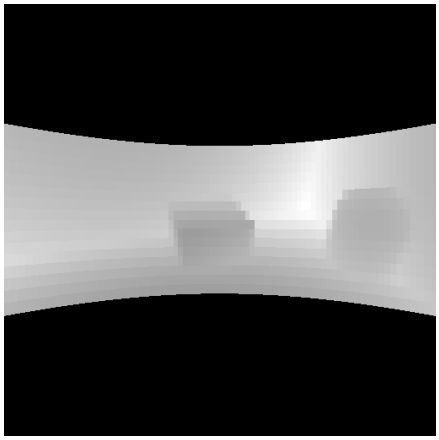


Figure 49: Depth map of M2C1

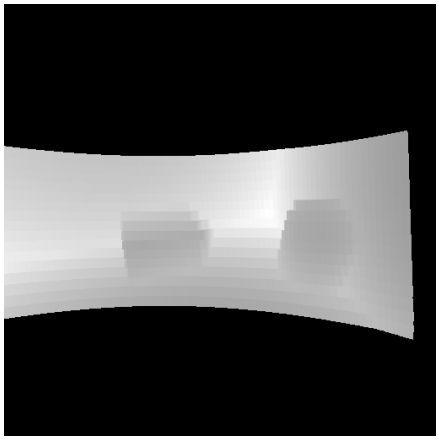


Figure 50: Depth map of M2C2

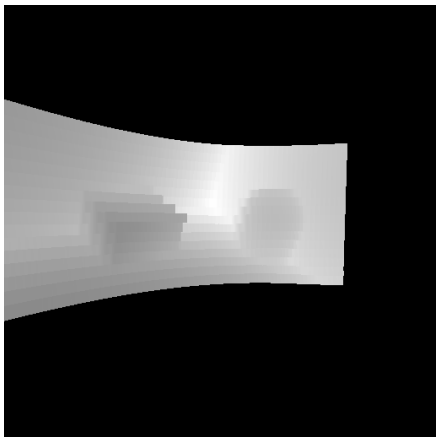


Figure 51: Depth map of M2C3

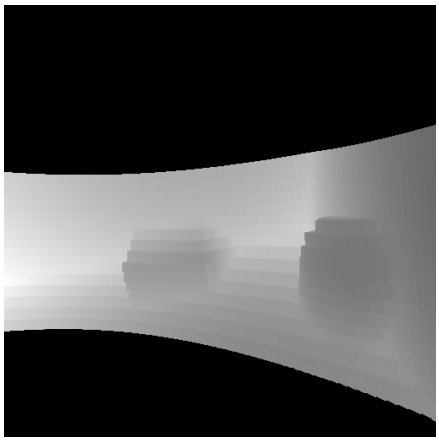


Figure 52: Depth map of M2C4

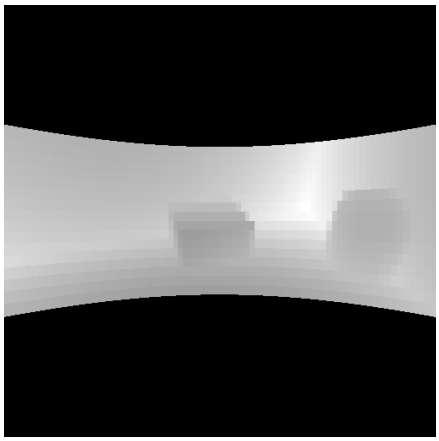


Figure 53: Depth map of M3C1

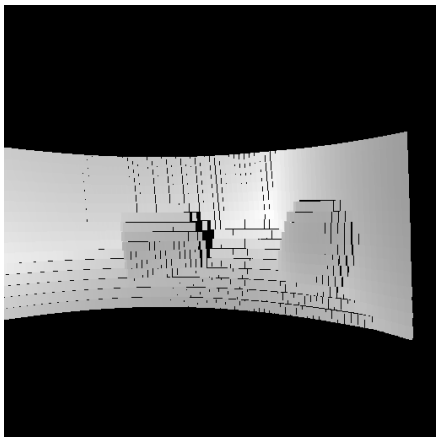


Figure 54: Depth map of M3C2

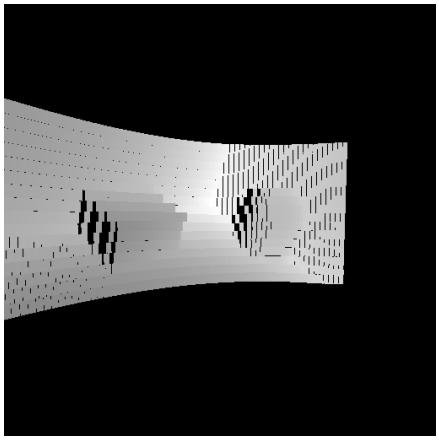


Figure 55: Depth map of M3C3

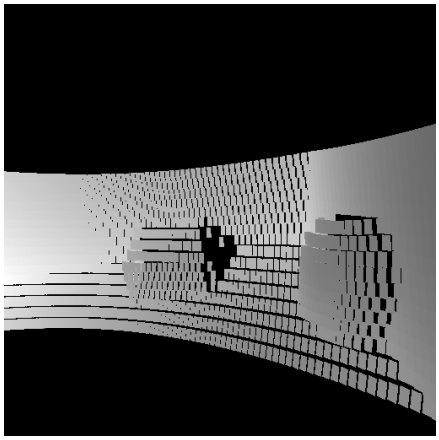


Figure 56: Depth map of M3C4

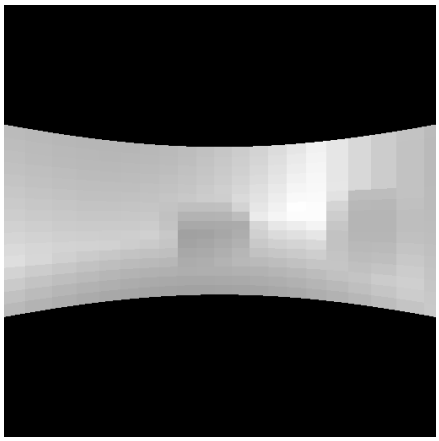


Figure 57: Depth map of M4C1

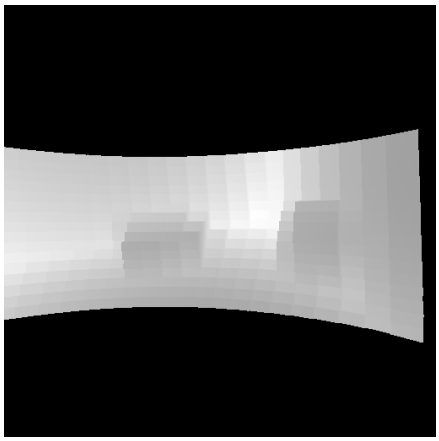


Figure 58: Depth map of M4C2

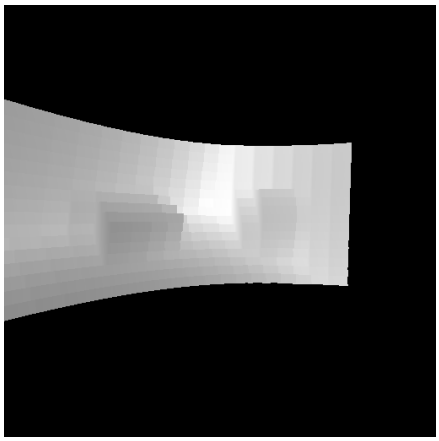


Figure 59: Depth map of M4C3

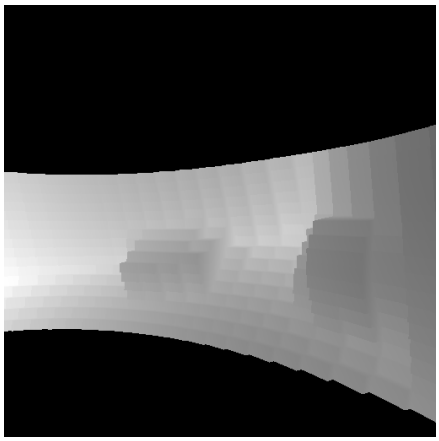


Figure 60: Depth map of M4C4

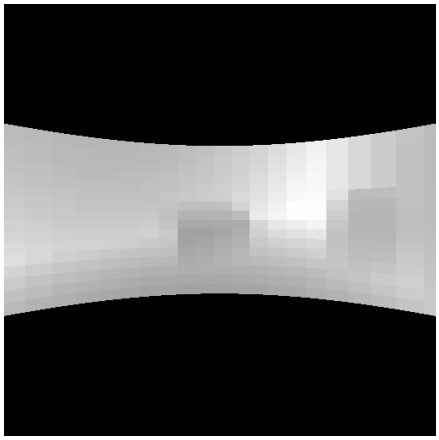


Figure 61: Depth map of M5C1

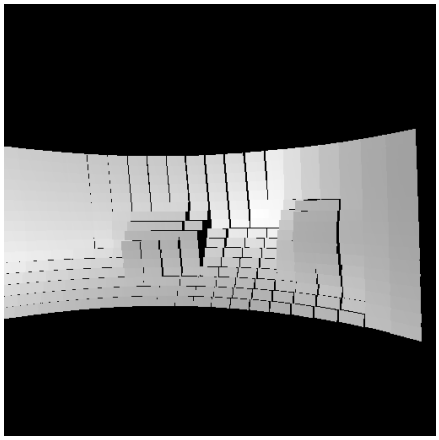


