Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Niklas Bakula

Concept development, evaluation and implementation of a method for image-based distance computation in AUDEx

Fakultät Technik und Informatik
Department Fahrzeugtechnik und Flugzeugbau

Faculty of Engineering and Computer Science
Department of Automotive and Aeronautical Engineering

Niklas Bakula

Concept development, evaluation and implementation of a method for image-based distance computation in AUDEx

Bachelorbeit eingereicht im Rahmen der Bachelorprüfung

im Sommersemester 2022
am Department Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

First examiner: Prof. Dr. Dirk Engel

Second examiner: Prof. Dr. Sven Füser

Abgabedatum 19.09.2022

Niklas Bakula

Title of the paper

Concept development, evaluation and implementation of a method for image-based distance determination in the AUDEx

Keywords

AUDEx, computer vision, python, calibration, rectification, correspondence, triangulation, object detection, OpenCV, deep learning, transfer learning

Abstract

This paper studies a machine learning based vision aid concept for the implementation on a teleoperated Taxibot. The mathematical background of computer vision as well as the key steps needed to configure a stereo camera system are discussed and a software prototype, capable of object classification and depth detection is implemented and tested.

Niklas Bakula

Thema der Bachelorarbeit

Konzeptentwicklung, Bewertung und Implementierung einer Methode zur bildbasierten Abstandsermittlung im AUDEx-Projekt

Stichworte

AUDEx, computer vision, python, calibration, rectification, correspondence, triangulation, object detection, OpenCV, deep learning, transfer learning

Kurzzusammenfassung

Diese Bachelorarbeit untersucht ein bildbasiertes Konzept zur Abstandsermittlung für die Implementierung auf einem teleoperierten Taxibot. Der mathematische Hintergrund des computerbasierten Sehens sowie die wichtigsten Schritte, die zur Konfiguration eines Stereo-Kamerasystems erforderlich sind, werden erörtert und ein Softwareprototyp, der zur Objektklassifizierung und Tiefenerkennung fähig ist, wird implementiert und getestet.

Content

CONTENT ..... **IV**

TABLE OF FIGURES ..... **V**

LIST OF TABLES ..... **VIII**

LIST OF SYMBOLS ..... **IX**

LIST OF ABBREVIATIONS ..... **XI**

1 INTRODUCTION **1**

2 FUNDAMENTALS **2**

2.1 DEPTH DETECTION TECHNOLOGY 2
2.2 THE MATHEMATICS OF A CAMERA 5
2.2.1 Camera intrinsics and coordinate systems 5
2.2.2 Distortion 8
2.2.3 Coordinate transformation 9
2.2.4 Planar homography 11
2.2.5 Field of view 12
2.2.6 Depth of field 14
2.3 THE MATHEMATICS OF STEREO CAMERAS 16
2.3.1 Stereo camera geometry 16
2.3.2 Stereo calibration 17
2.3.3 Stereo rectification 19
2.3.4 Depth computation 20
2.3.5 Conclusion 22
2.4 OBJECT DETECTION 23

3 REQUIREMENTS **25**

4 IMPLEMENTATION **27**

4.1 HARDWARE 27
4.1.1 Geometry analysis 27
4.1.2 Prototype 32
4.2 SOFTWARE 33
4.2.1 Installation 33
4.2.2 Model training 36
4.2.3 Depth computation 40

5 TESTING **58**

6 CONCLUSION AND OUTLOOK **61**

7 SOURCES **63**

8 APPENDIX **70**

Table of figures

Figure 1: Principles of light reflection [7] 2

Figure 2: Point cloud generated using LiDAR [10] 3

Figure 3: Components of a spinning LiDAR [11] 3

Figure 4: Components of a colour image sensor [14] 4

Figure 5: Pinhole camera model, based on [15] 5

Figure 6: Pinhole model used in computer vision, based on [16] 6

Figure 7: Example of radial distortion, [15] 9

Figure 8: Coordinate transformation (rotation), based on [15] 9

Figure 9: Projective transformation, based on [15] 11

Figure 10: Geometry of the field of view (FOV), based on [19] 13

Figure 11: Depth of field geometry, based on [21] 14

Figure 12: Simple stereo system, based on [15] 16

Figure 13: Epipolar geometry, based on [22] 18

Figure 14: Example of an image before rectification [23] 19

Figure 15: Example of the same image after rectification [23] 19

Figure 16: Visual representation of disparity, based on [23] 21

Figure 17: Fundamental architecture of a neural network [25] 23

Figure 18: Geometry of Taxibot and aircraft, units in [m] 27

Figure 19: Relation DOF to plane of focus for f=6mm, N=1,8, c=2,74um 30

Figure 20: Simple stereo hardware prototype 32

Figure 21: Display of activated virtual environment 33

Figure 22: Command to perform initial installation in an activated virtual environment 34

Figure 23: Display of successful installation 36

Figure 24: Selection of project folder im jupyter notebook 36

Figure 25: : Selection of training script in jupyter notebook 36

Figure 26: Selection of correct kernel in jupyter notebook 37

Figure 27: Path to model training environment 39

Figure 28: Main camera setup menu 40

Figure 29: Single camera setup menu 41

Figure 30: Display of captures images from the back 42

FIGURE 31: DISPLAY OF CAPTURES IMAGES FROM THE SIDE 42
FIGURE 32: STORAGE LOCATION FOR SINGLE CAM CALIBRATION IMAGES 43
FIGURE 33: COMPARISON OF THE REPROJECTION ERROR FOR VARIOUS SIZES OF IMAGE COMBINATIONS 45
FIGURE 34: COMPARISON OF REPROJECTION ERRORS IN MATLAB AND OPENCV 46
FIGURE 35: ORIGINAL IMAGE FRAME 47
FIGURE 36: UNDISTORTED IMAGE FRAME 47
FIGURE 37: DISPLAY OF OPTIONS FOR THE STEREO CAMERA SETUP 48
FIGURE 38: DIRECTORY OF GENERATED STEREO IMAGES 48
FIGURE 39: STEREO OVERALL MEAN REPROJECTION ERROR 49
FIGURE 40: HORIZONTAL OFFSET FOR STEREO SETUP 50
FIGURE 41: DETAIL VIEW OF UN-RECTIFIED FRAME 52
FIGURE 42: DETAILED VIEW OF RECTIFIED FRAME 52
FIGURE 43: DETECTABLE DEPTH RANGE - "HOROPTER" [50] 53
FIGURE 44: EXAMPLE OF DISPARITY MAPS USING DIFFERENT BLOCK SIZES 55
FIGURE 45: SAMPLE OF THE ORIGINAL IMAGE PAIR USED FOR STATIC TESTS 58
FIGURE 46: SAMPLE OF THE LEFT CAMERA FRAME, INCLUDING DETECTION 59
FIGURE 47: SAMPLE OF THE RIGHT CAMERA FRAME, INCLUDING DETECTION 59
FIGURE 48: STEREOBM, BLOCKSIZE = 5PX, GROUNDTRUTH=35CM, MEANDISTANCE=55CM 59
FIGURE 49: STEREOBM, BLOCKSIZE=21PX, GROUNDTRUTH=35CM, MEANDISTANCE =55CM 59
FIGURE 50: STEREOBM, BLOCKSIZE=5PX, GROUNDTRUTH=45CM, MEANDISTANCE=54,52CM 60
FIGURE 51: STEREOSGBM DISPARITY MAP, BLOCKSIZE=5PX, GROUNDTRUTH=45CM, MEANDISTANCE=50,90CM 60
FIGURE 52: ACCURACY COMPARISON FOR STEREO ALGORITHMS 60
FIGURE 53: COMPUTATION TIME COMPARISON FOR STEREO ALGORITHMS 60
FIGURE 54: GEOMETRIC REQUIREMENTS DEFINED IN [6] 74
FIGURE 55: FRONTAL FOVS TAXIBOT 75
FIGURE 56: FORWARD AND BACKWARD DOF 75
FIGURE 57: TOP VIEW OF HARDWARE PROTOTYPE 76
FIGURE 58: BOTTOM VIEW OF HARDWARE PROTOTYPE 76
FIGURE 59: STRUCTURE OF DIRECTORY 78
FIGURE 60: RECTIFICATION CHECK, ORIGINAL FRAMES 79
FIGURE 61: RECTIFICATION CHECK, RECTIFIED FRAMES 79
FIGURE 62: FIRST SAMPLE OF MODIFIED IMAGES USING ALBUMENTATION 80

FIGURE 63: SECOND SAMPLE OF MODIFIED IMAGES USING ALBUMENTATION 80
FIGURE 64: THIRD SAMPLE OF MODIFIED IMAGES USING ALBUMENTATION 80
FIGURE 65: FOURTH SAMPLE OF MODIFIED IMAGES USING ALBUMENTATION 80

List of tables

TABLE 1: TECHNICAL SPECIFICATION FOR BASLER ACE 2 [28] 28

TABLE 2: TECHNICAL SPECIFICATION BASLER KOWA LENS LM6HC [29] 29

TABLE 3: SUMMARY OF FURTHER REQUIRED CAMERA CONFIGURATIONS 31

TABLE 4: LIST OF REQUIRED PACKAGES 35

TABLE 5: PARAMETERS MODIFIED USING ALBUMENTATION 38

TABLE 7: ESTIMATION OF HARDWARE COSTS 77

TABLE 8: HARDWARE USED FOR STEREO VISION 77

List of symbols

Symbol	Description	Unit
D	Aperture	mm
b	Baseline	$\mathrm{mm, m, cm\ etc.}$
b	Blur circle	pixels
$\mathrm{Q_c = [X_c\ Y_c\ Z_c]^T}$	Camera coordinates	$\mathrm{mm, m, cm\ etc.}$
$\mathrm{O_{L,R}}$	Centre of projection	-
Z	Depth	$\mathrm{mm, m, cm\ etc.}$
d	Disparity	pixels
$\mathrm{c_x, c_y}$	Displacement of optical axis from top left image plane corner	pixels
R	Distance to object	$\mathrm{mm, m, cm\ etc.}$
i	Distance from image plane to lens	mm
$\mathrm{e_{L,R}}$	Epipole	-
E	Essential matrix	-
N	f-number	-
$\frac{\Theta_{\mathrm{V,H}}}{2}$	Field of view angle	$°, rad\ (radians)$
F	Focal length	mm
$\mathrm{f_x, f_y}$	Focal length	pixels
F	Fundamental matrix	-
H	Homography matrix	-
$\mathrm{q = [u\ \ v]^T}$	Image coordinates	pixels
$\mathrm{s_x, s_y}$	Pixel density	$\frac{\mathrm{pixels}}{\mathrm{mm}}$

C	Pixel size	pixels
$\mathrm{k_1, k_2, k_3}$	Radial distortion coefficients	-
Q	Reprojection matrix	-
$\mathrm{R, r, R_{rect}}$	Rotation matrices	$rad\ (radians)$
H	Sensor height	$mm, pixels$
W	Sensor width	$mm, pixels$
$\mathrm{p_1, p_2}$	Tangential distortion coefficients	-
t	Translation vector	$mm, m, cm\ etc.$
$\mathrm{Q_w = [X_w\ Y_w\ Z_w]^T}$	World coordinates	$mm, m, cm\ etc.$

List of abbreviations

Abbreviation	Description
API	Application programming interface
AUDEx	Automotive Development in 1:x
BGR	Blue, Green, Red
Bm	Block matching
COCO	Common Object in Context
CSCT	Centre symmetric census transform
DOF	Depth of field
EASA	European Union Aviation Safety Agency
FoV	Field of view
LiDAR	Light detection and ranging systems
RGB	Red, Green, Blue
SAD	Sum of absolute differences
SGBM	Semi global block matching
SSD	Single-Shot-Detector
TF	Tensorflow
ToF	Time of Flight
WLS	Weighted Least Squares

1 Introduction

The introduction of computers has allowed many human tasks to be automated. From fast and accurate executions of mathematical operations using calculators to self-driving vehicles, the high availability of today's computational resources has dramatically increased the usage of new technologies in our lives.

Automotive companies, such as Audi, Mercedes or Tesla, have frequently been the subject of headlines for testing new technologies that take the use of computers in cars to the next level [1]. These technologies, many of which use machine learning, have the aim of increasing the safety, comfort and efficiency of our travels. Especially Tesla has been very vocal about the benefits of computer vision for vehicles [2]. But the application of such technologies has been tested for the use at airports as well. From driverless trains transporting passengers between terminals [3] to autonomous baggage tractors [4], companies are interested in tech that increases safety, reduces (operational) costs and lowers emissions.

The "Automotive Development in 1:x" (AUDEx) project at the University of Applied Sciences in Hamburg enables students to study various concepts of teleoperation using scaled vehicle models with different kinds of sensors and actors. In particular, a simulator was built, which allows a driver to teleoperate a vehicle using virtual reality glasses that provide a view of the environment around the vehicle as well as an augmented display of the vehicle's cockpit to ensure a better sense of orientation. Furthermore, visual and haptic feedback are given to the driver [5].

Previous work [6] has studied the adaptation of such a teleoperation concept for the use in tow tractors at airports while focusing on the visual requirements to ensure that an operator in a remote location is provided with views at all critical points at or around the aircraft. This paper provides an in-depth analysis of a computer vision based visual aid system for one of the concepts described in the previous work. Furthermore, the aim of this work is to provide a hardware and software prototype for such a system.

2 Fundamentals

2.1 Depth detection technology

Computer vision applications often rely on the use of depth detection technology. Especially the field of autonomous driving has seen a considerable increase in the usage of this technology. It allows systems to detect distances to obstacles and, as a result, increases the safety of a drivers, passengers or bystanders. Some of the most known depth detection technologies shall be introduced in the following paragraphs.

Light

The human eye can perceive its environment when there is light. A light source, such as the sun, emits electromagnetic waves with certain wavelengths or, using an idealisation, light rays, which travel down to earth. When light rays hit an object in space they can be absorbed, transmitted and/or reflected [8]. Depending on whether the surface is smooth (*regular reflection* on a polished metal) or textured (*diffused* or *scattered reflection*) the amount of light that is reflected changes. Furthermore, coloured objects absorb and reflect different wave lengths of the light, depending on the colour their surface is made of. A red object, for instance, absorbs much of the blue and green light and a red light ray is reflected [7]. If the light ray is reflected into the human eye, it is perceived in this colour.

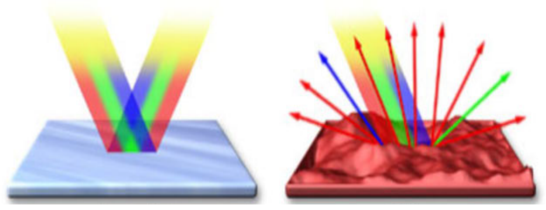


Figure 1: Principles of light reflection [7]

Lidar

Light detection and ranging systems (LiDAR) belong to the category of Time of Flight (ToF) systems [9]. While there are various implementations of LiDAR technology, the fundamental concept of how distances are measured is based on a laser that emits a light ray. The light ray travels in the direction that it is emitted in, is reflected by an object and returns to the LiDAR installation, where it is recorded by a scanner. The distance to the object can be calculated by measuring the time it took for the light ray to travel from the laser to the scanner and by using the known speed of light for the particular wavelength [10]. *Mechanical spinning systems* use one or multiple lasers and scanners, which measure the distance to multiple points in the environment at once [11]. By rotating the LiDAR system, a field of view (FoV) of 360° can be achieved. Mirrors can be used to reflect the emitted light source from the laser vertically in order to generate a vertical FoV. The result is a point cloud, which is a large number of points with known 3D

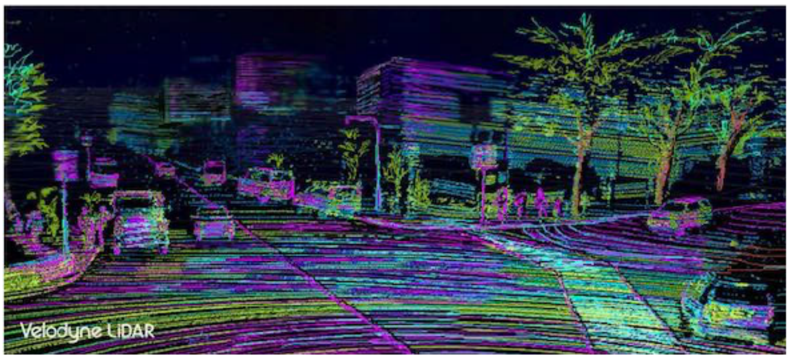


Figure 2: Point cloud generated using LiDAR [10]

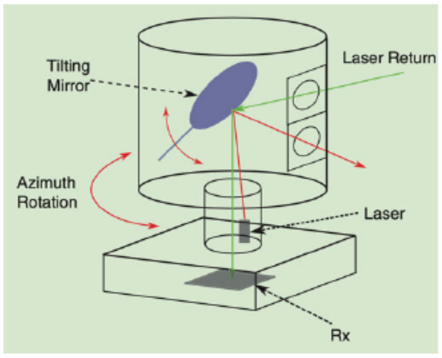


Figure 3: Components of a spinning LiDAR [11]

coordinates, as can be seen in Figure 2. LiDAR systems offer precise distance measurements for objects in a distance of up to 300m. They are capable of recording point clouds of their surrounding at a frequency of 10 frames per second. Furthermore, distances can be measured accurately independent of visibility conditions because they are active sensors that emit light. Other sensors, as we will find out later, depend on good lighting conditions because they record reflected light from objects and do not have an own light source. The primary downside of mechanical spinning systems is their price tag. While price information is not given on manufacturer websites, studies have shown single sensors to cost more than 70.000USD [10].

Recently, a new LiDAR System was announced, which is called *solid state lidar.* Similar to the mechanical spinning system, a solid laser emits a light beam. Different from the previous systems, the laser beam is diffused in a manner that allows a single light source to illuminate a certain area of the environment, instead of a single point. The light is then reflected and absorbed by an array of scanners with a similar shape as the illuminated area. A point cloud of the environment is created. In comparison to the mechanical spinning systems solid state systems offer a cheaper price point with some costing around 500USD [12]. However, a single solid-state system can only be used for distance detection in one specific direction, leading to a FoV of 90° for some systems, while the spinning system provides a 360° FoV. Furthermore, the solid systems can measure distances to objects located up to 100m from the system [13]. Spinning systems provide greater distance measurement capability.

Cameras

Depth computation can also be performed using cameras, in particular two cameras in a type of vision called *binocular vision*. While the following chapters will discuss how depth detection using such as stereo camera setup works, the components of cameras shall be explained in this section.