Figure 62: Depth map of M5C2

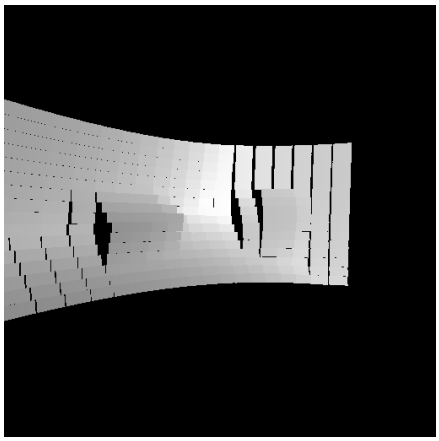


Figure 63: Depth map of M5C3

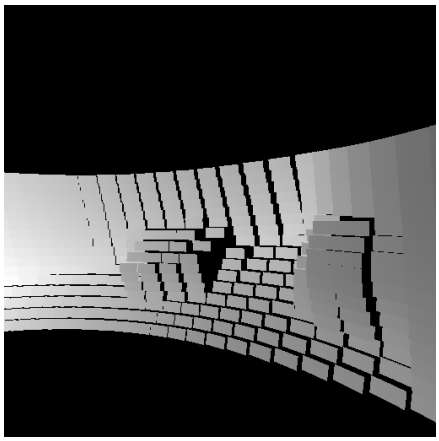


Figure 64: Depth map of M5C4

Appendix B: Error Maps

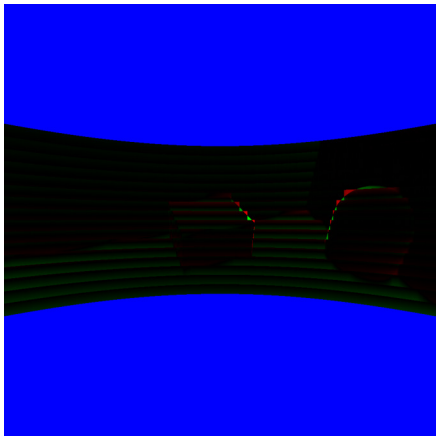


Figure 65: Error map of M0C1

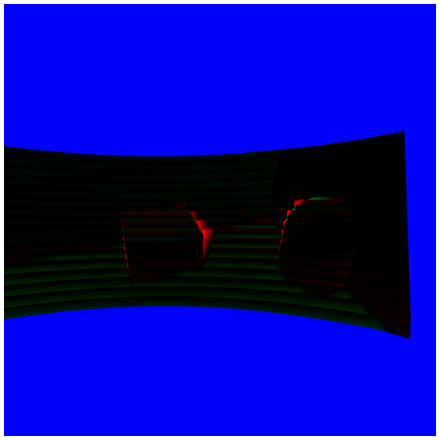


Figure 66: Error map of M0C2

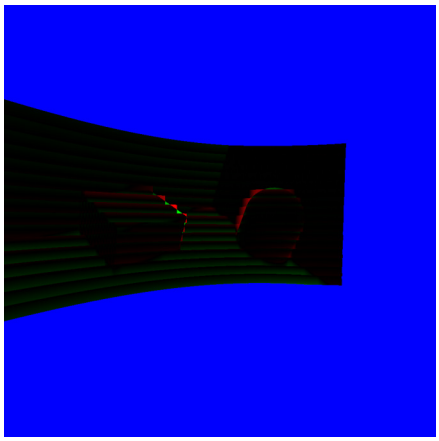


Figure 67: Error map of M0C3

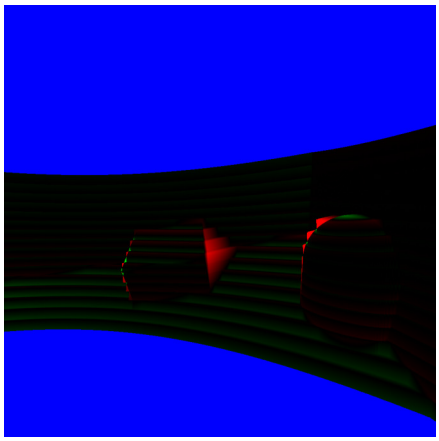


Figure 68: Error map of M0C4

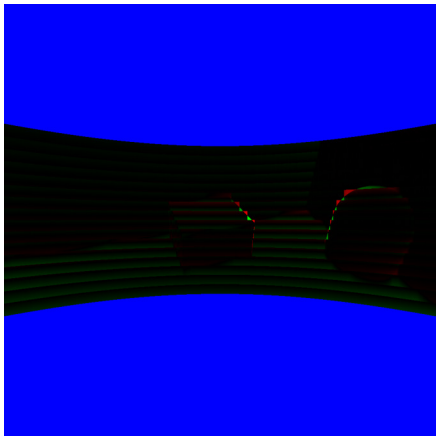


Figure 69: Error map of M1C1

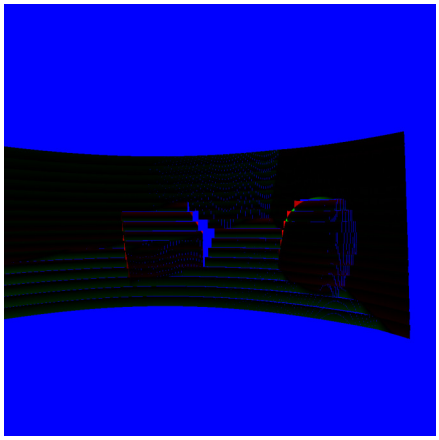


Figure 70: Error map of M1C2

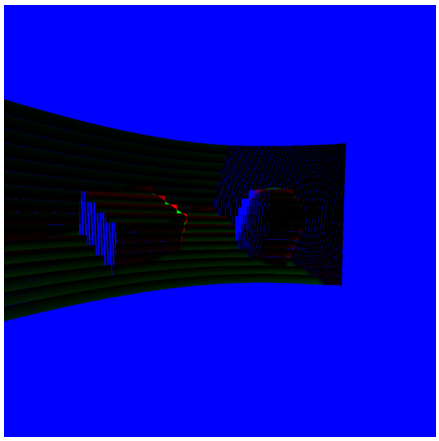


Figure 71: Error map of M1C3

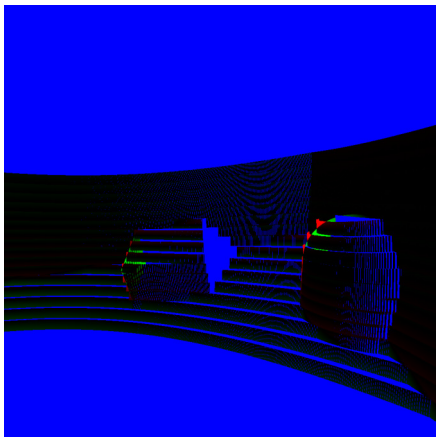


Figure 72: Error map of M1C4

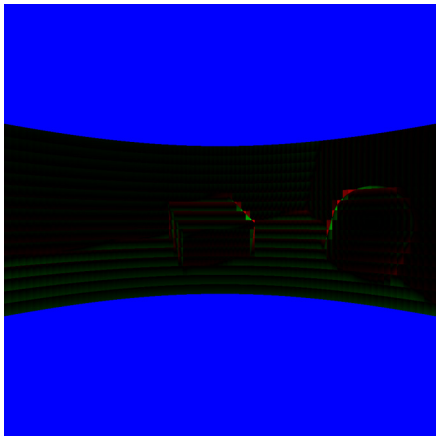


Figure 73: Error map of M2C1

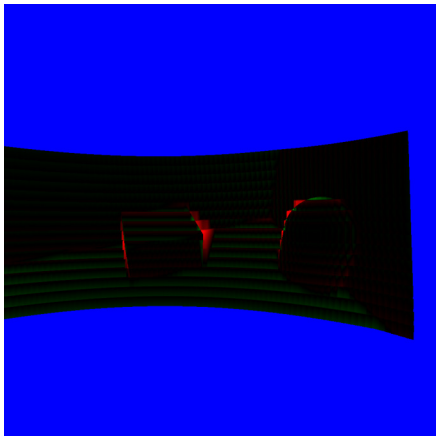


Figure 74: Error map of M2C2

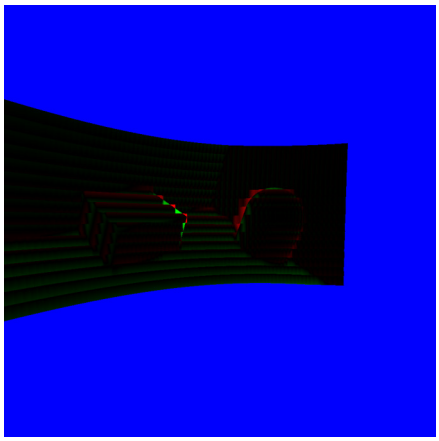


Figure 75: Error map of M2C3

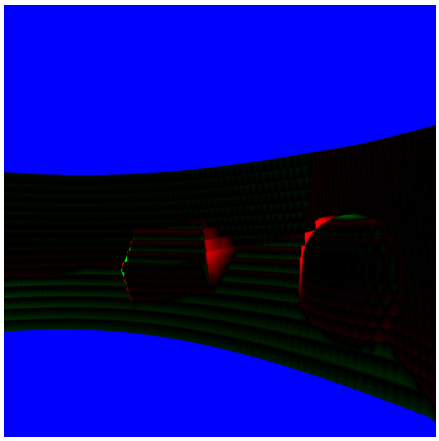


Figure 76: Error map of M2C4

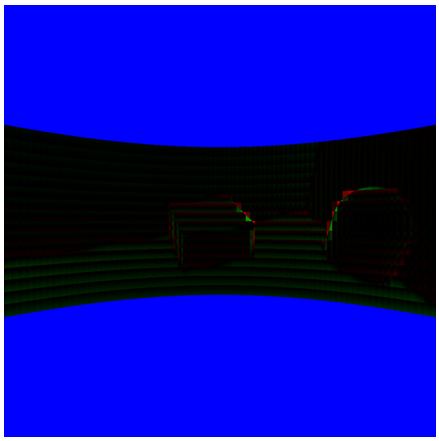


Figure 77: Error map of M3C1

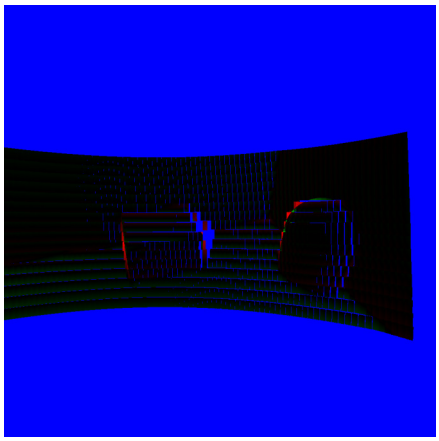


Figure 78: Error map of M3C2

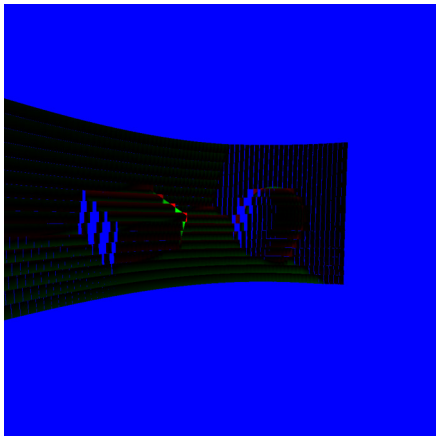


Figure 79: Error map of M3C3

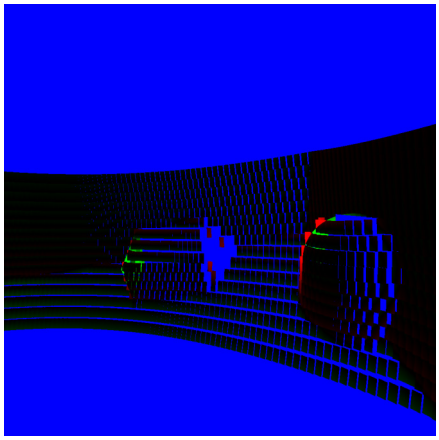


Figure 80: Error map of M3C4

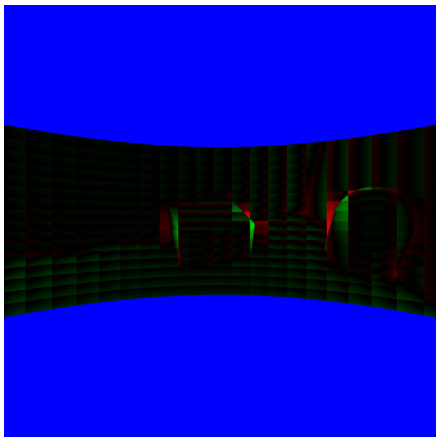


Figure 81: Error map of M4C1

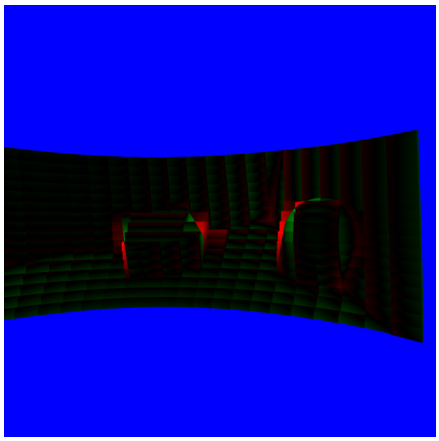


Figure 82: Error map of M4C2

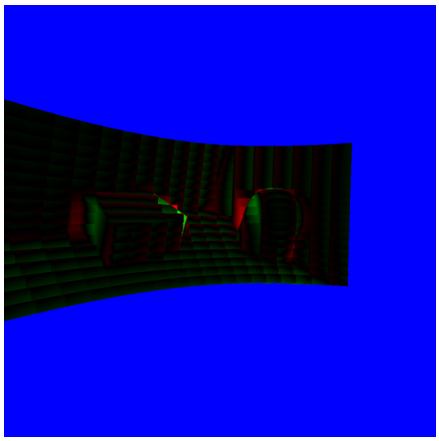


Figure 83: Error map of M4C3

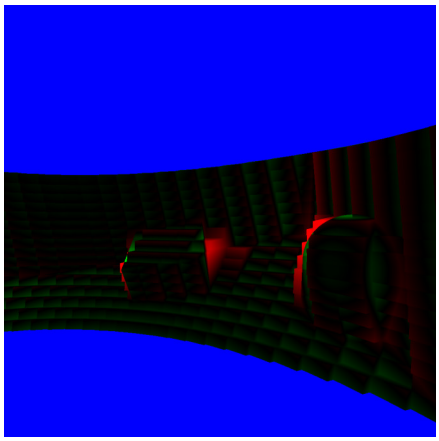


Figure 84: Error map of M4C4

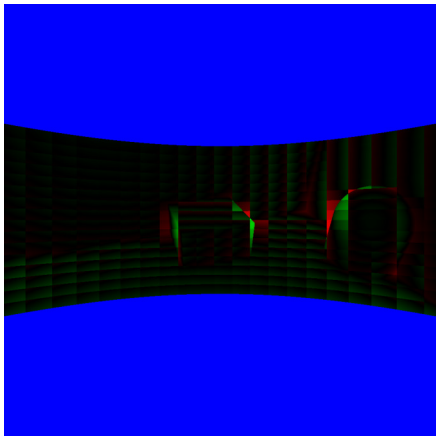


Figure 85: Error map of M5C1

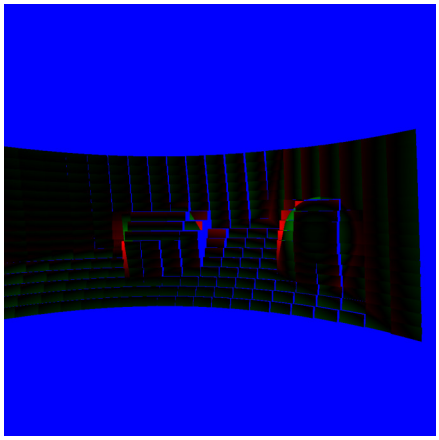


Figure 86: Error map of M5C2

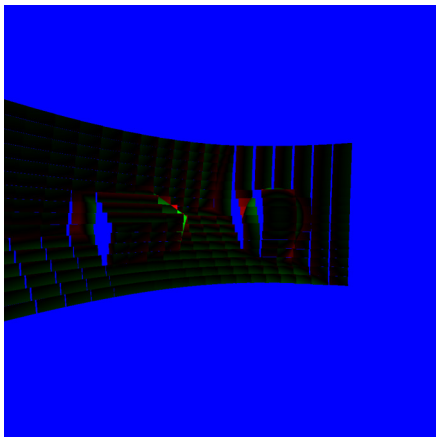


Figure 87: Error map of M5C3

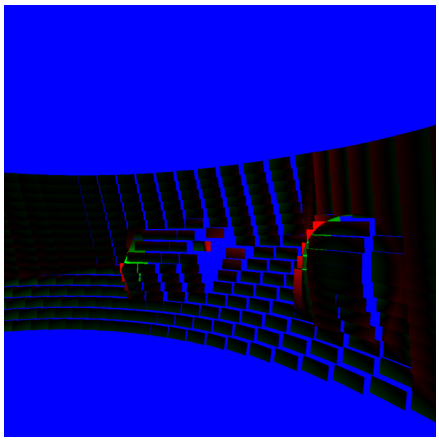