The manner in which a camera can capture an image of its environment is similar to how the human eye perceives light. A light ray that is reflected on an object enters the camera through a lens. The lens, through its physical shape, transmits and focusses the ray on an image sensor. The image sensor consists of many pixel elements that are essentially photodiodes equipped with a microlens as well as a colour filter. Each pixel element includes some electric circuits on its edges, which cannot capture light. The light from the main lens is focussed on the active pixel area, which can capture light, be a microlens. A colour filter below the microlens filters the respective colour, with neighbouring pixel elements having one red, green and blue (RGB) filter [14].

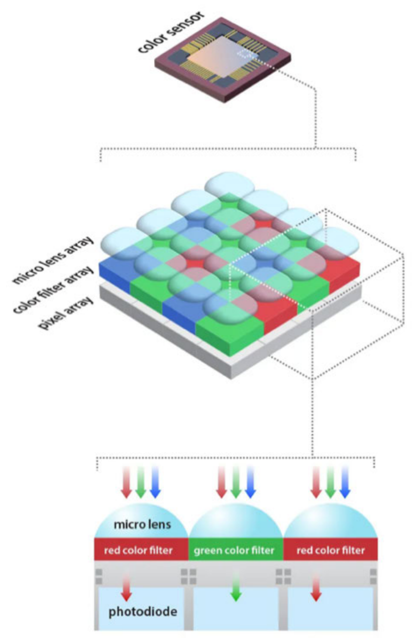


Figure 4: Components of a colour image sensor [14]

The three pixel elements provide their respective colour information for one point on the image sensor, from which the actual RGB colour of a point is computed. After it passed through the colour filter, a light ray enters the photodiode, where the electromagnetic light wave's energy is converted into an electric signal that can be further processed by the electric circuits.

The quality of the image a camera produces greatly depends on the quality of the (main) lens, as well as the sensor quality. A measure of the quality of an image sensor is its resolution, i.e., how many pixel elements it has per millimetre. Sensors of 4K cameras have 3840 horizontal and 2160 vertical pixel elements with each pixel element having a length of down to 1,67 micrometres. Some image sensors have rectangular pixel elements, instead of square ones.

The function of a lens is to capture a large number of light rays and focus these rays on the image sensor. The wider the lens, the more light can be captured and, thus, the more scenery can be captured on a single image. Cameras with such a large field of view are called *wide angle cameras.*

2.2 The mathematics of a camera

The following sections provide a high-level introduction to the physics and mathematics of (stereo) cameras that are required to understand key requirements for and operation of computer vision components.

2.2.1 Camera intrinsics and coordinate systems

The *pinhole camera model* (see Figure 5) is an idealised camera model that provides a basic understanding of what are called the *camera intrinsics*. A light ray from an object in space $Q = [X\ \ Y\ \ Z]^T$ enters the camera through a pinhole and is projected onto the *image plane* (also called *projective plane)*.

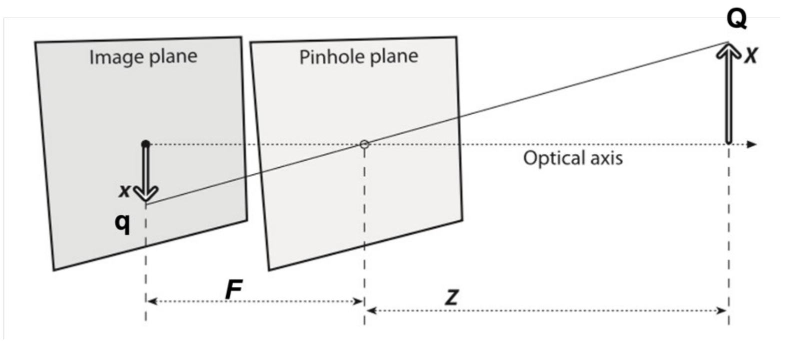


Figure 5: Pinhole camera model, based on [15]

The *optical axis* is defined as a straight line passing through the pinhole while perpendicular to the image plane. The *centre of projection* lies in the centre of the pinhole. The point where the optical axis intersects with the image plane is called *principal point*. Using similar triangles, the size of the projected object Q on the image plane can be calculated as:

$$\frac{-x}{F} = \frac{X}{Z} \rightarrow -x = F * \frac{X}{Z} \quad and \quad \frac{-y}{F} = \frac{Y}{Z} \rightarrow -y = F * \frac{Y}{Z} \tag{1}$$

The *focal length (F)* is defined as the distance between the centre of projection and the image plane. Equation (1) makes it evident that the focal length is the parameter of a camera that determines in which size an object in the real world is projected onto the image plane. The focal length of a lens is specified in its technical documents and given in world units (i.e., usually mm). A larger focal length creates a larger projection on the image plane. Both coordinates on the image plane being negative shows that the projected object is upside down as well as horizontally flipped compared to the real world. The above-mentioned relationship is usually illustrated in a simpler manner, in which the image plane is mirrored along the optical axis from the centre of projection (Figure 6). As a result, the image plane as well as the object Q are both at a positive distance from the centre of projection and thus the negative values for coordinates x and y in equation (1) become positive [15].

In computer vision there are three coordinate systems to consider. The first is the world-coordinate system, in which real world units (i.e., cm, m, inches etc.) are used. A point $Q_w = [X_w \ Y_w \ Z_w]^T$ in the world-coordinate system can be expressed in camera coordinates $Q_c = [X_c \ Y_c \ Z_c]^T$, which use the same units as the world coordinate system. While the world-coordinate system can be located anywhere in space, the camera coordinate system originates in a camera's centre of projection.

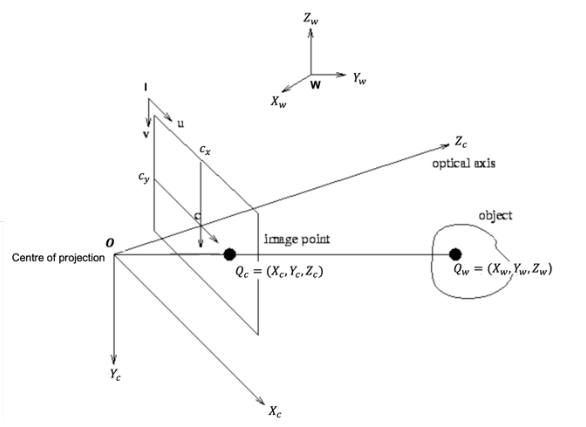


Figure 6: Pinhole model used in computer vision, based on [16]

Its Z_c-axis is parallel to the optical axis, the Y_c-axis points downwards and the X_c-axis is perpendicular to both. When a point in camera coordinates Q_c is projected onto the image plane, it's position can be expressed in *image coordinates* $q = [u \ v]^T$, which specify its location in pixel coordinates. The image coordinate system originates in the upper left-hand corner of the image plane. As it was mentioned in the last subsection, pixel elements can be of rectangular or quadratic shape. The position in image coordinates is determined by multiplying the horizontal and vertical position in camera coordinates with the pixel density of the image sensor in the respective direction:

$$u = s_x * X_c \quad \text{and} \quad \text{v}= s_y * Y_c \qquad (2)$$

In equation (2) the pixel densities (s_x, s_y) have units in $\frac{pixels}{mm}$, image coordinates (u, v) are in units of pixels and camera coordinates (X_c, Y_c, Z_c) are given in mm. Since coordinates in the camera coordinate system are measured from the optical axis and the image coordinate system does not originate in the same point, the location of the optical axis with respect to the origin of the image coordinate system is considered using the parameters c_x and c_y. Furthermore, in reality the optical axis is not perfectly aligned with the centre of the image plane. This is due to the fact that the lens (the pinhole in this idealised model) is not aligned precisely due to manufacturing inaccuracies. As a result, the displacement of the image plane's centre from the optical axis is considered in the parameters c_x and c_y. Substituting equations (2) into equation (1)[1] and defining two focal lengths, f_x and f_y (in pixels) as the product of the focal length (F) and the respective sensor density $(f_x = s_x * F, and\ f_y = s_y * F)$ yields:

$$u = f_x * \left(\frac{X_c}{Z_c}\right) + c_x \qquad (3)$$

And

$$\text{v}= f_y * \left(\frac{Y_c}{Z_c}\right) + c_y \qquad (4)$$

These four parameters, the two displacements c_x, c_y and the two focal lengths f_x ,f_y, are called the *camera intrinsics* and are noted in the *camera matrix (M):*

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

[1] Remember that in this idealised model the equation has a positive sign

To provide a linear system of equations, matrices are noted in homogenous coordinates. Equations (3) and (4) can be written as:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{5}$$

The actual screen coordinates in equation (5) are determined by $u = \frac{su}{s}$ with $s = Z_c$. A point in camera coordinates Q_c can thus be transformed to image coordinates with:

$$q = MQ_c \ , where\ q = s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{6}$$

The reverse transformation can be achieved using the inverse of the camera matrix:

$$Q_c = M^{-1}q \tag{7}$$

It shall be noted that for this paper the intrinsic parameter skew (s) is not considered, in contrast to other work mentioned in this paper. This is due to the fact that modern cameras have a very small skew, which can sufficiently be neglected [15].

2.2.2 Distortion

A camera in the real world, in comparison to the pinhole camera, allows multiple rays to enter from various directions by using a *lens*. Depending on the shape of the lens, the light rays are focussed onto the image plane. The physical characteristics of the lens introduce another import parameter that has to be considered when working with computer vision: *Distortion*.

Radial distortion is caused by the shape of a lens, which instead of ideally being parabolic, in reality has a spherical shape. This causes a light ray travelling through the lens to be diverted, different to how it would ideally be, when entering at the edges of the lens. Light rays in the centre are not diverted (distortion at the centre is zero).

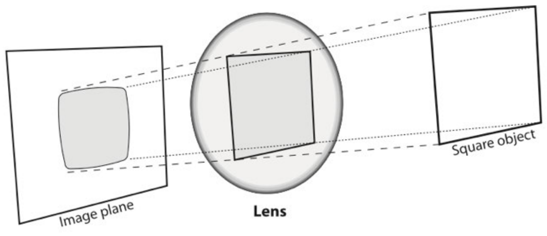


Figure 7: Example of radial distortion, [15]

Tangential distortion arises from the assembly process of the camera when the image plane is not placed exactly parallel to the lens.

Another characteristic to consider is that wide angle lens produce a higher distortion than regular lenses [17].

The effect of distortion can be computationally corrected as long as the distortion coefficients, k_1 through k_3 for radial distortion and p_1 and p_2 for tangential distortion, are known.

These four intrinsic parameters $(f_x, f_y, c_x \ and \ c_y)$ as well as the distortion coefficients $(k_1, k_2, k_3, p_1, p_2)$ of a camera are determined during the camera calibration process, which will be discussed in detail.

2.2.3 Coordinate transformation

The position and pose of the world coordinate system can be offset from the camera coordinate system by translation and/or rotation about any or all of the three axis. A coordinate transformation determines the coordinates of a point $Q_w = [X_w \ Y_w \ Z_w]^T$ in the camera coordinate system $Q_C = [X_C \ Y_C \ Z_C]^\mathrm{T}$, as shown for a rotation about one axis in Figure 8. The new location of one

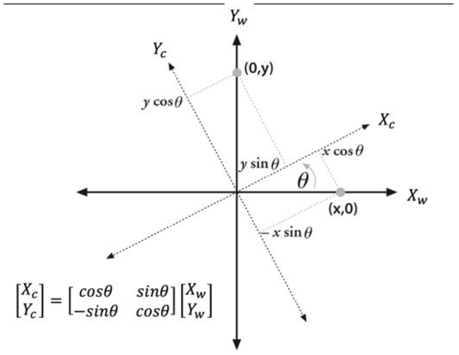

Y_w
Y_c
(0,y)
$y\cos\theta$
X_c
$x\cos\theta$
$y\sin\theta$
θ
X_w
(x,0)
$-x\sin\theta$
$\begin{bmatrix} X_c \\ Y_c \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \end{bmatrix}$

Figure 8: Coordinate transformation (rotation), based on [15]

point, relative to its location before the transformation, can be expressed using trigonometry and written in matrix form:

$$Q_c = R_{z(\theta)} Q_w, \qquad where\ Q_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}, Q_w = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}, R_{z(\theta)} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Because there are three axes in three-dimension space, a rotation about all or any of these axes is possible. Hence, the total rotation vector is the product of the three 2D rotation matrices:

$$R = R_{z(\theta)} R_{y(\varphi)} R_{x(\psi)} \tag{9}$$

The rotation matrix has the property $R^T = R^{-1}$ and the inverse transformation can be achieved with [15]:

$$Q_w = R_{z(\theta)}^{-1} Q_c \tag{10}$$

The second offset between two coordinate systems, the translation, is expressed in the translation vector (t). The translated location of any point Q_w in the world coordinate system can be determined in the camera coordinate system as follows:

$$Q_c = Q_w - t, where\ t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{11}$$

To understand the upcoming derivations in this paper, translation vector and rotation matrix are written into a single 3x4 matrix:

$$W = [R \mid t] \tag{12}$$

2.2.4 Planar homography

Planar homography describes the projective mapping of two planar surfaces to one another. The *homography matrix (H)* maps a point in world coordinates (Q_w) to a point in image coordinates system (q) in a mathematical operation called *projective transformation* [15]. It considers the camera intrinsics as well as the geometric relation (extrinsics) between the two coordinate systems.

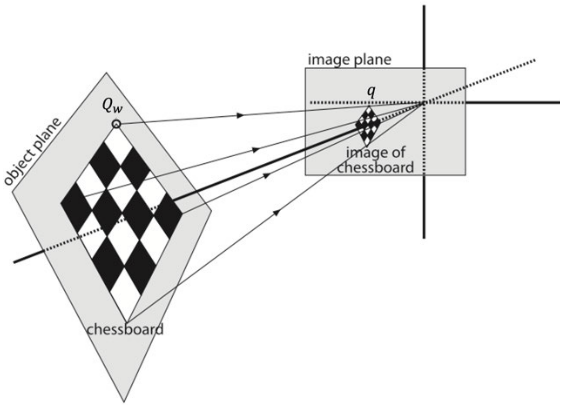


Figure 9: Projective transformation, based on [15]

Different from the notation in equation (5), a scale factor (s) is considered on the right side of the equation to provide a required constraint for the system of equations to be solvable [18].

$$q = sHQ_w = sMWQ_w \tag{13}$$

To provide a better understanding, the equation is written in full using homogenous coordinates. This time, the world coordinates are homogenous as well, which is needed for the multiplication of W_{3x4}:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = s \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [r_1\, r_2\, r_3\, t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{14}$$

By definition, planar homography maps two planes to one another. As a result, all three-dimensional points on the planar surface ("object plane" in Figure 9) must have one equal coordinate, for instance, $Z_w = 0$. Equation (14) can thus be simplified to:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = s \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [r_1\, r_2\;\, t] \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{15}$$

Using this assumption, the homography matrix is a 3x3 matrix, providing a homogenous system of equations that maps a three-dimensional point in world coordinates to a two-dimensional point in image coordinates.

Knowing the 3D world coordinates of multiple points of the same plane, as well as their projected coordinates in image coordinates, allows one to compute one homography matrix for each point. Doing so for multiple points in the same plane, multiple homography matrices are computed. The resulting system of equations can be solved for the camera intrinsics. The details of this procedure are beyond scope for this paper.

For this project, images of a chessboard with n=9 rows and m=6 columns are used. The chessboard corners, whose (planar) world coordinates are related by a horizontal and vertical offset of the length of a chessboard's square, provide enough points per image to solve for

2.2.5 Field of view

When setting up a (stereo) camera system, one of the first steps is the selection of a *field of view* (FOV). At the beginning of this paper, in equation (1), it was discussed that, for the idealised pinhole camera model, the image plane is located at distance F from the pinhole. The physics of real lenses or lens-systems are rarely covered in fundamental literature that does not reach beyond the scope of this work, but one essential relation is called the *thin lens law* [18], which defines that the distance between the image plane and the lens (*i*) is generally not equal to the focal length (F) but depends on the distance between the lens and the object in space (Z):

$$\frac{1}{F} = \frac{1}{Z} + \frac{1}{i} \tag{16}$$

That being said, it is generally proposed [19] for a first system definition to determine the FOV assuming the camera to be focussed on an object at infinity ($Z = \infty$), making equation (16) to be $F = i$. Using this assumption and knowing the geometry and distance to an object relative to the camera, the focal length (F) for the camera can be determined.

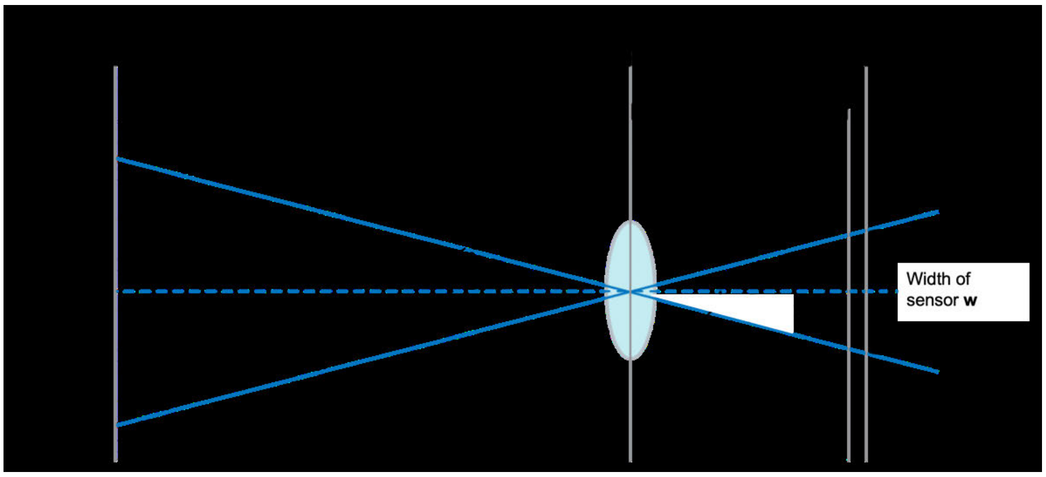


Figure 10: Geometry of the field of view (FOV), based on [19]

For an object of width (D) in at a distance (Z) from the camera's lens, the required horizontal FOV noted using angle $\frac{\Theta_{\mathrm{H}}}{2}$ can be determined:

$$\frac{\Theta_{\mathrm{H}}}{2} = \arctan\left(\frac{\frac{D}{2}}{Z}\right) = \arctan\left(\frac{D}{2Z}\right) \tag{17}$$

In equation (17), the geometry is specified in world units. Similarly, such equation can be defined for the right side of the lens using the sensor width (W) and the focal length (F) in world units:

$$\frac{\Theta_{\mathrm{H}}}{2} = \arctan\left(\frac{W}{2F}\right) \tag{18}$$

Knowing $\frac{\Theta_{\mathrm{H}}}{2}$ from (17) for a given geometry, equation (18) can be used to determine one of the required camera properties. These cannot be chosen arbitrarily but must be chosen from a range of commercially available sensor sizes and focal lengths. This relationship can be determined for the vertical FOV analogously:

$$\frac{\Theta_{V}}{2} = \arctan\left(\frac{H}{2Z}\right) \tag{19}$$

2.2.6 Depth of field

In real camera lenses, the *aperture* defines through which diameter of the physical lens light can enter the camera [20]. A lens' available aperture settings are defined as *f-number (N)*, which is the quotient of focal length (F) and available aperture diameter (D):

$$N = \frac{F}{D} \tag{20}$$

The thin lens law in equation (17) defines that a *plane of focus* exists for a particular focal length (F) for which the light rays are focussed onto the image plane, where they intersect and thus a single point is projected. The object is in perfect focus.