Figure 88: Error map of M5C4

Appendix C: Time Maps

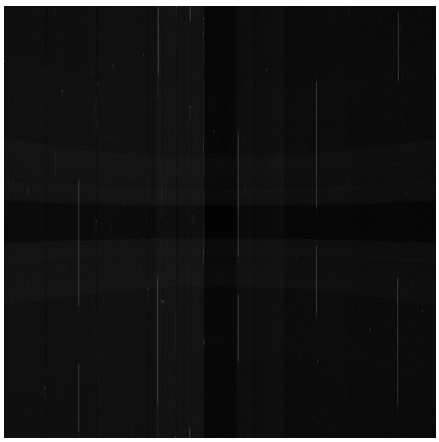


Figure 89: Time map of M0C1

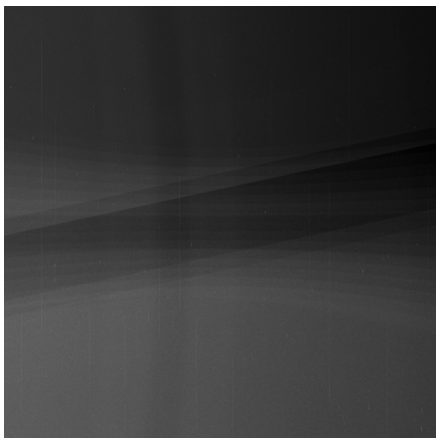


Figure 90: Time map of M0C2

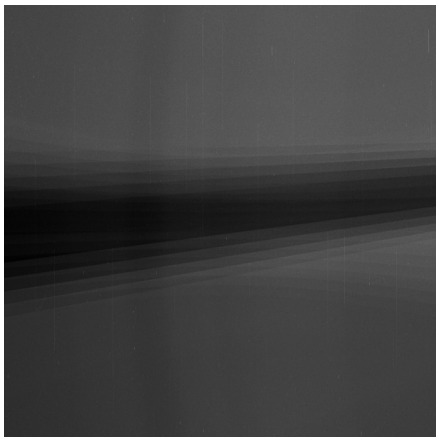


Figure 91: Time map of M0C3

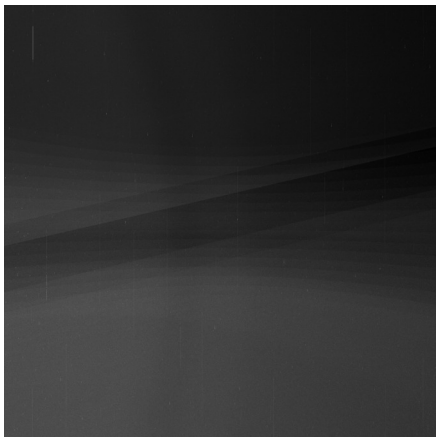


Figure 92: Time map of M0C4

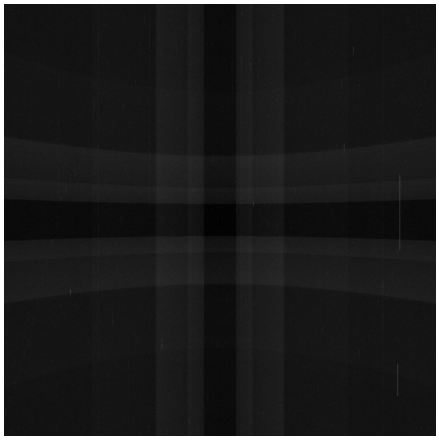


Figure 93: Time map of M1C1

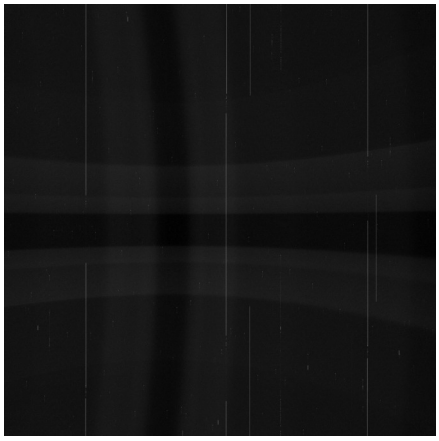


Figure 94: Time map of M1C2

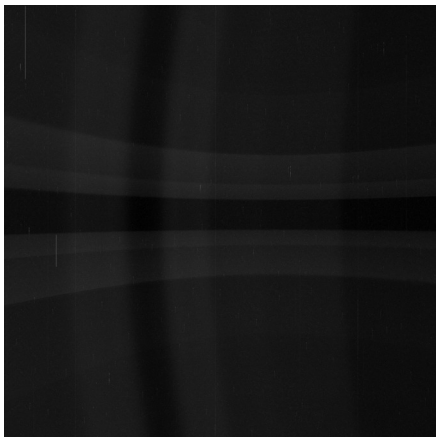


Figure 95: Time map of M1C3

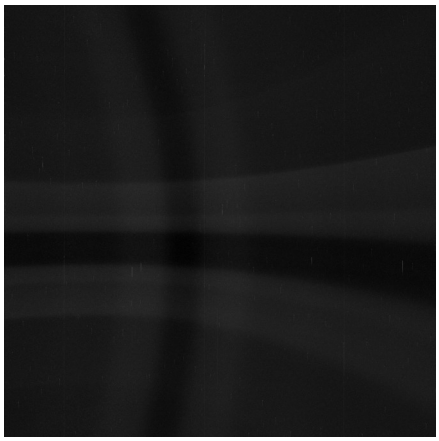


Figure 96: Time map of M1C4

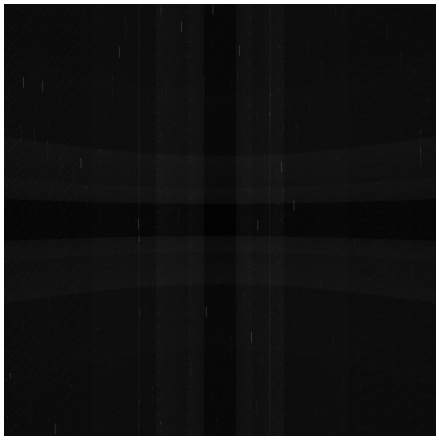


Figure 97: Time map of M2C1

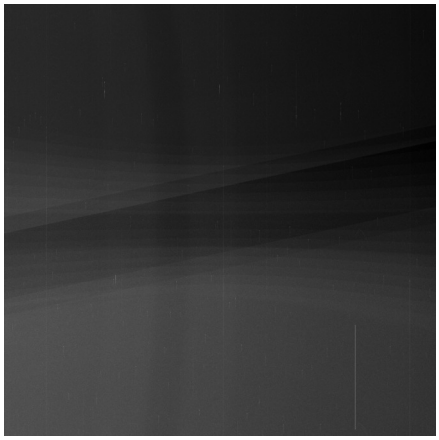


Figure 98: Time map of M2C2

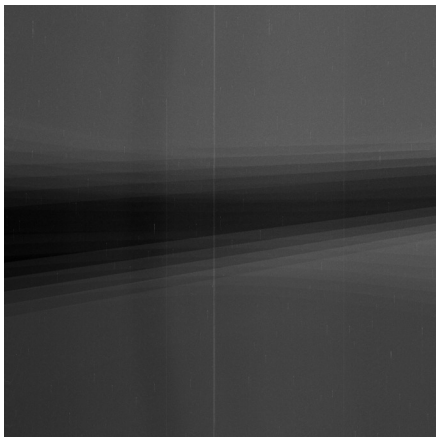


Figure 99: Time map of M2C3

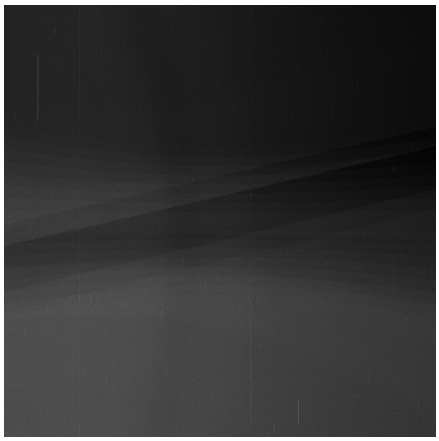


Figure 100: Time map of M2C4

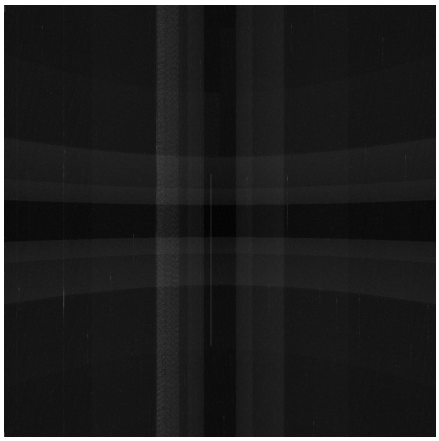