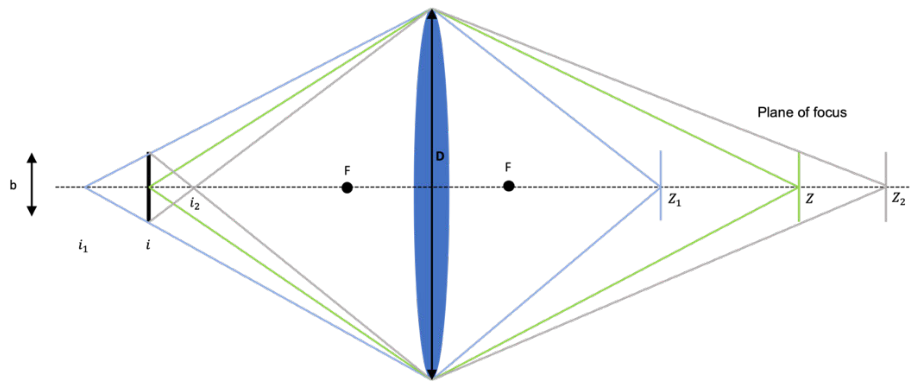


Figure 11: Depth of field geometry, based on [21]

Objects closer or farther from the camera, compared to the plane of focus, form a *blur circle* (also *called circle of confusion*) on the image plane and are thus not perfectly in focus. Alternatively, when selecting a camera configuration, defining the maximum allowed size of the blur circle defines the range for a known focal length (F), in which the projection of any point is sufficiently accurate, thus smaller than the blur circle. This range is called *depth of field (*DOF):

$$Z_2 - Z_1 = \frac{2ZF^2bN(Z-F)}{F^4 - b^2N^2(Z-F)^2} \tag{21}$$

The derivation is shown in the appendix on page 70. Equally, the range limits can be calculated using equations (34) and (35) to be:

$$Z_1 = \frac{ZF^2}{bN(Z-F)+F^2} \tag{22}$$

And

$$Z_2 = \frac{-ZF^2}{bN(Z-F)-F^2} \tag{23}$$

2.3 The mathematics of stereo cameras

The stereo camera arrangement that is used in this paper is called *frontal parallel* (or "*simple stereo system*") and is shown in Figure 12 This particular arrangement features two *identical* cameras that are placed coplanar and horizontally offset from one another. Such placement is not possible in reality, because of manufacturing inaccuracies causing different intrinsics and distortion parameters for two cameras of the same kind [15].

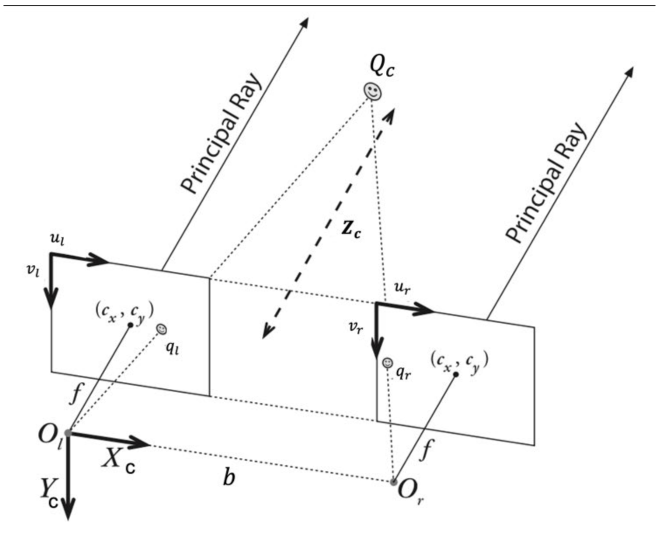


Figure 12: Simple stereo system, based on [15]

Furthermore, mounting the two cameras manually produces inaccuracies and an exact coplanar placement cannot be achieved. This chapter summarises the steps that are required to create a mathematical model, which allows one to transform a real-world-setup into the idealised model.

2.3.1 Stereo camera geometry

Two cameras with identical intrinsics are displayed in Figure 12, using the same geometry that was introduced in Figure 6, horizontally offset by a *baseline* (*b*). Let us assume, for now, that the world coordinate system and camera coordinate system are identical. The camera coordinate system for this stereo setup has its origin in the principle point of the left camera. In computer vision the left camera is usually defined as the main camera. A point Q_c is projected onto the left and right images planes at coordinates $q_l = [\,u_l \;\; v_l]^T$ and $q_r = [u_r \; v_r]^T$ Looking at point Q_c from the left image plane, it's depth (Z_c) cannot be determined because it could lie on any point on the ray $\overrightarrow{O_l Q_c}$. However, it can be calculated when combining the information provided by the two cameras.

Using equation (3), the geometric relation can be defined for both cameras as:

$$u_l = f_x * \left(\frac{x_c}{z_c}\right) + c_x \quad \text{and} \quad v_l = f_y * \left(\frac{y_c}{z_c}\right) + c_y$$

And:

$$u_r = f_x * \left(\frac{X_c - b}{Z_c}\right) + c_x \quad \text{and} \quad v_r = f_y * \left(\frac{Y_c}{Z_c}\right) + c_y$$

Solving these equations for X_c, Y_c and Z_c, assuming that c_x, c_y, f_x and f_y are the same for both cameras results in:

$$X_c = \frac{b(u_l - c_x)}{(u_l - u_r)} \;, Y_c = \frac{b * f_x(v_l - c_y)}{f_y * (u_l - u_r)} \; and \; Z_c = \frac{b * f_x}{(u_l - u_r)} \tag{24}$$

For a known baseline (b) and known camera intrinsics, the depth (Z_c) to an object can thus be determined. While in some cases the *depth* to an object is of interest, in other cases the *distance* (R) to the object is more relevant. It can be calculated as follows:

$$R = \sqrt{X_c^2 + Y_c^2 + Z_c^2} \tag{25}$$

2.3.2 Stereo calibration

The goal of stereo calibration is to identify the geometric relationship between the two stereo cameras, which is expressed in the translation vector (t) and the rotation matrix (R). The notation of both was discussed for the projective mapping of a chessboard and a single camera in chapter 2.2, only now it maps both cameras to one another. Furthermore, the first coordinate of the translation vector is the baseline (b) that is needed to determine the depth to an object in equation (24). Both matrices are also needed for the next step, which is the rectification.

Two image planes that are tilted towards one another are displayed in Figure 13. The projected point Q_c in camera coordinates is visible on both image planes q_l and q_r. Since the two cameras are rotated towards each other, the centre of projection of the right camera (O_R) represents another point in space that is projected onto the left camera's image plane at point (e_L). This point is called the *epipole.* The right image plane also has one epipole (e_R) where the left camera's centre of projection is projected onto the image plane. The two epipoles as well as point Q define a plane in space, which is called *epipolar plane* [15].

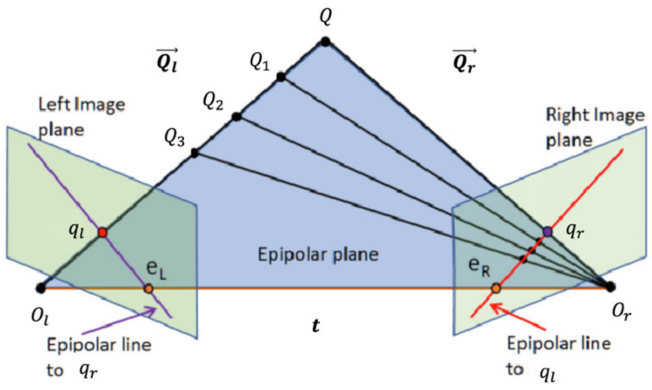


Figure 13: Epipolar geometry, based on [22]

When viewed from the left camera alone, the depth of point Q cannot be determined, because it could lie anywhere on the ray $\overrightarrow{O_l Q}$. This is highlighted in Figure 13 using points Q1 through Q3, which all prove to be possible locations of point Q when looking at it from the left image plane. One characteristic, however, is that all these points must, by definition, lie in the epipolar plane and, thus, all rays from the right camera's centre of projection to the points Qi must be in the same plane as well. All these rays project one line on the image plane of the left camera, called the *epipolar line.* The same applies for the epipolar line on the right image plane. Viewed from the left camera, the actual location of point Q in space must be somewhere along the left epipolar line.

Using the characteristics of epipolar planes and epipolar lines, the following relation between corresponding image coordinates of the left and right camera can be defined (see page 71 for the derivation):

$$[u_r \; v_r \; 1] \, F \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = 0 \tag{26}$$

In equation (26), u and v are the image coordinates in the left and right camera written in homogenous coordinates. The *fundamental matrix* (F) relates the images points of the left and right camera and provides a system of equations that is solved, using additional constraints that are beyond the scope of this paper, for each chessboard view. Furthermore, the fundamental matrix can be split into the two camera matrices, M_r and M_l, as well as the *essential matrix* (E), where $F = {M_r^{-1}}^T E\, M_l^{-1}$. Equation (26) thus becomes:

$$[u_r\ v_r\ 1]\, {M_r^{-1}}^T E\, M_l^{-1} \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = 0 \tag{27}$$

Since both camera matrices are known, as they have been determined during the single camera calibration procedure, the essential matrix can be solved for. Knowing the essential matrix, the translation vector t and the rotation matrix R are computed. The resulting translation vector and rotation matrix transform a point in the image plane of the right camera to the image plane of the left camera. When this is done for all points in the image, the resulting images are coplanar, which was the first, ideal, characteristic of the frontal parallel stereo system. Furthermore, the distance between the centres of projection of the two cameras are related by $t = \left[t_x\ t_y\ t_z\right]^T$, the first value of vector t is equal to the baseline b.

2.3.3 Stereo rectification

The goal of stereo rectification is to transform two images, which after the stereo calibration procedure are coplanar, in a manner that all epipolar lines of both images are horizontal and colinear, i.e., have the same vertical coordinate. Given the pose and position of the epipolar lines these images are called *row aligned (see* Figure 15*)*. Doing so will prove to have a considerable advantage during the depth computation procedure, which will be discussed in the next step. The most relevant operations for this paper during the stereo rectification procedure shall be outlined below.

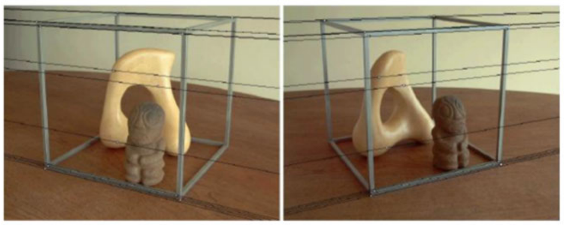


Figure 14: Example of an image before rectification [23]

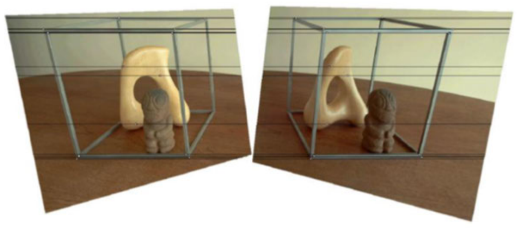


Figure 15: Example of the same image after rectification [23]

Bouguet's algorithm [15] starts with the rotation matrix R, which was computed during stereo calibration and rotates, in this case, the right camera's image into the left camera's image plane.

Instead of rotating one image plane, Bouguet splits the rotation matrix R into two, rotating both images towards one another :

$$R = r_L + r_R \tag{28}$$

In equation (28), the rotation of the left camera is r_L and for the right camera it is r_R. The row alignment of the images is achieved by rotating the images around their centres of projection in a manner that aligns the epipolar lines with the (horizontal) baseline b that connects the two cameras, which is defined in coordinates in translation matrix T. This rotation is defined in the rectification matrix R_{rect}. The required mathematical derivation is beyond scope for this paper. The two single rotation terms define the total rotation matrices:

$$R_L = R_{rect} r_L \quad and \quad R_R = R_{rect} r_R \tag{29}$$

Row aligned images are thus achieved by performing a coordinate transformation, first rotating the images towards one another followed by a rotation around their respective centres of projection. To achieve the required equal focal lengths, the rectified images are scaled and the new camera intrinsics modified accordingly. The equal focal lengths f_x, f_y and the equal displacements c_x, c_y are noted in the rectified camera matrix M_{rect}.

2.3.4 Depth computation

Once the stereo rectification has produced two row-aligned and coplanar images, the depth computation can begin. Equation (24) shows that for a frontal parallel stereo setup, the depth can be computed for a known focal length (f) and known horizontal image coordinates (u_l, u_r) of the projected point. The displacement of the horizontal coordinates of the projected point is called disparity $d = u_l - u_r$. To understand what disparity is, one shall examine the following images.

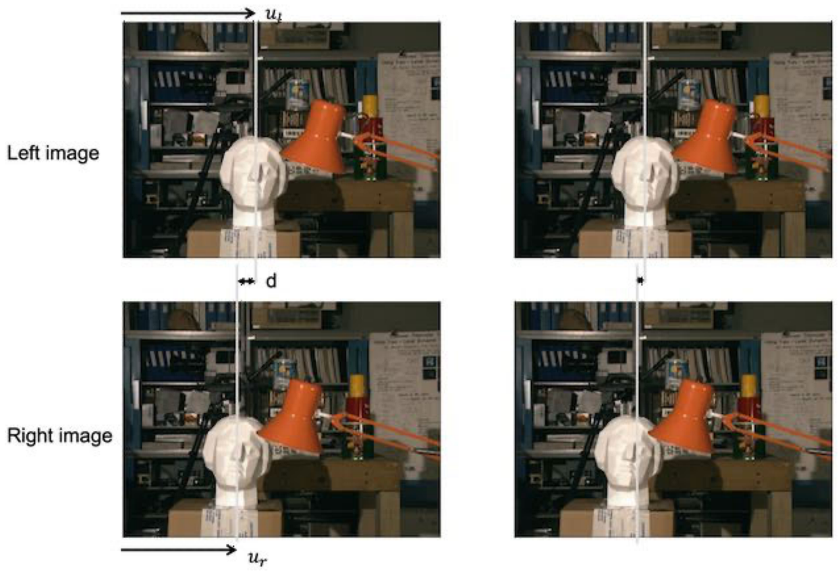


Figure 16: Visual representation of disparity, based on [24]

An object close to the camera, such as the centre of the sculpture's nose in Figure 16, left, is visible in the left image at coordinate u_l from the left edge of the screen, while in the right image it is located at u_r from the left edge. The object is close to the camera, the depth to the object is small, and the disparity is large. On the other hand, an object at the back, for instance the centre of the vertical shelf pillar, produces a smaller disparity. The depth to the object is larger. If the depth to the object goes towards infinity the disparity goes towards zero and the depth cannot be calculated. In practice, if an object is too close to a camera, it might not be visible in both camera frames and no disparity or depth can be determined either.

Since the depth to an object is determined by the disparity of horizontal image coordinates, the advantage of rectified images becomes clear. Let's recall that the projected point must lie somewhere on the epipolar lines of the two cameras. These now have the same vertical coordinate in both images. To find the disparity of a certain feature or point, the computer only has to search along the epipolar line in the right image, instead of searching the entire image. This reduces the computational effort considerably.

In this paper, the initial point in the left camera, for which the disparity and depth will be calculated, is determined using object detection, which is discussed in the next step. Let's summarise the above-mentioned procedure quickly.

2.3.5 Conclusion

The goal of the *single camera calibration* procedure is to compute the camera intrinsics as well as the distortion coefficients, which are parameters that describe individual camera inaccuracies from the ideal pinhole camera model.

During the *stereo calibration process,* the intrinsics and distortion coefficients are used to determine the geometric relationship, the *translation* (and *baseline*) and *rotation* between the two cameras.

Third comes the *rectification process*, which, given the geometric relationship between the two cameras, computes transformation matrices that produce coplanar and row-aligned images, which are called *rectified images*.

The *rectified images* are used to compute *correspondences* of features in the left image and the right image. Knowing the disparity of a projected point, the depth to the object can be computed.

2.4 Object detection

A grayscale image is read by the computer as a matrix, with the number of rows and columns defined by the vertical and horizontal resolution of the image (the number of single pixel elements in the respective direction). Each matrix element of an 8bit grayscale image has a value between 0 and 255 representing the intensity of the pixel element. An 8bit colour image is represented as a tensor with three channels. Each channel stores the colour intensity for one of the colours blue, green, red (BGR).

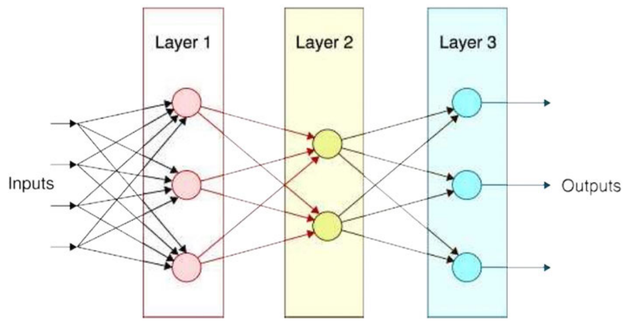


Figure 17: Fundamental architecture of a neural network [25]

The task of object detection requires a computer to analyse information in an image and, in our case, perform the *classification* of objects in the image. To do so, it must make a decision using the available data, which is a task that defines the disciple of *machine learning*. Using an existing data set, a computer can use statistical algorithms, such as *k-nearest-neighbour* (k-NN), to compare the characteristics of newly input data with those of existing data points [26]. Knowing a class of the existing data point, the classification (e.g., country, age, height, cost, object) of the new input can thus be approximated. Deep learning is a sub-discipline of machine learning and uses deep neural networks to perform such classifications for instance. Their general architecture consists of an *input layer*, an *output layer* and a number of (*hidden) layers*. In its very essence, information entering the input layer is passed into the *nodes* (“*neurons*”) of the first hidden layer, where various properties of the input are evaluated. More specifically, the properties of the input are compared to known properties of known objects. The output layer then returns the class of the object that the algorithm has found the highest resemblance to. One manner in which the mathematics of the hidden layers can be simplified is that the neurons at the top of the network (on the left side of the layer in Figure 17) evaluate a small piece of the input image, such as one pixel element, regarding its (colour) intensity. Comparing this *weighted* value (called *activation*) to the known intensity of a larger feature, e.g., comparing whether a black pixel element corresponded to a *corner* in correctly classified images, the probability of such correspondence is passed along to the next layer. Neurons in the next layer evaluate the received information and, again, determine the probability that the feature corresponds to a shape, e.g., a black corner is part of a chessboard square, and so on [25] .