Figure 101: Time map of M3C1

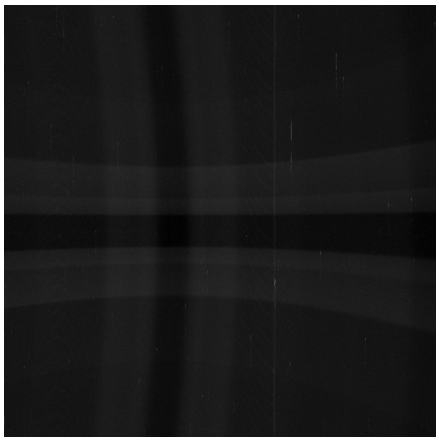


Figure 102: Time map of M3C2

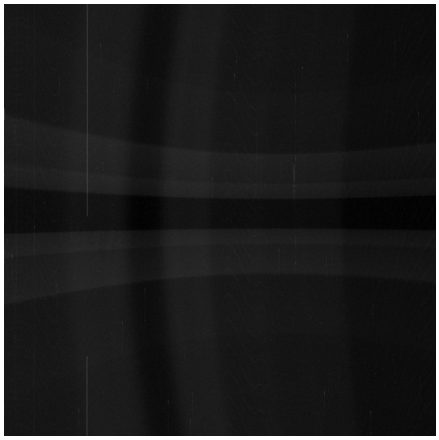


Figure 103: Time map of M3C3

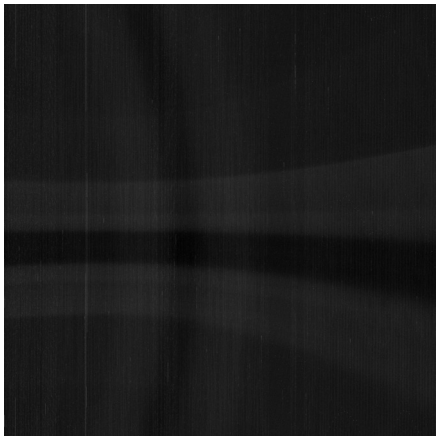


Figure 104: Time map of M3C4

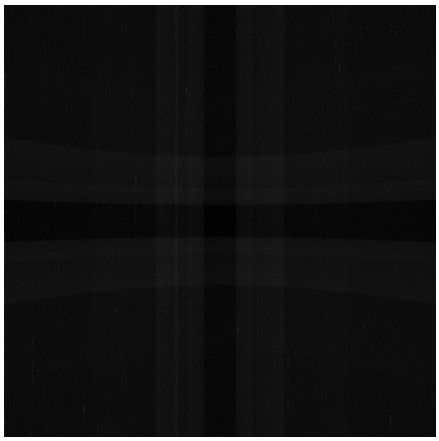


Figure 105: Time map of M4C1

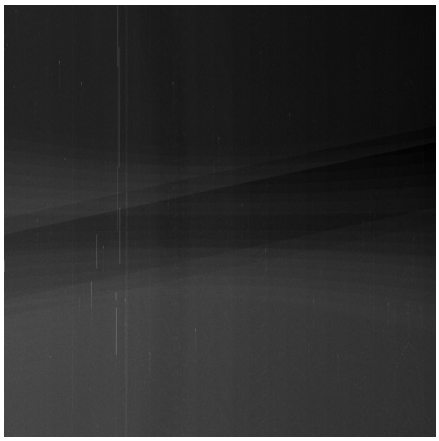


Figure 106: Time map of M4C2

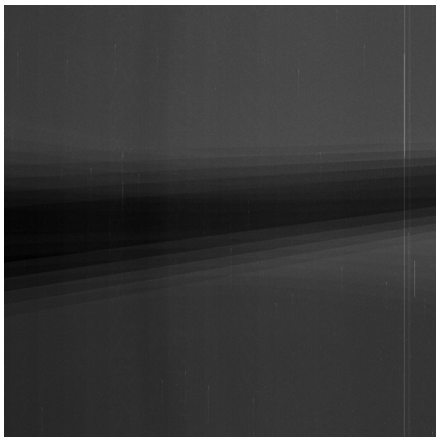


Figure 107: Time map of M4C3

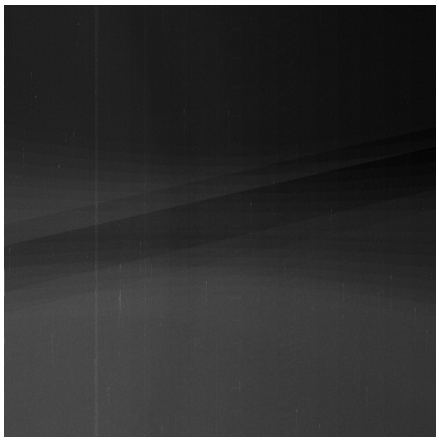


Figure 108: Time map of M4C4

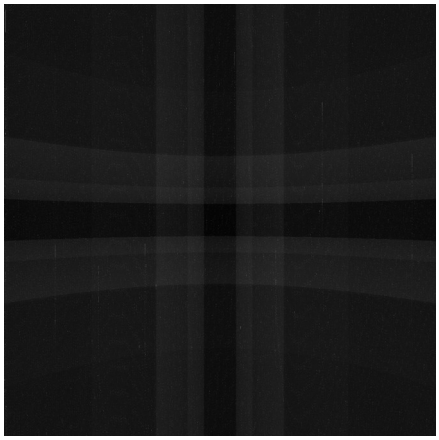


Figure 109: Time map of M5C1

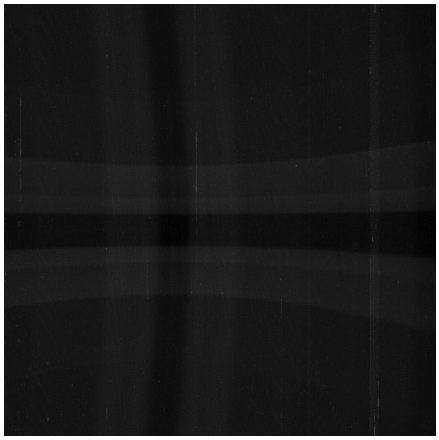


Figure 110: Time map of M5C2

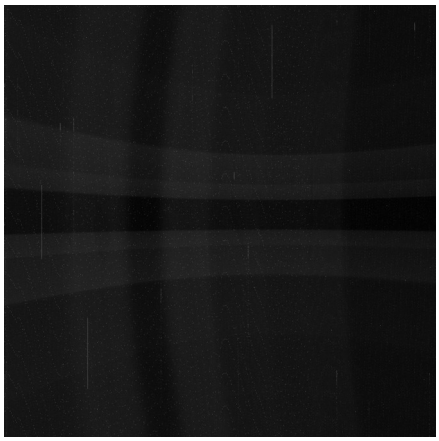


Figure 111: Time map of M5C3

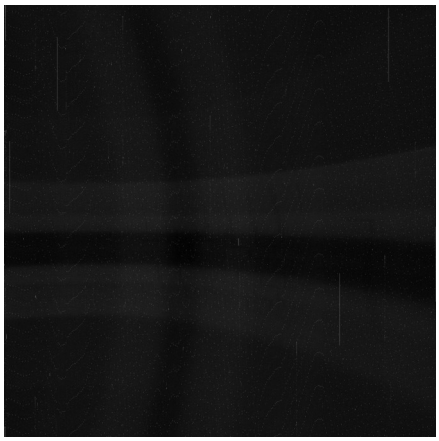


Figure 112: Time map of M5C4

Appendix D: Experiment Setups

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 113: Experiment setup 1