In order to allow any neural network to identify known geometry, the weights in all neurons have to be adapted to evaluate the correspondence of input data to the new geometry. This is done during the *training* process, using *supervised learning*. Supervised learning includes providing an input as well as an output to the model and using an algorithm to adjust all weights to maximize the similarity of input and output. The success of a training is measured with the metrics *precision*, i.e., out of all tests, how many matches between known input and output were correct (true positive), and *recall*, i.e., out of all known objects in a known image, how many were correctly identified.

Instead of completing an entire model training for the purpose of this paper, existing pre-trained models can be used and adapted to properly classify new objects in a process called *transfer learning.* This is done, in the case of single-shot-detector models (SSD), by retaining the *backbone* of the network, which extract features, such as lines, corners and shapes (and many more) from the images, while adapting the *head*, which matches these shapes to new objects and classifies these as such. While the architecture of SSD models is far more complex than the simple neuron-layer model described in this chapter, the fundamental principles of feature and object detection are similar. Because this paper uses object detection merely as a supportive function, the above-mentioned concept of neural networks shall be sufficient to understand the work of this paper. A detailed architecture review can be found in [27].

3 Requirements

Now that the fundamentals for this project were discussed, the next step is the definition of requirements for the computer vision system.

Functional requirements

The previous work in [6], which this paper is based on, highlights that the pushback procedure is one of the tasks that requires a considerable amount of money due to the requirement of multiple members of staff. Costs could be reduced by providing a teleoperated version of the *Taxibot*, which is a towbarless tow tractor developed by Israel Aerospace Industries and others and certified by the European Union Aviation Safety Agency (EASA) in 2014. It is currently certified for the operation with Airbus' A320 family as well as Boeing's type B737 aircraft [28].

This paper analyses a concept for a teleoperated version of the Taxibot for the *pushback procedure only*, i.e., the process that starts after the Taxibot has coupled with the aircraft and ends when the aircraft is in a position from which the pilot can navigate the aircraft to the runway. Furthermore, the analysis and development will only consider Airbus' A320 aircraft.

A teleoperated version of the Taxibot must allow the operator to view crucial points of the aircraft and the aerodrome, such as the *wingtips*, the *uplifting system* of the tow tractor, *airport signs*, *paths*, other *aerodrome infrastructure*, the pavement and *any other obstacles* that could damage the aircraft while it is moving and coupled with the tractor. Hence, views of the Taxibot should be given in each direction of movement; *forward, backward* and *sideward* [29]. It shall be the aim of this paper to develop a concept, which provides the operator with such views, as well as distances to said objects. As a result, the software must be able to *detect* the above-mentioned *objects*. These objects shall be detected once they enter a certain area around the aircraft or the tow tractor. Previous work [6] has assumed a visibility of 50m behind the aircraft's tail to be sufficient for sudden breaking manoeuvres, which shall define the distance in which objects must be detected from the aircraft's tail. Since the pushback of the aircraft does not happen sidewards, a detection radius of 1m from the aircraft's wingtips shall be provided. It is important to note that all objects underneath the aircraft, including underneath the wings, must be detected. The resulting geometry requirements are displayed in Figure 54 in the appendix, with arbitrary angles for visibility chosen to begin with. Their analysis will follow shortly.

Moreover, the system should provide guidance to the Taxibot operator regarding the magnitude of change in throttle and steering that needs to be performed to push the aircraft along the intended pushback path, marked by *yellow lines*, in the most efficient manner. Hence, the system should be able to *detect* the *pose and position* of the aircraft relative to the Taxibot to provide the operator with the required information.

An additional requirement shall be that the *real-time* information required to obtain the distances to the above-mentioned objects shall be produced by a *local system on the Taxibot* or at the venue from which it is teleoperated. This has the aim to reduce a dependency from other airport service providers and allow the Taxibot to be deployed ‘out of the box’ without prior airport or aircraft modifications[2] or wire(-less) connections.

Regulatory requirements

All aircraft and aviation products must be proven to be safe and provide their intended functions. The European Union Aviation Safety Agency is one of the regulatory bodies, which provides certification specifications as well as means of compliance that provide guidance on how these specifications can be complied with. The development of aviation products is heavily based on these guidelines. For the concept work in this paper, however, the regulatory requirements are not considered because the software part of the visual aid system shall not be accessible to the pilots of the aircraft. Providing the same visual display to the pilot, including object detection, would require a change to the software of the aircraft, which requires additional certification for each aircraft that uses the Taxibot. Regulatory requirements for autonomous or teleoperated ground vehicles are not included in the certification specification at the time of writing.

[2] Not including the software that is already installed on A320s allowing the pilot to control the Taxibot or the potential need for further certification

4 Implementation

4.1 Hardware

The before-mentioned, rough geometry requirements for the teleoperated Taxibot concept (see Figure 54 in the appendix for reference) will be analysed in depth in this chapter. Furthermore, the required camera parameters to provide the desired views will be determined. First, the horizontal frontal view shall be analysed.

4.1.1 Geometry analysis

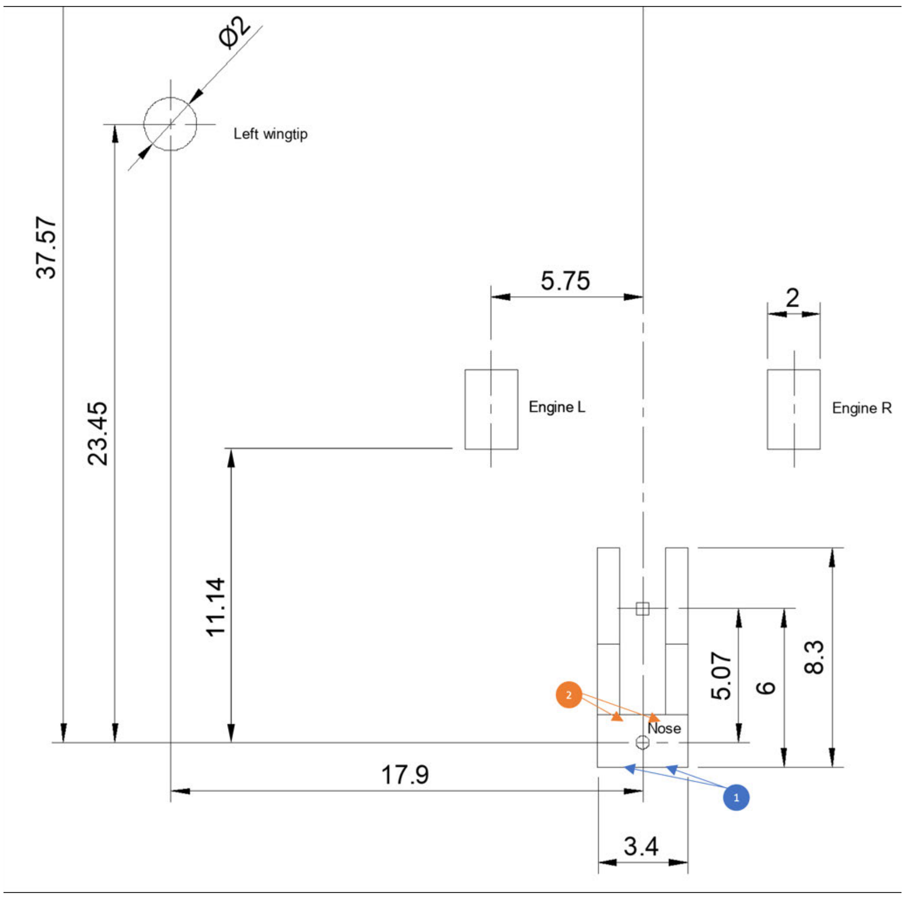


Figure 18: Geometry of Taxibot and aircraft, units in [m]

Figure 18 displays the underlying geometry, to scale, using publicly available dimensions of an A320neo [30]. Dimensions for the Taxibot are taken from [6], which assumes it to have the standard dimensions of a towbarless tractor $(h = 2m,\ l = 8{,}3m, w = 3{,}4m)$. Furthermore, it is assumed that the uplifting system is located at circa 6m from the Taxibot's back end. In general, the aim should be to avoid the usage of wide-angle lenses, which introduce higher distortion compared to *regular* lenses. Regular lenses are defined to be such with a focal length F longer than 35mm [31].

First, a camera setup with a baseline b=0,75m is analysed where the cameras are placed at the very back of the Taxibot's ceiling, marked with (1) in Figure 18. The left- and rightmost key points that must be detected by both cameras are the safety radii of the wingtips. That is, the right camera must have the left wingtip in its field of view, since if it is the case, then the left camera will geometrically have it in its view too. The required horizontal field of view is thus, from equation (17):

$$\frac{\Theta_H}{2} = \arctan\left(\frac{18{,}9m + \frac{0{,}75m}{2}}{24{,}38m}\right) = 38{,}33°$$

Since it is desired to have a high resolution for camera views, which need to detect far distances, the sensor and camera in Table 1 are selected.

Table 1: Technical specification for Basler ace 2 [32]

Basler ace 2 - a2A4096-30ucPRO	
Sensor size (H x V) $[mm]$	11,2 x 8,2
Pixel size (H x V) $[\mu m]$	2,74 x 2,74
Resolution (H x V) $[pixels]$	4096 x 3000
Price [€]	1399,00

The available horizontal FOV for a *regular* lens, using said camera is:

$$\frac{\Theta_H}{2} = \arctan\left(\frac{11{,}2mm}{2 * 35mm}\right) = 18{,}18°$$

A regular, non-wide-angle lens can thus not provide the required field of view. Instead, using the same sensor and camera, the usage of a wide-angle camera must be considered. However, it's placement shall be analysed for the inside of the cockpit. Such is advantageous, as the devices will be located in a dry environment with a windshield wipers, that can be made to have a constant temperature, thus improving the view quality for the cameras and increasing their lifetime. For the new required FOV, due to the changed camera placement, at 4,3m from the uplifting system, marked by (2) in Figure 18 is $\frac{\Theta_H}{2} = 40{,}36°$. Using this new FOV as well as the known sensor parameters, the maximum allowed focal length F to be searched for in a lens is:

$$F = \frac{W}{2 * \tan\left(\frac{\Theta_H}{2}\right)} = \frac{11{,}2mm}{2 * \tan\left(40{,}36°\right)} = 6{,}59mm$$

The lens closest to that is Basler's $F = 6mm$ wide-angle lens with the specification seen in Table 2. It provides a horizontal FOV of is $\frac{\Theta_H}{2} = 43{,}03°$.

Table 2: Technical specification Basler Kowa Lens LM6HC [33]

Kowa Lens LM6HC	
Focal length $[mm]$	6
F-number [-]	1,8
Price [€]	549,00

To provide a reproducible DOF, one plane of focus shall be chosen, and potential autofocus features must not be used. The relation between the depth of field and the plane of focus for the chosen lens is displayed in Figure 19. A maximum allowed blur circle of $b = c = 2{,}74\mu m$ is chosen, which is the size of one pixel element.

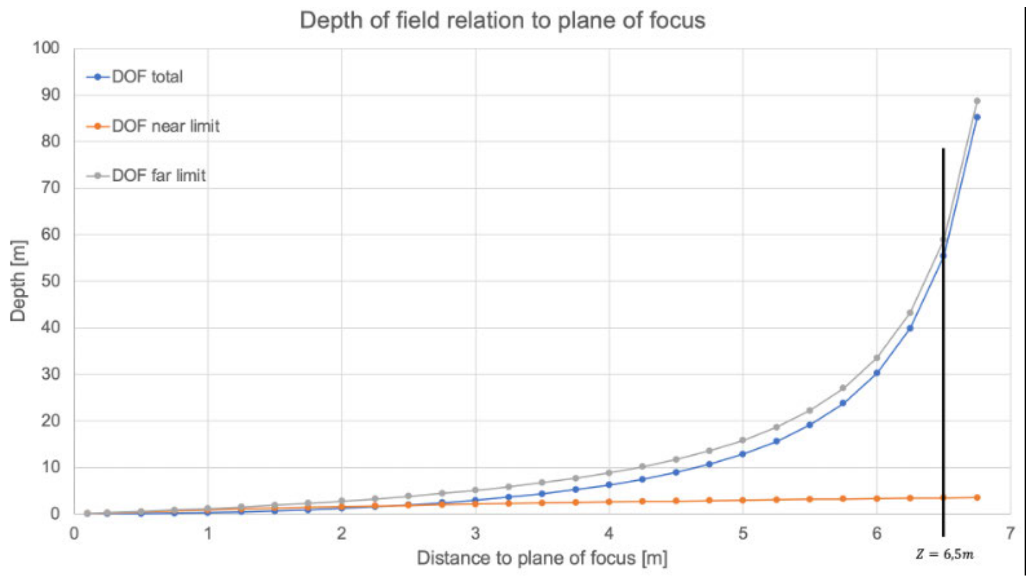


Figure 19: Relation DOF to plane of focus for F=6mm, N=1,8, c=2,74um

It becomes evident from Figure 19 that increasing the distance of the plane of focus increases the available depth of field considerably for this sensor and lens combination. With this particular stereo setup mounted inside of the Taxibot's cockpit, the plane of focus should be selected to be close to the front face of both engines ($Z = 10{,}37m$), as these are very important key points for the navigation of the aircraft, for which a precise position must be calculated. Furthermore, a near detection limit shall be chosen, which allows depths to be detected accurately at the distance that the front landing gear is located at ($Z_1 = 4{,}3m$). This would allow for the system to be used during the coupling procedure as well. A plane of focus of $Z = 6{,}5m$ is selected, resulting in a near DOF limit of $Z_1 = 3{,}44m$, a far limit of $Z_2 = 58{,}92m$ and a depth of field of $Z_2 - Z_1 = 55{,}48m$[3].

To detect objects at farther distances, another stereo pair should be placed at a lower location, overseeing the area behind the aircraft. Furthermore, two cameras must be placed to provide a

[3] These calculations were made based on the assumption that the thin lens theory applies for wide-angle lenses as well, which differentiate themselves from regular lenses by the curvature of the glass and not it's thickness. A plausibility check shows that Tesla uses wide-angle lenses to detect depths of up to 60m, which increases the plausibility of these results: https://www.tesla.com/autopilot

view towards the front of the Taxibot. The selected cameras, their FOV and DOF are summarised in Table 3. A sketch highlighting all camera setups, to scale, is shown on page 75 in the appendix. Sensors and cameras are the same, as the one previously described.

Table 3: Summary of further required camera configurations

Further camera configurations				
Lens	**Properties**	**Direction of view**	**FOV**	**DOF**
Kowa Lens LM6HC	$N = 1{,}8$ $F = 6mm$	Away from aircraft	$\frac{\Theta_H}{2} = 43{,}03°$	$Z = 6m$ $Z_1 = 3{,}29m$ $Z_2 = 33{,}55m$
Basler Lens C23-3520 [34]	$N = 2{,}0$ $F = 35mm$	Towards aircraft	$\frac{\Theta_H}{2} = 9{,}09°$	$Z = 60m$ $Z_1 = 34{,}7m$ $Z_2 = 82m$

To provide a list of costs, which is commonly included for any concept work, further devices and installations would have to be analysed, which are beyond the scope of this paper. One of these is the computer, which would be used for the processing of all camera livestreams. It could be possible to avoid the installation of powerful computers in all Taxibot's by installing a less powerful device, which would be capable to transmit the captured images to a remote facility, from where the Taxibot is teleoperated. Since the computationally expensive object detection and depth computation task is only required, when the Taxibot is being teleoperated, the facility could use one operator and one powerful computer to connect to the Taxibot, read its livestream and perform the intended task. Table 6 in the appendix provides a cost calculation for the required cameras and lenses described in this section.

4.1.2 Prototype

The functional requirements on page 25 defined that the pose and position of the aircraft shall be detected by the computer vision system. Given the previously discussed camera configuration and the respective depths of field, only the wide-angle cameras with the FOV towards the aircraft are plausible to be used for the detection of the aircraft.

For this project work, a hardware prototype, featuring two horizontally offset cameras (simple stereo system) will be used to simulate the functions of the wide-angle cameras. Raspberry Pi cameras are not used, since the computational resource required to run an object detection and depth computation algorithm are beyond those provided by standard raspberry computers. The two webcams used are Logi and Logitech C920HD cameras. They offer a resolution of 1980 x 1020 pixels, but further technical specifications found on the internet could not be verified. Moreover, both cameras have an auto-focus feature, which does not allow a useful depth of field calculation since the plane of focus changes constantly.

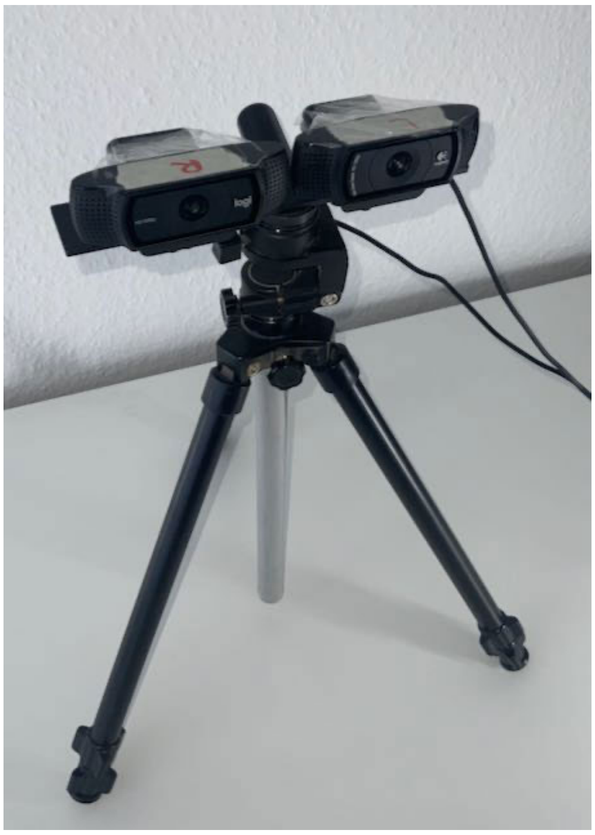


Figure 20: Simple stereo hardware prototype

The two cameras are screwed to a tripod, which prevents them from changing their positions relative to each other (see Figure 20 and page 76 in the appendix). The camera calibration is done using a printed chessboard version in DIN A4 as well as on an iPad Pro with a resolution of 264ppi. A computer, running on macOS is used for debugging as well as testing of computation time for the algorithm (see Table 7 in the appendix). Unfortunately, neither a Windows nor a Linux machine are supported in the software implementation, due to time limitations.

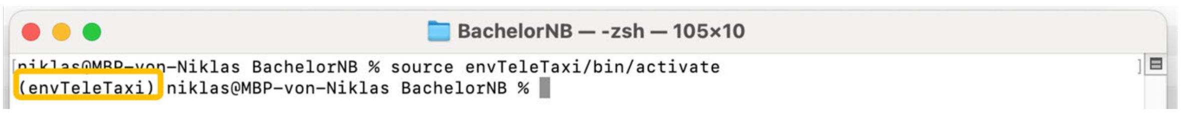
4.2 Software

This chapter will cover the software implementation of a stereo vision system using two cameras, which shall detect and compute the distance to two aircraft engines. Key steps and functions to achieve this goal will be summarised.

4.2.1 Installation

The start of a new programming project should begin with creating a new virtual environment. It allows packages and specific versions of these to be installed in this environment without effecting other programs on the same machine, which might require different package versions. The following steps require python version 3 and the open-source package manager “pip” to be installed globally on the machine by downloading it from the web [35] [36]. Furthermore, “git”, a package allowing repositories to be cloned from GitHub must be installed from [37].

The first step to do so is to navigate any desired folder on the machine, from where the depth computation program will be run. This folder should be empty and must have a recognisable name, such as “*BachelorNB*”. Next, the *shell*, a human machine interface allowing the user to perform various operations on the machine, must be opened. Now, using *cd ‘pathToBachelorNB’* the current working directory of the computer is changed to the newly created one.



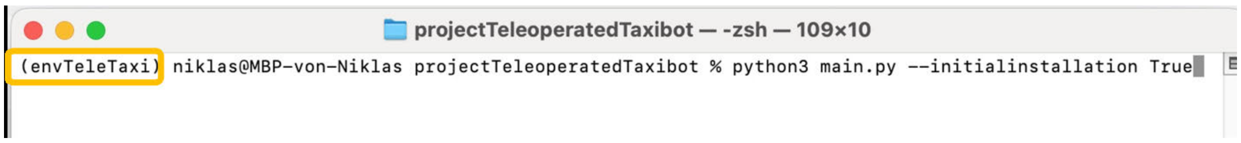
```
niklas@MBP-von-Niklas BachelorNB % source envTeleTaxi/bin/activate
(envTeleTaxi) niklas@MBP-von-Niklas BachelorNB %
```

Figure 21: Display of activated virtual environment

A new virtual environment, here using the name “envTeleTaxi” is created with *python3 -m venv envTeleTaxi*. Then, key files required for the program are cloned to the directory from GitHub using *git clone https://github.com/joshba06/projectTeleoperatedTaxibot.git*. The virtual environment is then activated with *source envTeleTaxi/bin/activate*, see Figure 21. Next, a jupyter notebook kernel has to be installed, which allows python code (segments) to be executed and analysed step by step in an appealing graphic environment in the web browser [38], which will be used for the model training as well as the stereo algorithm analysis. This is done using the shell command *pip install ipykernel* and, in a separate call, *python3 -m ipykernel install --user --name=teleTaxi*.

Now, the *wget* package must be installed manually. It allows content to be downloaded from the web, using URLs. On macOS, the homebrew package installer must be installed first from [39] which describes how this is done. Then, using *brew install wget* installs said package in the active virtual environment.

Back in the shell, *cd projectTeleoperatedTaxibot* is used to navigate into the previously cloned folder. The file *userSettings.py* is opened with any text editor. In this file, which is the basis for the installation of the following packages, the variables *homePath* and *pathLocalInterpreter* must be modified appropriately. Essentially, the file path to the left of /BachelorNB/ must be changed to the path that leads to said folder. By activating python in the shell, in which the desired environment is activated, this path can be printed using commands *import sys* followed by *sys.executable*.



```
(envTeleTaxi) niklas@MBP-von-Niklas projectTeleoperatedTaxibot % python3 main.py --initialinstallation True
```

Figure 22: Command to perform initial installation in an activated virtual environment

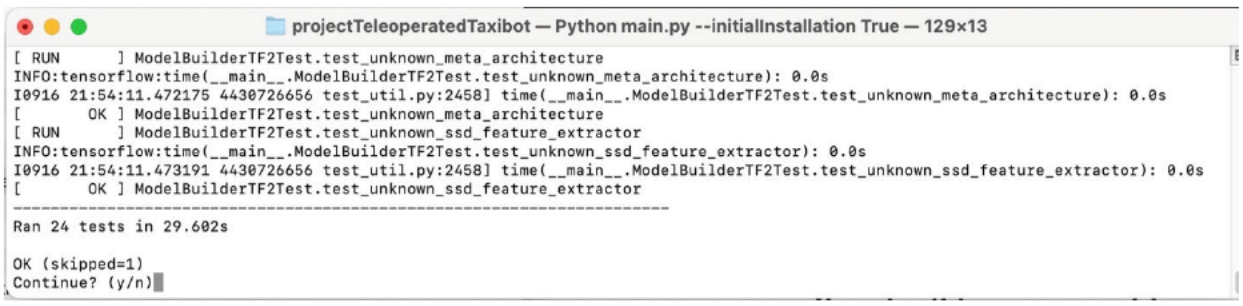
Now, entering *python3 main.py --initialInstallation True*, the main script is executed in the “first installation” mode. This script creates the folder structure pictured in Figure 59 in the appendix. Then, the routine installs all required packages, which are summarised in Table 4 and which functions were, in part, covered by previous papers [40]. The procedure for the installation of packages required for object detection is based on the tutorial available in [41] and [42].

The sequence in which the installation of some packages takes place has an important role. First, the package GitPython is installed, which allows cloning of repositories from GitHub from within a python file. More advanced packages that are installed afterwards include Tensorflow 2, which is a Google-built platform for machine learning, through which’s APIs machine learning tools can be accessed [43]. Tensorflow’s Model Garden, maintained by Google, is an optimised directory and file architecture for state-of-the-art machine vision and can be cloned from [44]. Afterwards, supporting packages “Common Object in Context API” (COCO) and Google’s protocol buffers (also called *protoc* or *protobuf*) are downloaded and installed from [45] and [46]. Then, the Tensorflow object detection API, which is a framework built by Google for Tensorflow to provide means for easy training and deployment of models, is installed from the “*setup.py”* file in the downloaded model garden.

Table 4: List of required packages

Seq.	Package name	Function
1	wget	Download content and files from webservers
2	GitPython	Cloning of repositories from GitHub
3	TF model garden	Implementation and workflow of state-of-the-art machine vision architecture
4	Protobuf	Mechanism for operating structured data
5	COCO Api	Image dataset for object detection
6	TF object detection API	Framework interacting with Tensorflow to train and deploy object detection models
6	Tensorflow (TF)	Platform for machine learning, incl. specific libraries
7	opencv-contrib-python	Image processing
-	matplotlib	Creation of figures in jupyter notebook
-	pandas	Library for data analysis
-	albumentations	Multiplication and modification of images

The installation of the object detection API includes the installation of Tensorflow 2, which in turn includes the installation of further supporting packages and libraries, which shall not be the focus of this section. One important consideration, however, is that this installation automatically installs some versions of the open source computer vision library (OpenCV) that cause issues in later steps of the program. Of the available versions "*opencv-python*", "*opencv-python-headless*", "*opencv-python-contrib*" and "*opencv-python-headless-contrib*" only "*opencv-python-contrib*" must be installed. As consequence, the program "main.py" uninstalls all versions that are not required and updates the needed version if necessary. In the last step, the installation of the object detection API is checked using a built-in script provided by Google, which returns "OK", if the installation of the object detection API and thus all previously install packages, has been successful. The user should confirm the system report by entering (y) in the shell.



```
projectTeleoperatedTaxibot — Python main.py --initialInstallation True — 129×13
[ RUN      ] ModelBuilderTF2Test.test_unknown_meta_architecture
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
I0916 21:54:11.472175 4430726656 test_util.py:2458] time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_meta_architecture
[ RUN      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
I0916 21:54:11.473191 4430726656 test_util.py:2458] time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
----------------------------------------------------------------------
Ran 24 tests in 29.602s

OK (skipped=1)
Continue? (y/n)
```

Figure 23: Display of successful installation

That being said, the script occasionally fails when importing 'git' just after having it installed for some reason. Re-entering the script in initial installation mode fixes this issue.

4.2.2 Model training

The deep learning model that was trained during this project work can be found in _0_UserInput/trainedModels_. The script that performs the training is called _modelTraining.ipynb_ and is written in form of a jupyter notebook. To open the script, the virtual environment must be active. Typing the command _jupyter notebook_ into the shell starts the jupyter kernel. A URL, starting with _http://localhost:8888_ is returned, which must be pasted into a web browser.

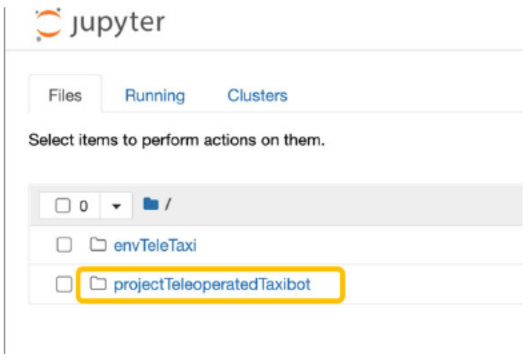

jupyter
Files
Running
Clusters
Select items to perform actions on them.
0
/
envTeleTaxi
projectTeleoperatedTaxibot

Figure 24: Selection of project folder im jupyter notebook

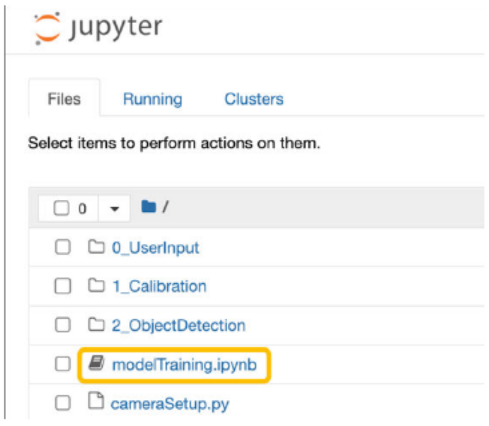
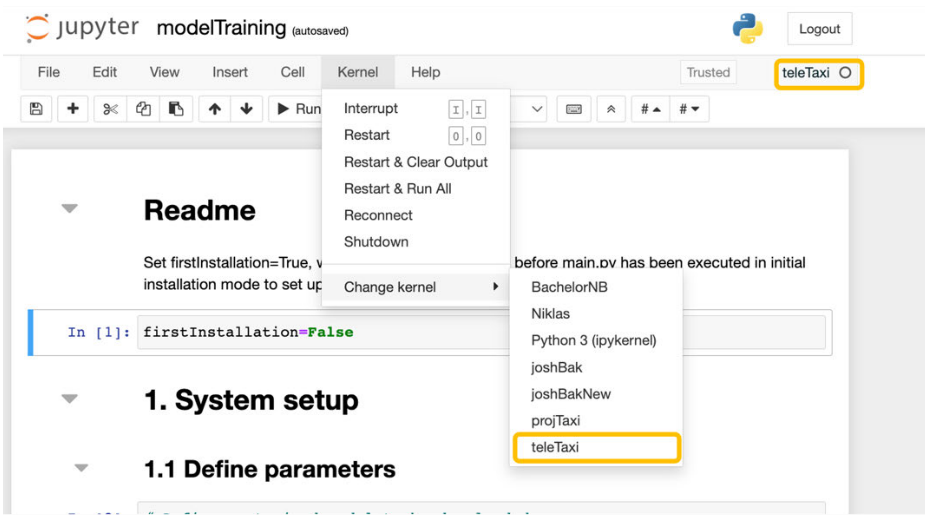


Figure 25: : Selection of training script in jupyter notebook

Once the notebook environment has opened, the correct file must be selected, as indicated in Figure 24 and Figure 25. Furthermore, the correct kernel must be selected to access the previously installed packages (see Figure 26). By default, selecting the menu register _Cell→Run_

All will run the complete preparation and transfer learning procedure, without the user having to intervene in any way, as along appropriate background and object images are provided in *0_UserInput/backgrounds* and *0_UserInput/objects/'classname'.* A new class must also be defined in the *userSettings.py* file. The following section will summarise what the routine does.

To begin with, a pre-trained single-shot-detector (SSD) model has to be downloaded. A list of available models compatible with Tensorflow 2 can be found in [47]. For this project work, the *"SSD MobileNet V2 FPNLite 640x640"* was selected, primarily due to its fast computation (39 [ms]) as well as it's acceptable mean average precision (28,2 [%]). The pre-trained model is downloaded into the directory *2_ObjectDetection/2_Tensorflow/workspace*, which is the folder where all files for the transfer learning are stored. The transfer learning for this model requires a dataset of images featuring the object that the model shall be able to detect. Furthermore, XML files defining the image's names, the class(es) of the object in the image as well as the image



Readme

Set firstInstallation=True, v before main.py has been executed in initial installation mode to set up

```
In [1]: firstInstallation=False
```

1. System setup

1.1 Define parameters

Figure 26: Selection of correct kernel in jupyter notebook

size and the *bounding box* size are required. A bounding box is a rectangle that is drawn onto an image, marking the position of an object of interest. For the specific SSD model that is trained for this paper, images of various backgrounds were created in the size of 640x640 pixels.

Images of aircraft were downloaded from the internet and the engines extracted from the photos using photoshop. The objective of the model training in this paper is to identify an object to determine its location. It shall be sufficient to identify the exact object image, which was used in the model training. A real software prototype should be trained thoroughly using appropriate backgrounds, similar to the scene seen at airports. Furthermore, the extraction of engines using photoshop leads to unrealistic edges around the engines, which might lead to detection issues on different engine images than the ones that were used for training. For this project this is not considered an issue.

The backgrounds and object images that have been saved in the above-mentioned directory are multiplied and modified using the script *imageProcessing.py*, which is a slight modification of the script *randomTrafficSign.py*, provided in [40]. The modified script automates the partitioning of images as well as their placement in the respective training and testing folders. The routine *randomTrafficSign.py* uses the *Albumentation* package, which modifies existing images by changing parameters, such as their rotation, saturation and brightness, to simulate a variety of environments that the deep learning model is trained on. For the purpose of this paper, the focus of the modifications was placed on the (RGB) colour shift to allow the possibility for engines of different colours to be detected. Furthermore, a variety of angles at which the engines are visible in images is generated to improve object detection from different perspectives. For a given parameter, the desired probability of it being applied to the image is specified in Table 5. Samples of the images after modification are displayed on 80 in the appendix.

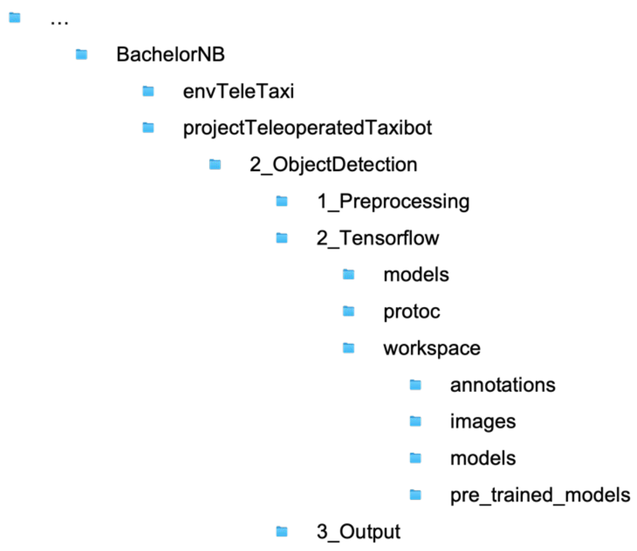
Table 5: Parameters modified using Albumentation

Parameter	Value (max change)	Probability
Brightness	30%	33%
Contrast	30%	
RGB colour shift	30%	66%
Rotation	45°	66%

Furthermore, the scale of the object images is randomly modified by the routine up to pre-defined scale limits. Compared to the limits selected in the related work in [40], which's object detection function had the purpose of identifying street signs in relatively far distances, the need to identify large aircraft engines at a close distance requires larger limits. This was found to be true after testing a model trained on images using the scale limits from [40] on large engine images, which did not detect the engines. Table 5 displays the scale limits used in the before-mentioned project as well as the ones used for this paper.

In addition to the modification of objects and images, each object image is multiplied by factor of 250 to provide a large dataset and variety.

The XML- and image files resulting from the multiplication and modification steps are first stored in the directory 2_ObjectDetection/1_Preprocessing. Then, the image and XML pairs are selected randomly and partitioned into the directories testing and training, seen in Figure 27. Of all available images, 85% are used for training and 15% for testing.



```
...
    BachelorNB
        envTeleTaxi
        projectTeleoperatedTaxibot
            2_ObjectDetection
                1_Preprocessing
                2_Tensorflow
                    models
                    protoc
                    workspace
                        annotations
                        images
                        models
                        pre_trained_models
                3_Output
```

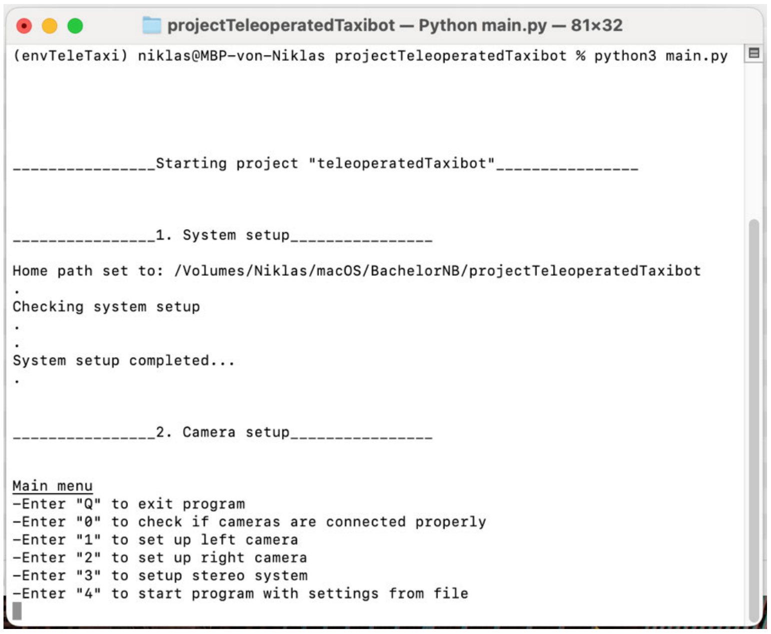
Figure 27: Path to model training environment

Next, the *label map*, which allows a model to identify the class of the detected object, given its integer object ID, is created. Furthermore, the image data from the images used for testing and for training as well as the information from the previously created XML files is converted into the TFRecord format, which is required by Tensorflow and used to start the training. The conversion is handled by a script available in [42].