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 114: Experiment setup 2

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 115: Experiment setup 3

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 116: Experiment setup 4

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 117: Experiment setup 5

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 118: Experiment setup 6

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 119: Experiment setup 7

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 120: Experiment setup 8

(a) left camera view **(b)** right camera view **(c)** jitter of laser scanner

Figure 121: Experiment setup 9

Appendix E: Analysis Results

avg time [μs] $\pm$ std [μs] min / max [μs]	M0	M1	M2	M3	M4	M5
C1	4809.863488 $\pm$ 215.245893 4321 / 9488	2699.07055 $\pm$ 101.287751 2359 / 4775	950.692945 $\pm$ 28.319899 871 / 1949	546.869051 $\pm$ 26.173324 468 / 1211	241.867496 $\pm$ 7.443073 222 / 900	138.406904 $\pm$ 6.265475 119 / 284
C2	10368.107226 $\pm$ 1243.917621 7508 / 17435	2727.126483 $\pm$ 135.336375 2356 / 4863	2091.611707 $\pm$ 251.535674 1519 / 3750	551.4578 $\pm$ 22.765891 474 / 1905	531.005911 $\pm$ 63.97694 385 / 1590	140.36722 $\pm$ 6.900079 119 / 1121
C3	10583.420257 $\pm$ 1109.525552 7188 / 17694	2703.991936 $\pm$ 115.342039 2361 / 5225	2133.69899 $\pm$ 221.901335 1450 / 3379	547.584564 $\pm$ 25.194294 475 / 1192	538.738063 $\pm$ 56.301175 363 / 1194	139.249915 $\pm$ 8.021091 119 / 725
C4	10807.108897 $\pm$ 1216.568229 8087 / 18230	2742.259652 $\pm$ 93.306874 2404 / 5038	2179.841847 $\pm$ 247.265164 1626 / 4714	563.056052 $\pm$ 22.015795 485 / 1196	551.798358 $\pm$ 63.30844 410 / 1456	141.301005 $\pm$ 7.384413 123 / 855

Table 3: Average ray tracing calculation times per pixel in each setup (in [μs]). Evaluated in section 4.3.1.

time [μs]	M0	M1	M2	M3	M4	M5
C1	1623	1621	1089	1117	1022	1007
C2	2310	2306	1856	1774	1835	1724
C3	2011	2370	1475	1440	1340	1345
C4	2349	2307	1806	1842	1696	1789

Table 4: Cuboid generation times in each setup (in [μs]).

pos. error [m] neg. error [m] comb. error [m] no depth (ratio)	M0	M1	M2	M3	M4	M5
C1	0.009216 ± 0.026902 -0.002989 ± 0.018567 0.006228 ± 0.033495 62.147903%	0.009216 ± 0.026902 -0.002989 ± 0.018567 0.006228 ± 0.033495 62.147903%	0.009802 ± 0.027609 -0.003609 ± 0.019653 0.006194 ± 0.034713 62.147903%	0.009802 ± 0.027609 -0.003609 ± 0.019653 0.006194 ± 0.034713 62.147903%	0.012969 ± 0.037935 -0.00704 ± 0.032984 0.00593 ± 0.052279 62.147903%	0.012969 ± 0.037935 -0.00704 ± 0.032984 0.00593 ± 0.052279 62.147903%
C2	0.007557 ± 0.022374 -0.003913 ± 0.029929 0.003644 ± 0.038167 64.343166%	0.007307 ± 0.022519 -0.002967 ± 0.020837 0.00434 ± 0.031315 65.458202%	0.00794 ± 0.022679 -0.004688 ± 0.031972 0.003252 ± 0.040214 63.957691%	0.007493 ± 0.022466 -0.003865 ± 0.025471 0.003628 ± 0.034816 65.586472%	0.01148 ± 0.030547 -0.0081 ± 0.043326 0.003381 ± 0.054518 62.785816%	0.010481 ± 0.029261 -0.007437 ± 0.039979 0.003044 ± 0.051227 64.590836%
C3	0.006889 ± 0.024314 -0.002857 ± 0.022305 0.004033 ± 0.033478 70.749092%	0.006733 ± 0.023826 -0.002425 ± 0.01936 0.004308 ± 0.031081 72.086716%	0.007361 ± 0.026669 -0.003261 ± 0.021797 0.004101 ± 0.035071 70.612621%	0.007041 ± 0.026256 -0.00275 ± 0.018909 0.004292 ± 0.032819 72.287655%	0.009664 ± 0.03163 -0.005709 ± 0.029262 0.003955 ± 0.044008 70.280361%	0.008944 ± 0.029681 -0.004932 ± 0.024886 0.004013 ± 0.040013 71.869469%
C4	0.00983 ± 0.025998 -0.007047 ± 0.047492 0.002783 ± 0.055506 54.366112%	0.009131 ± 0.037367 -0.004061 ± 0.030394 0.005071 ± 0.048815 59.843063%	0.01002 ± 0.026471 -0.008597 ± 0.052895 0.001423 ± 0.06074 54.362392%	0.008983 ± 0.035351 -0.005879 ± 0.040884 0.003105 ± 0.055009 60.491085%	0.01216 ± 0.02936 -0.013801 ± 0.059924 -0.001642 ± 0.069103 54.312801%	0.010743 ± 0.034772 -0.011258 ± 0.051632 -0.000516 ± 0.063856 60.190868%

Table 5: Average error values of each setup (in [meter], %). Evaluated in section 4.3.1.

Appendix F: Experiment Setups Statistics

Setup	Tag	Camera	Distance Model	Standard Deviation	Delta from LRF	Distance LRF
S1	T1	left	7.6270397059	± 0.0054733611	-0.0109602941	7.638
		right	7.6560584603	± 0.003729596	0.0180584603	
	T2	left	17.6057011176	± 0.0106332354	-0.0612988824	17.667
		right	17.6022800124	± 0.0077256081	-0.0647199876	
S2	T1	left	9.7973379048	± 0.0060097856	-0.0606620952	9.858
		right	9.7765284545	± 0.0064733504	-0.0814715455	
	T2	left	16.4030116	± 0.012691796	-0.0799884	16.483
		right	16.4047931371	± 0.009669433	-0.0782068629	
S3	T1	left	12.0728508421	± 0.0063452082	-0.0581491579	12.131
		right	12.0537248302	± 0.0062733896	-0.0772751698	
	T2	left	14.0669131579	± 0.006939209	0.0289131579	14.038
		right	14.0704134349	± 0.0082744343	0.0324134349	
S4	T1	left	14.040965059	± 0.0121887296	-0.004034941	14.045
		right	14.0365550709	± 0.0147800532	-0.0084449291	
	T2	left	14.0745234615	± 0.006085638	-0.0134765385	14.088
		right	14.0768870955	± 0.0045263142	-0.0111129045	
S5	T1	left	7.8727330588	± 0.0076547932	-0.0682669412	7.941
		right	7.8660132929	± 0.007824201	-0.0749867071	
	T2	left	16.4588517059	± 0.0047833597	-0.0751482941	16.534
		right	16.4571858824	± 0.0078419391	-0.0768141176	
S6	T1	left	11.6216157647	± 0.0087215117	-0.0653842353	11.687
		right	11.5937364598	± 0.0078803744	-0.0932635402	
	T2	left	11.8053281765	± 0.0064042569	-0.0456718235	11.851
		right	11.8436005	± 0.0070669893	-0.0073995	
S7	T1	left	NaN	± NaN	NaN	11.69
		right	NaN	± NaN	NaN	
	T2	left	NaN	± NaN	NaN	12.151
		right	NaN	± NaN	NaN	
S8	T1	left	8.9841229	± 0.0056416783	NaN	NaN
		right	8.9956141607	± 0.00730289	NaN	
	T2	left	22.7103699412	± 0.0093788885	NaN	NaN
		right	22.6803233889	± 0.0128153515	NaN	
S9	T1	left	9.4711898333	± 0.0067245498	NaN	NaN
		right	9.4842050385	± 0.0066199314	NaN	
	T2	left	33.2919965833	± 0.0110453421	NaN	NaN
		right	33.2755210968	± 0.0058390863	NaN	

Table 6: Experiment setups statistics. Evaluated in section 5. Data in [meter].

Declaration

I declare that I have written this Bachelor thesis inclusive attached material independently and using only the specified literature. Every literally or analogous content from other work is marked in every single case. Furthermore, I declare that I have not previously submitted the same work to any other examination board for obtaining an academic degree.

I am aware, that the violation of this declaration and conscious deception will result in a grading of this thesis with the mark 5.0.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit einschließlich aller beigefügter Materialien selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Werken entnommen sind, sind in jedem Einzelfall unter Angabe der Quelle deutlich als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.

Mir ist bekannt, dass Zuwiderhandlungen gegen diese Erklärung und bewusste Täuschungen die Benotung der Arbeit mit der Note 5.0 zur Folge haben kann.

Place, Date

(Tobias Neumann)