The downloaded, pre-trained model includes a checkpoint folder as well as a pipeline.config file. The former captures the current state of the model, by storing the weights of the neural network. This checkpoint is used as a starting point to perform the fine-tuning of the model, i.e., adding weights to perform object detection and classification. The pipeline is used to configure the new training job. Parameters, such as the number of classes the model shall be trained on, the paths to checkpoint, TFRecord and the labelmap must be added to the pipeline. Furthermore, the batch size, i.e., the number of training jobs performed in parallel, must be set. A batch size of 4 is selected for this project is selected [41] since the computation is performed on a computer with limited resources. In the last step, the routine that starts the training job, called *model_main_tf2.py* is started, setting the number of training steps to 10.000.

4.2.3 Depth computation

The routine *main.py* guides the user through the system and camera setup process by using the shell only. Ordinarily, once the installation is completed, the program shall be started with the shell command "*python3 main.py*". This will setup the previously mentioned directory structure and display a menu in the shell, which we will go through on the following pages.



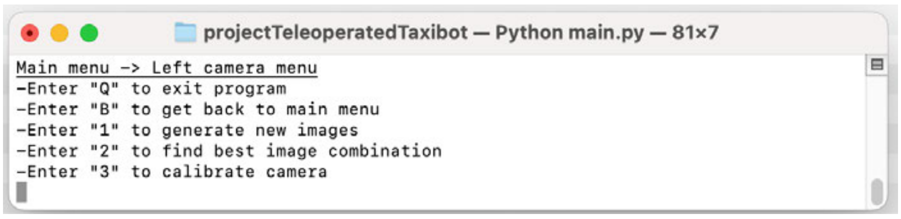
```
projectTeleoperatedTaxibot — Python main.py — 81×32
(envTeleTaxi) niklas@MBP-von-Niklas projectTeleoperatedTaxibot % python3 main.py



________________Starting project "teleoperatedTaxibot"________________


________________1. System setup________________

Home path set to: /Volumes/Niklas/macOS/BachelorNB/projectTeleoperatedTaxibot
.
Checking system setup
.
.
System setup completed...
.


________________2. Camera setup________________


Main menu
-Enter "Q" to exit program
-Enter "0" to check if cameras are connected properly
-Enter "1" to set up left camera
-Enter "2" to set up right camera
-Enter "3" to setup stereo system
-Enter "4" to start program with settings from file
```

Figure 28: Main camera setup menu

Before starting further computations, one should select option "0", which will open the livestream of both cameras in which it should be checked whether the window "Camera R" and "Camera L" actually display the correct cameras. If they do not, the simplest solution is to re-connect the cameras to the opposite USB port, respectively.

4.2.3.1 Single camera calibration

First, the left camera shall be set up by typing “1” in the main menu. This will open the left camera menu providing three set up options displayed in Figure 29. This section will describe the process for the left camera but the same steps apply for the setup of the right camera too.



```
Main menu -> Left camera menu
-Enter "Q" to exit program
-Enter "B" to get back to main menu
-Enter "1" to generate new images
-Enter "2" to find best image combination
-Enter "3" to calibrate camera
```

Figure 29: Single camera setup menu

When using a new camera, one must compute the camera intrinsic and distortion coefficients first. Before explaining the functions that compute these parameters, one should first discuss how a successful calibration can be identified.

Reprojection error

It was described at the beginning of this paper that a homography matrix maps a point in world coordinates to a point in image coordinates and that, if this procedure is completed multiple times, the homography matrices provide a system of equations that can be solved for the camera intrinsics as well as the translation and rotation values.

Looking at the procedure in reverse, using the computed camera intrinsics, extrinsics and distortion coefficients, a homography matrix can be computed for a given view of a chessboard. The homography matrix computed in this manner may well differ from the matrix computed in the original procedure because the camera intrinsics are refined with every additional chessboard image for which the homography is computed. Using the homography matrix from the inverse procedure, and using the set of known object points, the projected image coordinates can be computed. These coordinates may vary from the *actual* images coordinates that were previously recorded when taken the image of the chessboard. The offset in x- and y- coordinates is the *reprojection error*. It provides an insight of how accurate the computation of camera intrinsics and distortion coefficients is. The goal for a calibration is, thus, to minimize the reprojection error.

For the comparison with Matlab's Camera Calibrator the following terminology shall be used: While for a *single point* the reprojection offset is given in x- and- coordinates, the root mean square (RMS) of the two will be called *reprojection error*. The mean of all reprojection errors for one chessboard view is called *mean reprojection error.* Furthermore, the *overall mean reprojection error* is the mean value of all mean reprojection errors for all images.

Generating images

Back to the left camera menu, the first step is to take multiple images of a chessboard. Therefore, option "1" is selected, which runs a routine that opens a livestream of the selected camera. Now, a chessboard must be placed in front of the camera. For this paper, a chessboard with n=9 rows and m=6 columns ("9x6") is used. Multiple photos of the chessboard are taken. The aim is to have photos of the chessboard from different perspectives, with different orientations and with varying distances to the chessboard. The board must be in focus, all squares must be visible on screen and there should be a thick white border outside of the chessboard area, which must be visible on screen as well [48]. Furthermore, areas of high brightness due to high light reflection, such as from a lamp or from the sun shall be avoided. The literature that is quoted by the OpenCV documentation [15] recommends using at least 10 images with a large difference in pose and position of the chessboard. Posts by official staff of OpenCV's online forum add, based on experience, that the chessboard shall not be photographed when parallel to the camera [49] and that the total mean reprojection error shall be below 0,5 pixels to ensure an accurate depth computation. Figure 30 and Figure 31 display the orientation, pose of and distance to the chessboard in the images generated for this paper.

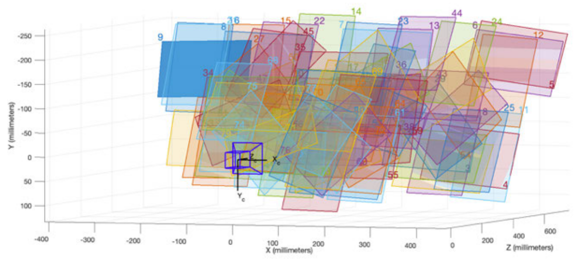


Figure 30: Display of captures images from the back

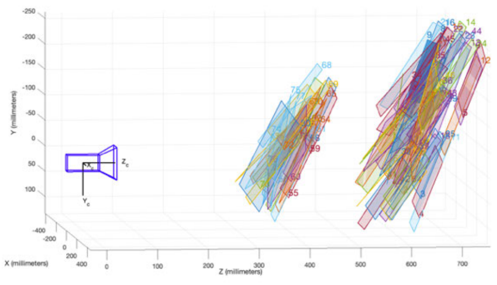
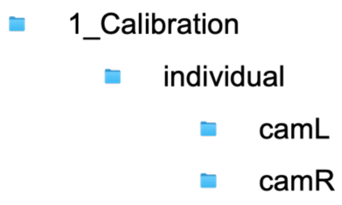


Figure 31: Display of captures images from the side

The routine takes a photo of a chessboard and uses OpenCV's *findChessboardCorners()* to find all chessboard corners. If all corners are found, the locations of all corners are refined, and coloured points are drawn onto the frame where the position of chessboard corners was computed. The image is then saved into the “individual calibration” folder of the respective camera, where it will be analysed in the next step (Figure 32).



- 1_Calibration
  - individual
    - camL
    - camR

Figure 32: Storage location for single cam calibration images

Finding best combination

The second option “2” in the left camera menu is a routine that finds the best combination of images of the ones that were taken in the last step. The routine starts by reading all images and storing the chessboard’s corner locations as two-dimensional image coordinates (in pixels) in a matrix called “imagePoints”. The 3D world coordinates are stored in a matrix called “objectPoints”, with the third coordinate set to $Z = 0$ (because one coordinate was chosen to be zero, see page 11). This matrix has an important role in the overall depth computation.

$$objectPoints = \begin{bmatrix} (x_1\ y_1\ 0), \\ (x_2\ y_1\ 0), \\ (x_3\ y_1\ 0), \\ \dots \\ (x_{n-1}\ y_{m-1}\ 0) \end{bmatrix} = \begin{bmatrix} (0\ 0\ 0), \\ (1\ 0\ 0), \\ (2\ 0\ 0), \\ \dots \\ (8\ 5\ 0) \end{bmatrix}$$

In the above example, the object points are stored in normalized coordinates, i.e., a chessboard square has a length of 1 unit. While using this approach does not influence the resulting camera intrinsics and distortion coefficients, it does influence the translation vector that is the result of the stereo calibration procedure and defines the units in which the distance computation will later be returned. The matrix “objectPoints” is, therefore, multiplied by the manually measured length of a square, which is 2,2cm for this paper but must be changed by the user when starting a new calibration procedure accordingly. Note that units are not stored in the matrix itself and one must remember which units were defined for the lengths in this matrix.

$$objectPoints = objectPoints * squareSize = \begin{bmatrix} (0\ 0\ 0), \\ (2{,}2\ 0\ 0), \\ (4{,}4\ 0\ 0), \\ \dots \\ (17{,}6\ 11{,}0\ 0) \end{bmatrix}$$

OpenCV's routine *calibrateCamera()* is then used to calibrate the camera using the object points, image points and some further parameters as input. Particularly important parameters to consider are the input-distortion matrix as well as the calibration criteria. The distortion matrix (n=5, m=1) has to be pre-defined, because if it is not OpenCV computes a (n=1, m=5) matrix by default, which causes issues in later functions. The calibration criteria specify the accuracy of the computes results, the camera intrinsics matrix and the distortion coefficients. The term *"CV TERMCRIT EPS"* defines to which decimal value (1/10, 1/100, 1/1000) the computed values, e.g., intrinsics and distortion coefficients, shall be calculated before the results are accepted and the algorithm stops. Criterion *"CV CRITERIA MAX ITER",* on the other hand, specifies after which iteration the algorithm should stop refining the results, if the target accuracy is not reached [15]. Following multiple test runs, the settings $Term\_Criteria\_EPS\ = 1 * 10^{-6}$ and $Term\_Criteria\_Max\_Iter\ = 100$ produced the most accurate calibration results.

The routine computes the homography for every chessboard view and solves for the camera matrix. In a second iteration, the distortion coefficients are calculated (without derivation [15]) using the now known intrinsics parameters as a starting point to narrow down the actual position of a point in the undistorted frame. Because the intrinsics are calculated for images that assume a non-existing distortion, both matrices, distortion coefficients and intrinsics, are passed into *getOptimalNewCameraMatrix()*, which returns the distortion-considering-intrinsics as the *new camera matrix.*

Calibrating a camera using all images that were generated in the previous step can lead to large *overall mean reprojection error* that can be caused by single images, which, for one of the reasons listed before, have a large *mean reprojection error* and thus increase the overall mean. To optimise the image combination for the lowest *overall mean reprojection error*, the following approach is used:

First, the 5 images resulting in the lowest *overall mean reprojection error* are selected. Since the camera calibration function may fail for fewer images, this is achieved by choosing a *base* of 5 consecutive images and adding 5 images, one by one, while computing a new *overall mean* *reprojection error*. This means that a complete camera calibration is performed for every image combination to find the overall mean reprojection error. In theory, if an image is added that only increases the *overall mean reprojection error* insignificantly, this image should have a low *mean reprojection error*. After 5 images are checked, these last 5 images are used as the new base and one by one, all images that are left are added and the *overall mean reprojection error* for the new image combination is computed. If an image increases the *overall mean reprojection error* beyond a certain threshold, it is removed, and a different image is tested instead. The results are displayed in Figure 33.

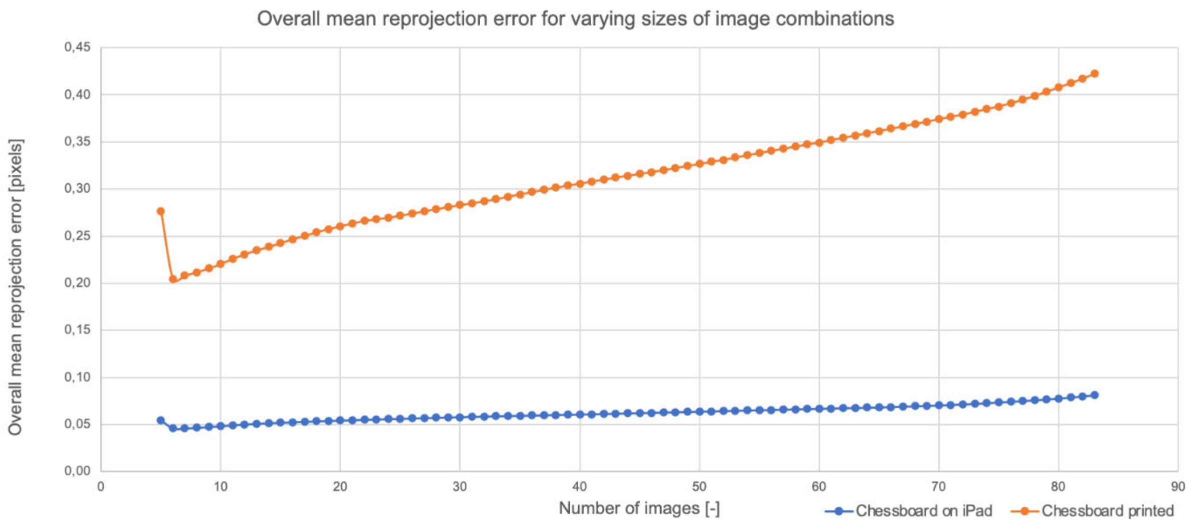


Figure 33: Comparison of the reprojection error for various sizes of image combinations

From Figure 33 one can derive that compared to a combination of 5 images that were shown to have the lowest mean reprojection errors, increasing the number of images yields better results, i.e., lower total mean reprojection errors. However, this is only true for up to circa 25 images. The global minimum is achieved for 6 images with an *overall mean reprojection error of 0,046 pixels*. The right end of the graph shows a steep increase in the *overall mean reprojection error*. This might be caused by the images that were used for the computation of the error. These are the ones that produced large errors when they were checked for smaller image groups and were thus removed from those groups.

The specific combination of images that lead to the lowest *overall mean reprojection error* in OpenCV is analysed using Matlab's Camera Calibrator. Compared to OpenCV, Matlab produces an overall mean reprojection error of 0,12 pixels, which is nearly 2,5 times larger (worse) than OpenCV's result.

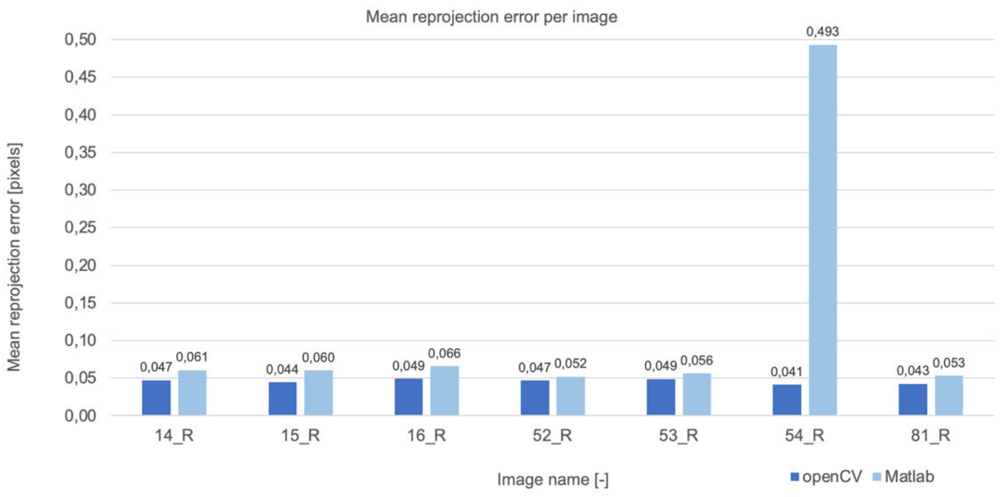


Figure 34: Comparison of reprojection errors in Matlab and OpenCV

Figure 34 displays the mean reprojection error for each image in the best image combination. Matlab's mean reprojection errors are only slightly larger than OpenCV's values. However, for image 54_R there is a significant difference in the errors.

Throughout the many (failed) calibration attempts that were made during the project work on this paper, another significant factor became clear, which is displayed in Figure 33. Compared to the overall mean reprojection errors that result from using the chessboard display on an iPad, the printed version of a chessboard yields larger errors and thus implies a less accurate calibration. The best image combination of 6 images results in an overall reprojection error of 0,204 pixels. It is the case even though very similar lighting conditions were used while generating both image series as well as using similar orientations, poses and distances to the camera. Possible explanations are the resolution of both mediums, i.e., the printing resolution displaced chessboard corners compared to the higher resolution on the iPad. Furthermore, while chessboard corners on the iPad are precisely equidistant, attaching the printed version of the chessboard to a book and placing it in various angles on object on the floor might introduce a slight bend, causing the distances between the corners to be not as precisely equidistant as the iPad's ones. As a result, it is derived that the quality of calibration therefore relies on displaying the chessboard in a high quality. Hence, the iPad will be used for calibration throughout this paper.

The names of the images of the best image combination are stored in the calibration folder.

Calibration

In the last step, the image combination stored in the previous step is loaded and, using OpenCV's *calibrateCamera()* the camera intrinsics and distortion coefficients are computed (once again). This is made to be a separate function to provide a quick insight into the best calibration results (errors, image numbers, intrinsics and distortion coefficients), as the "Combination testing" function takes a lot of time, circa 15 minutes on a dual core CPU, to compute the results.

Validation

Starting the program by calling "*python3 main.py --debugging True*" and selection option 4 in the main menu, computes a validation for each step in the camera setup. For the single camera calibration such a plausibility check is shown in Figure 35 and Figure 36, which display the distorted and undistorted image of a chessboard. The function used to undistort the images, OpenCV's *undistort(),* uses the cameraMatrix, newCameraMatrix and the distortionCoefficients as inputs and thus these three results can be checked for plausibility by looking at the images. Noticeable are for an undistorted image are the black corners, where pixels were remapped to remove the distortion effect.

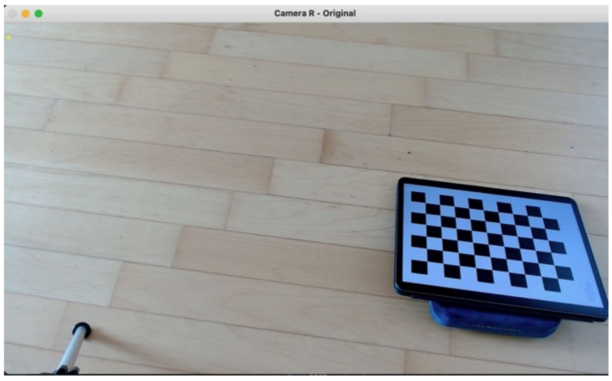


Figure 35: Original image frame

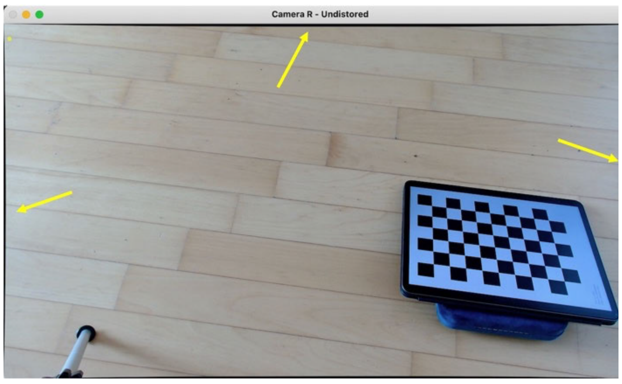
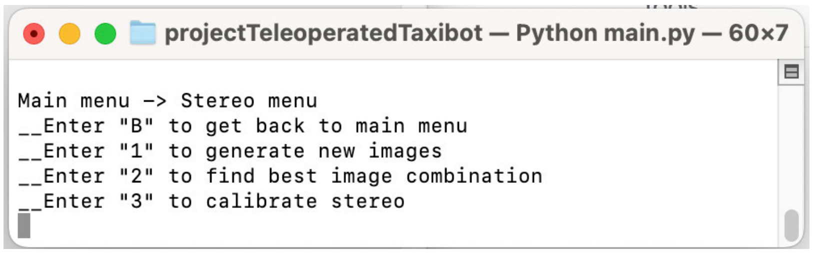


Figure 36: Undistorted image frame

4.2.3.2 Stereo calibration

After the single camera calibration has been completed, the next step is the stereo calibration procedure, which is very similar to the calibration of the individual cameras and has the same user options as the single camera calibration.



projectTeleoperatedTaxibot — Python main.py — 60×7

```
Main menu -> Stereo menu
__Enter "B" to get back to main menu
__Enter "1" to generate new images
__Enter "2" to find best image combination
__Enter "3" to calibrate stereo
```

Figure 37: Display of options for the stereo camera setup

Generating images

First, photos of the chessboard in different locations and poses must be taken using the stereo setup, in a manner that the full chessboard is visible in both frames at once. Images for which the chessboard pattern is identified are stored in the “stereo calibration” folder.

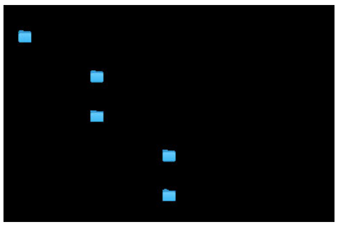
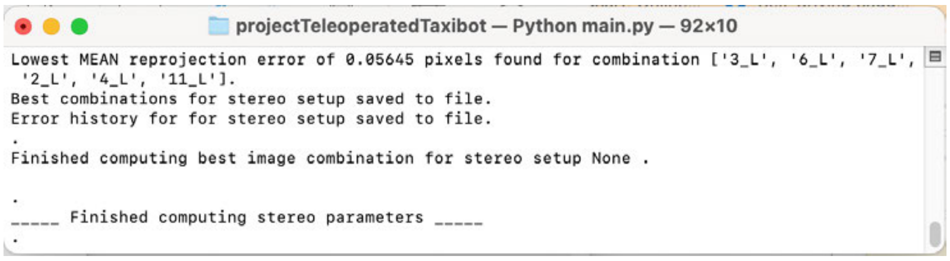


Figure 38: Directory of generated stereo images

While it is possible, in theory, to compute the camera intrinsics using the images generated during the stereo calibration process, it is not recommended. Especially the distortion coefficients, which are parameters that are returned by the single camera calibration, are more significant at the edges of the cameras’ frames than in their centre. When taking photos using two cameras, it is geometrically not possible, as both cameras are horizontally offset, to generate images that have a chessboard both, in the left camera’s left edge and in the right camera’s right edge. Hence, both steps are completed separately.

Finding best combination

After a sufficient number of images is generated, the next step is to, once again, find the combination of images leading to the lowest reprojection error. This is accomplished using the same function that was used in the single camera calibration. It is important to note the return value of the program for the overall mean reprojection error:


projectTeleoperatedTaxibot — Python main.py — 92×10

```
Lowest MEAN reprojection error of 0.05645 pixels found for combination ['3_L', '6_L', '7_L',
 '2_L', '4_L', '11_L'].
Best combinations for stereo setup saved to file.
Error history for for stereo setup saved to file.
.
Finished computing best image combination for stereo setup None .

.
_____ Finished computing stereo parameters _____
.
```

Figure 39: Stereo overall mean reprojection error

These values must only be seen as a comparison metric relative to the results of other stereo image combinations in terms of which images have the best quality for stereo calibration. The results are not the actual overall mean reprojection errors for the stereo calibration procedure, as they do not include the previously computed intrinsics and distortion coefficients.

Calibration

After the image combination with the lowest overall reprojection error has been computed, the stereo setup is calibrated using OpenCV's *stereoCalibrate().* The relevant outputs for this paper, described in chapter 2.3 , are the translation matrix (T) and the rotation matrix (R).

Validation

One can use the first value of matrix T, which is the baseline b, to get a very rough indication whether the stereo calibration procedure was successful. For the stereo setup used in this paper, the translation is $T = [-9{,}62,\ -0{,}11,\ -0{,}19]^T\ in\ cm.$

It is important to keep in mind that these matrices are used to transform a point in the right camera's coordinate system into the one of the left camera. Hence, the values in matrix T are negative, because, for the horizontal element, the left camera is located further towards the coordinate system's origin than the right camera, according to the coordinate system definition on page 5.

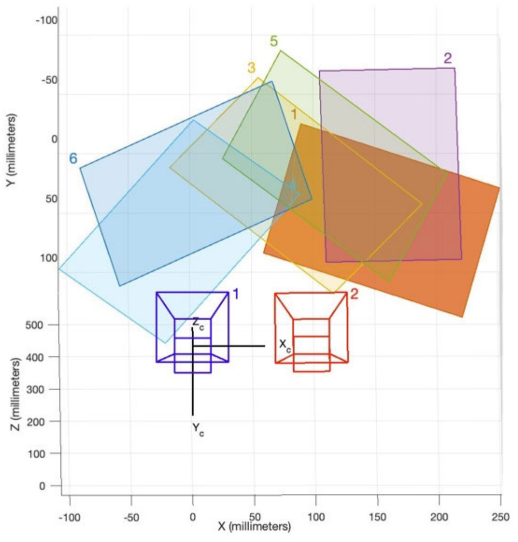


Figure 40: Horizontal offset for stereo setup

Matlab's Camera Calibrator provides a visual display of the result, which shows the location of both cameras relative to the image pair (which is displayed as a single image) of the chessboard. Doing so the offset between both cameras becomes evident, displaying a horizontal displacement of circa 100mm in Figure 40, which offers a rough plausibility check for the success of the calibration.

While the intrinsics and distortion coefficients do not change for a camera, the rotation and translation matrices for a stereo setup may change for instance if the object they are mounted to changes its shape. One reason for this could be that the object is made from a metal, which elongates and shortens depending on the environment's temperature or fatigue in a material that bends the object over time. Moreover, if two single cameras are used to enable a stereo-setup and if these cameras are rotated individually by an electric mechanism to provide a different field of view, these matrices change as well depending on the accuracy of the movement mechanism. For reference, such camera rotation might be of interest for a camera pair at the front of the Taxibot, which performs frontal depth detection during the movement of the airplane but is then rotated towards the aircraft's front landing gear to provide the operator with a better view on the uplifting system during coupling and decoupling.

4.2.3.3 Rectification

The rectification function, *cv.stereoRectify(),* uses both cameras' intrinsics and distortion coefficients as well as the rotation and translation matrices R and T between the two cameras to compute two rotation matrices R_L and R_R, two projection matrices P_L and P_R as well as the reprojection matrix Q. While the rectified camera matrices are the first three columns in the projection matrices, the **re**-projection matrix can be used to reproject all points in a disparity map into a point cloud, consisting of three dimensional coordinates in the left cameras' coordinate system.

One important parameter, which must be activated in the rectification function is "*cv.CALIB_ZERO_DISPARITY*". What it does is that it generates these transformation matrices in a manner that both principal points ($c_{x,l}$, $c_{x,r}$) have the same values, which is the fundamental assumption that was made to derive the depth equation on page 16.

OpenCV's *initUndistortRectifyMap()* uses the results from *stereoRectify()* to compute two rectification maps per camera. These maps, one for the horizontal and one for the vertical direction, transform all points in a newly captured image pair to be coplanar and row aligned.

Validation

After the rectification procedure has been completed, the success of the stereo calibration and rectification steps should be verified. This is done by using a chessboard, not an existing image of a chessboard that was taken previously, since the rectification is based on this image, but by holding a chessboard, once again, in a manner that both cameras' live streams display the pattern. Once the chessboard pattern is found and drawn onto the two frames, the user shall press the *SPACE* key. This starts a routine, which displays the un-rectified (therefore original) versions of both frames, as well as the rectified image pair. In addition, the algorithm connects a chessboard corners in the image pair using an orange line and displays the horizontal pixel coordinates for both images.

Given that the epipolar lines in both rectified images are horizontal and row-aligned, a line that goes through a chessboard corner in the left image, must do the same for the corner in the right image. Horizontal lines are thus also drawn on both images. The full results are displayed in Figure 60 and Figure 61 in the appendix, while Figure 41 and Figure 42 display a detail view of a region in the right frames. In the Figure 41, the blue horizontal lines, which have the vertical coordinate of the corners in the left image pair, do not go through the corresponding corner in the right image. The yellow line connecting both corresponding corners are clearly offset from the blue horizontal lines. Such a display is common for either an un-rectified frame or a badly rectified frame and can be used to properly validate the results that were computed so far.

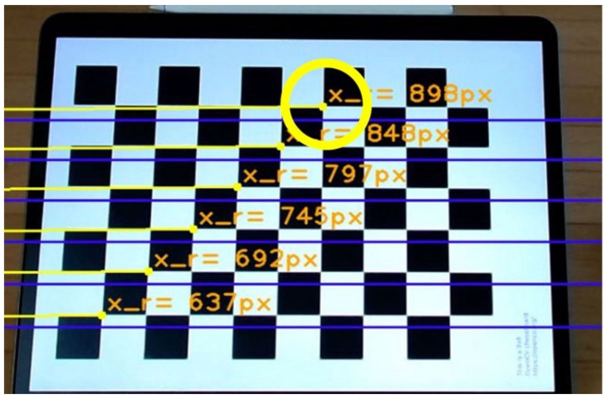


Figure 41: Detail view of un-rectified frame

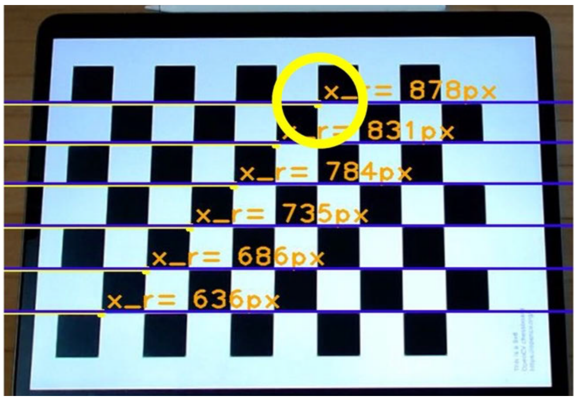


Figure 42: Detailed view of rectified frame

In Figure 42, on the other hand, the blue horizontal lines go through the corresponding corners in the right image. Furthermore, the yellow line is now almost colinear with the blue line, proving that the image is correctly rectified.

4.2.3.4 Correspondence

Finding a feature from the left image in the right image is called *correspondence problem*. In general, the search for corresponding points requires a lot of computational resources and time. While their search has been greatly reduced by rectifying the images pair, some further important parameters, allowing the computational time to be reduced, shall be explained.

Horopter

Generally speaking, *correspondence algorithms* extract the information of a single pixel or a region of pixels from the left image, which is called *window* or *block*, to find the corresponding pixel or window in the right image. For a given coordinate (u_l, v_l) in the left image, the search for a match in the right image usually begins at the same coordinate (u_l, v_l). An object at infinity, or at a large distance, would have a disparity close to zero and, thus, the same image coordinates in both frames. The parameter *minimum disparity*, which is zero by default, can be set to larger values, if the detection range for objects can be limited i.e., only objects closer to the camera are of interest. The parameter *number of disparities*, which specifies the range in pixels measured from the coordinate (u_l) in the right frame, can be used to limit the search area. While the depth to far objects can still be computed, setting the *number of disparities* to a low value will limit the range at which minimum depth can be detected. These geometric dependencies are shown in Figure 43, which displays the 3D-volume, a *horopter*, in which depth can be detected.

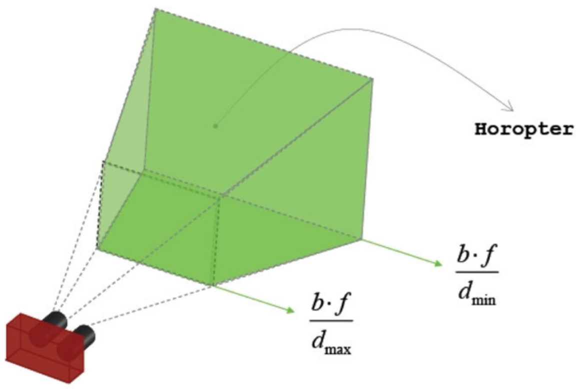


Figure 43: Detectable depth range - "horopter" [50]

Alternatively, the *maximum disparity* can be set to limit the search range for a feature with the *number of disparities* being the difference between the *maximum* and *minimum disparity*.

Correspondence algorithms and their applications have been studied in many papers such as [51] and [52]. This chapter will outline the key differences between OpenCV's *StereoBM* and *StereoSGBM* algorithms and study their implementation for the previously described geometry. The goal of this chapter is to compute a disparity map, storing the disparity information of all points in the image pair.

StereoBM

OpenCV's *Block Matching* (BM) algorithm extracts the intensity value for the pixels in a window from the left image and determines the same properties for a window of the same size in the right image. The resulting sum of absolute differences (SAD), indicating the similarity between the feature in the left and right window, is calculated for all disparity levels, defined by the parameter *numDisparities.* The best match is determined using the "winner takes all" principle, with the overall lowest computed value selected as the correct match. This algorithm works best on feature-rich images because correct matches rely on varying pixel and colour intensities to differentiate between, while low-textured regions can produce inaccurate results. Furthermore, OpenCV's StereoBM cannot be used to find matches for individual pixels.

The routine takes two inputs, *numDisparities* and *block size*. The latter defines the window, that is extracted from the left image and searched for in the right image. Choosing a small block size provides a greater amount of detail, displaying clearer edges and shapes (see Figure 44). It also leads to more noise, because a smaller window contains fewer unique features and thus the matching function might assume incorrect windows to be the correct match. To paint an example, a small window with dark pixels could belong to a car tire but be matched to a pair of sunglasses. Larger block sizes produce fewer details in smoother disparity maps in which the boundaries between foreground and background may not be displayed correctly. It is important to set the minimum disparity using *stereo.setMinDisparity()* after initialising the algorithm, which does not allow the parameter to be entered into the first initialisation call.

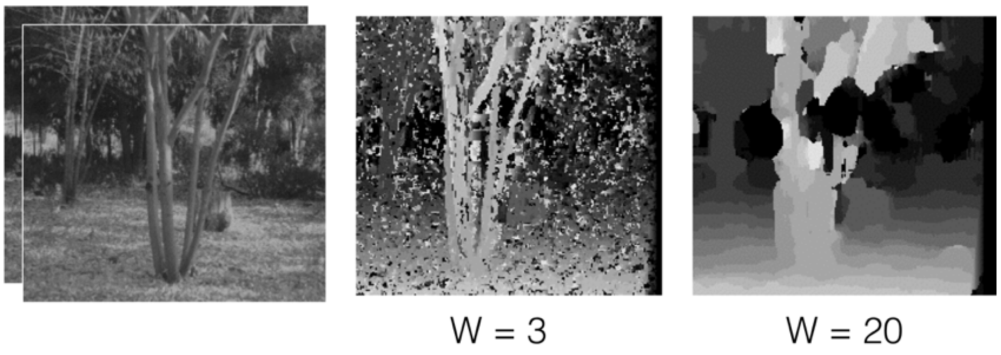


Figure 44: Example of disparity maps using different block sizes

Disparity maps, like the one in Figure 44, are typically encoded using intensities, where points or objects close to the camera are brighter than objects at greater distances, which are darker [50]. They are, furthermore, usually written in signed values.

StereoSGBM

The *Semi Global Block Matching* (SGBM) algorithm can be used with single pixels or windows. For every pixel or centre pixel in a window, it determines the "centre symmetric census transform" (CSCT), a 31-bit value that includes colour and intensities of surrounding pixels even if only searching for a single pixel match. The matching cost is calculated by comparing the CSCT for the original feature and the current window in the right image. Furthermore, smoothness of disparity changes for neighbouring pixels is considered in the overall matching cost, which aims to minimize sudden, falsely detected depth changes. Due to the complexity of this correspondence search, StereoSGBM requires more computation time to generate a disparity map, compared to the StereoBM algorithm. The results are said to be more accurate, however, which will be tested in the last chapter.

Both, StereoBM and StereoSGBM allow for more parameters to be input. These can be used to specify uniqueness thresholds that matches must differentiate themselves from the second-best potential match to be chosen as the winner, or parameters adding additional smoothness constraints. During this project work, however, only the parameters *blockSize*, *minDisparity* and *maxDisparity* (or *numDisparities)* shall be used to compare the most basic matching capabilities of both algorithms.

WLS-Filter

Since computational speed, as well as accuracy are both required for safety critical real-time applications, such as a Taxibot pushing an aircraft filled with passengers, an alternative to the two previously mentioned algorithms should be analysed. With StereoBM computing results quickly, but with lower accuracy, and StereoSGBM providing the opposite, a middle ground can be of interest. Such is provided by the "*Weighted Least Squares*"-Filter [53]. In a disparity map based on the *left* image, each element would contain one pixel value, e.g., $disp_l = u_l - u_r = 8$, implying the displacement of the corresponding point in the right image is at the horizontal coordinate $u_r = u_l - 8[pixels]$. Computing another disparity map based on the *right* image would in this case lead to the element having a value of $disp_r = u_r - u_l = -8$, with the horizontal coordinates of the point in the left image being at $u_l = u_r + 8[pixels]$ [54]. The WLS Filter computes two disparity maps, a left disparity map and a right disparity map, and filters inconsistent disparity values. The computed results shall be analysed for the use with StereoBM and compared to the accuracy of standalone StereoBM and standalone StereoSGBM.

Depth

Once the disparity map is computed, the next step is the depth computation, for which there are multiple approaches. The first approach, using equation (24), is to find a point of interest and its image coordinates in the left rectified image using object detection. This is achieved by modifying the visualisation functions of the object detection algorithm to detect and return the image coordinates as well as the class name of the two bounding boxes with the highest confidence ratings, which can then be processed in the script *main.py.* Then, the depth to this single point is determined using the known rectified camera matrix and the baseline.

A second approach, using OpenCV's reprojection matrix Q, reprojects all points in the image into a 3D point cloud, providing depth information about all points in the image. While it is the assumption that the first approach takes less time to compute, since only the information for a single point is computed, knowing the depth information for all points in the image is advantageous. For the first approach, an object detection algorithm must detect any object for which depth shall be known. To determine the pose and position of an aircraft, using its engines, it is sufficient. The second approach, however, makes the depth detection more robust allowing for unknown obstacles to be detected within the search range. Both approaches are tested in the next chapter.

Taxibot

The above-mentioned methods for reducing the computational time of the correspondence search shall be applied to the prototype stereo setup used in this paper. A minimum detectable depth of $z_{min} = 35cm$ and a maximum depth of $z_{max} = 70cm$ shall be possible. Using equation (24), the baseline $(b = -t_x)$ and the rectified focal length $(f_{x,rect})$ the minimum disparity (d_{max}) is calculated as:

$$d_{max} = \frac{1323{,}3 pixels * 9{,}62cm}{70cm} = 181{,}86\, pixels$$

Both, the value for minimum disparity and maximum disparity must be dividable by 16 for the algorithm to work. Thus, the result is rounded up to the next number dividable by 16 $d_{max} = 192\, pixels$. The minimum disparity is calculated analogously to be $d_{min} = 368\, pixels$.

5 Testing

Static tests

Static tests were completed for an image pair displaying two aircraft engines for distances of 35 to 70cm in 5cm increments. For each image pair, computation time, disparity and depth were determined using StereoBM, StereoSGBM and StereoBM with WLS filter. In addition, object detection was applied to both images to determine the disparity of the identified engine coordinates. All of these values were compared to the ground truth, which was measured using a measuring tape by hand, thus introducing slight inaccuracies. Figure 45 displays the original frames of one such measurement. Two printed images of two different engines were attached to a solid object to provide an insight into possible differences in detection and depth estimation between these engines.

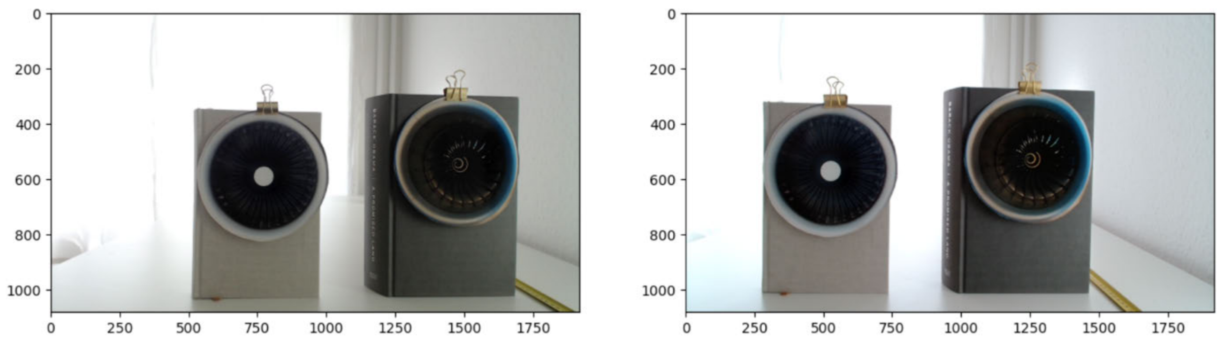


Figure 45: Sample of the original image pair used for static tests

Beginning with the localisation of the engine centre(s), Figure 46 and Figure 47 show two rectified images, for which the engine centres have been identified and the engines classified as such, by the object detection model.

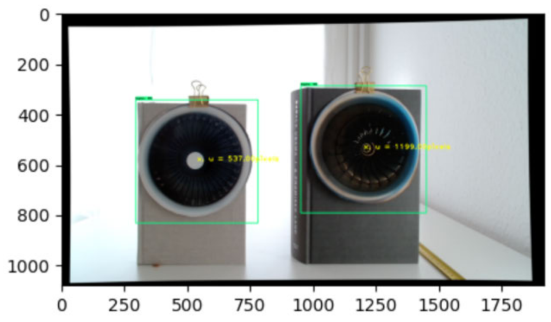


Figure 46: Sample of the left camera frame, including detection

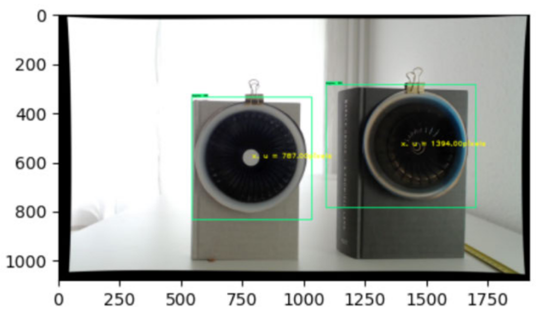


Figure 47: Sample of the right camera frame, including detection

What became apparent during this test series, is that a static definition of correspondence algorithm parameters does not produce accurate results in practice. Figure 48 and Figure 49 display the disparity map computed by OpenCV's block matching algorithm. The image on the left was computed using a block size of 5 pixels. The depth was estimated to be 55cm, whereas the actual distance was 35cm. Because the disparity and depth are only determined for a single point (the size of 1 pixel), which is returned by the object detection function, in the case of Figure 48 the computation error could be traced back to the point of interest being in a noisy area, where correspondences could not be identified by the algorithm. While, in theory, increasing the block size should reduce noise, quadrupling the block size to 21, displayed in Figure 49 did little to improve the result. The computed distance did not change.

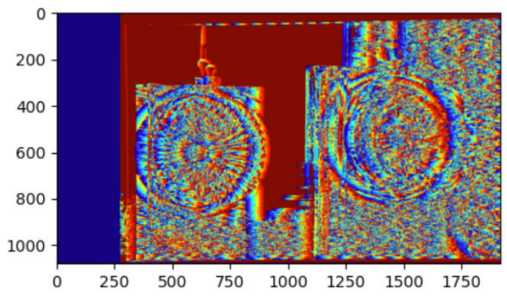


Figure 48: StereoBM, blockSize = 5px, groundTruth=35cm, meanDistance=55cm

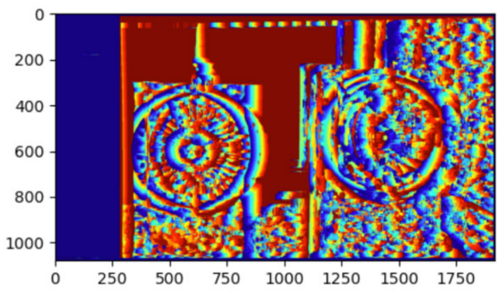


Figure 49: StereoBM, blockSize=21px, groundTruth=35cm, meanDistance =55cm

In other cases, pictured in Figure 50 and Figure 51, a block size of 5 pixels computed a mean depth, i.e., the mean computed depth for both engines, of 54,52 cm in the left image and 50,90 cm in the right image. However, comparing the two engines in the same image, the specific parameters in Figure 50 compute a depth of 61,49cm for the left engine and 47,54cm for the right engine. The ground truth depth in this example being 45cm, this results in an error of circa 5%, which is considerably lower than for the left engine where it is more than 20%. A possible explanation for this particular computation is that the right engine (see Figure 47) has a yellow feature on the fan, compared to the plain white surface of the left fan. Such a feature might increase the accuracy of correspondence calculations. Another interesting, but unexpected phenomenon is that for the right-hand engines in Figure 50 and Figure 51, depths of 47,53cm and 47,54cm were computed, although stemming from two different stereo algorithms, StereoBM and StereoSGBM, the latter of which requires considerably more computational time for the calculation.

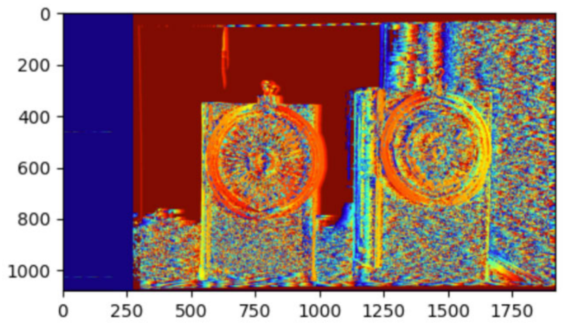


Figure 50: StereoBM, blockSize=5px, groundTruth=45cm, meanDistance=54,52cm

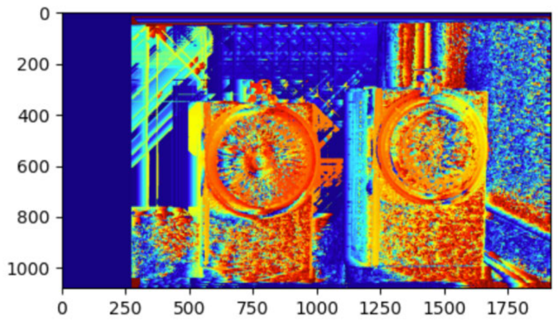


Figure 51: StereoSGBM disparity map, blockSize=5px, groundTruth=45cm, meanDistance=50,90cm

The statistic displayed using the diagram in Figure 52 visualises as the overall result that the StereoSGBM algorithm in combination with the WLS filter produced the most accurate outputs in tests, albeit at a considerably higher computational time than any other algorithm (see Figure 53). Relative to the ground truth, which for the entire test series was between 35cm and 70cm, the resulting deviation, on average, is up to 10%.

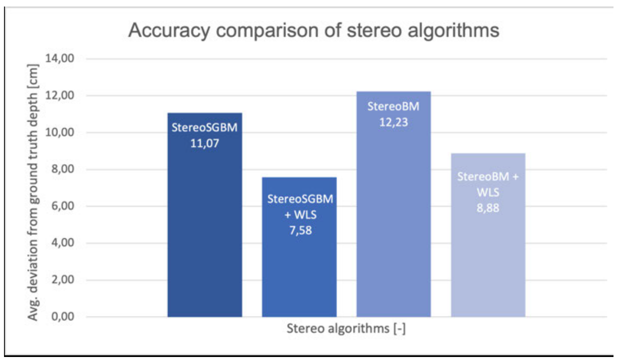


Figure 52: Accuracy comparison for stereo algorithms

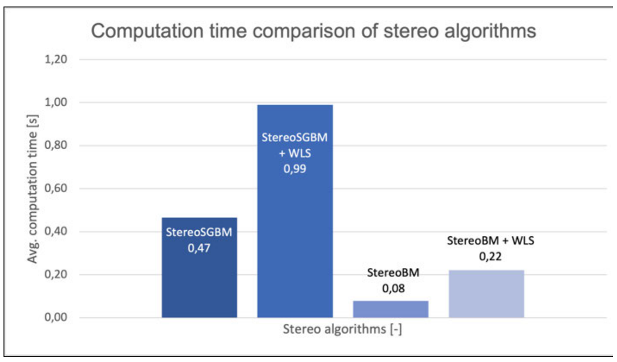


Figure 53: Computation time comparison for stereo algorithms

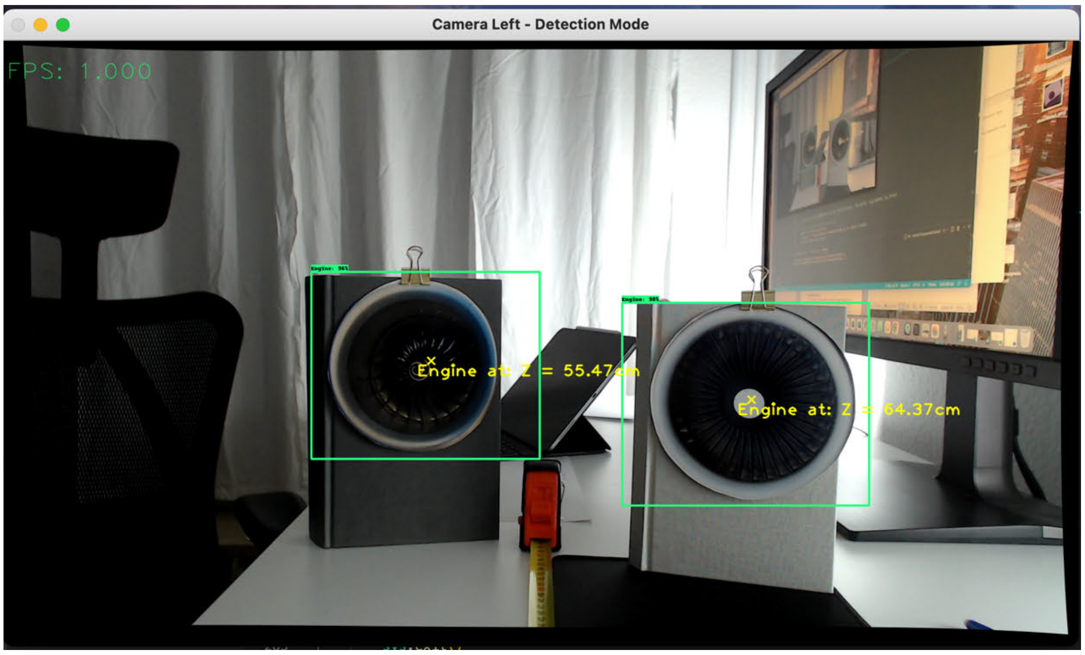
Due to the time limitation set for this paper, dynamic tests could not be completed.

6 Conclusion and outlook

This paper studied the implementation of a machine learning based vision aid for the implementation on a teleoperated Taxibot concept. The focus was placed on the software prototype, which had the aim of assisting the operator navigating the vehicle from a remote location, by detecting two engines of an Airbus a320, in order to determine the position and pose of the engines, and thus the aircraft, relative to the coupled Taxibot.

A considerable amount of time was invested in developing a framework, based on python and the open-source computer vision package, which navigates a future user through the steps of image generation, single camera calibration, stereo calibration, rectification, correspondence search as well as triangulation. Furthermore, existing object-detection model training modules were adapted to optimise a new transfer learning process as well as to detect and evaluate the distance to two aircraft engines in a livestream, using a simple stereo system.

The result of the project work is a fully and correctly configured stereo vision setup, which detects the depth to objects in a range between 35cm and 70cm with an average error below 10%.



The design concept that was followed was to provide a remote operator with a rectified camera view on the aircraft, mark obstacles (and engines) in a specific range and provide a bird-eye perspective view displaying the Taxibot as well as the surrounding geometry including obstacles and distances to objects. Such views are provided in many passenger vehicles nowadays. The next aim was to utilise related research concerning lane assist software and adapt it to detect aerodrome paths, which combined with the location and pose of the aircraft (engines) could be used to provide a visual display to the operator about the predicted and ideal angles of movement the aircraft would perform, depending on the magnitude of change that would be applied to the steering wheel of the Taxibot. Such technology already exists in passenger vehicles, displaying the optimal path to drivers to park the car in a specific position.

Before this next step can be approached, however, an in-depth study and automation of the parameters used for correspondence search must be performed. The static tests in this paper showed that engines prove to be difficult objects to find corresponding matches for, given their repetitive texture. Furthermore, illumination issues, such as shade coming into the engine depending on the time of day proved to be a difficult problem to solve. Given the considerable costs that were estimated a camera setup to be in case of the Taxibot, the usage of a 500 USD Velodyne Lidar system might be the better choice to detect engines. Computer vision could have a supportive role, such as performing perspective transformation to allow a bird-eye view or classify objects or even text recognition to determine the location of the Taxibot at the airport by reading airport signs, which’s location on an online or offline airport map could then be found. Especially the visibility issues in darkness, which is a regular operating mode in many countries in winter months or at airports that allow late or early flights, could prevent a successful usage of computer vision.

Given that the created framework configures a new stereo system from scratch easily and quickly, further research on these topics has just become a lot easier.

7 Sources

[1] Elektroniknet Logo, “Schlüsseltechnik fürs automatisierte Fahren,” [Online]. Available: https://www.elektroniknet.de/automotive/schluesseltechnik-fuers-automatisierte-fahren.154797.html. [Accessed 17 09 2022].

[2] IEEE Spectrum, “Tesla Places Big Bet on Vision-Only Self-Driving,” [Online]. Available: https://spectrum.ieee.org/tesla-places-big-bet-vision-only-self-driving. [Accessed 17 09 2022].

[3] SG Trains, “Changi Airport Skytrain,” [Online]. Available: https://www.sgtrains.com/network-cga.html. [Accessed 17 09 2022].

[4] navya, “Self-driving baggage tractor,” [Online]. Available: https://navya.tech/en/usecases/autonomous-baggage-tractor/. [Accessed 17 09 2022].

[5] Hochschule für Angewandte Wissenschaften Hamburg, “Automotive Development in 1:x,” [Online]. Available: https://www.haw-hamburg.de/forschung/forschungsprojekte-detail/project/project/show/audex/. [Accessed 17 09 2022].

[6] P. P. Otaegui, Analizing Teleoperated and Automated Airport Handling Services to Implement an Adapted Concept for an Efficient Ground Movement in AUDEx, Hamburg: HAW Hamburg, 2022.

[7] OLYMPUS Live Science Solutions, “Introduction to the Reflection of Light,” [Online]. Available: https://www.olympus-lifescience.com/en/microscope-resource/primer/lightandcolor/reflectionintro/#:~:text=reflection%20of%20light.-,What%20is%20Reflection%20of%20Light%3F,waves%20away%20from%20the%20surface.. [Accessed 03 08 2022].

[8] Maggie's Science Connection, “Relfection, Absorption, Transmission,” [Online]. Available: https://maggiesscienceconnection.weebly.com/reflection-absorption--

transmission.html#:~:text=When%20light%20hits%20an%20object,absorb%20and%2For%20transmit%20light.. [Accessed 17 09 2022].

[9] Framos, “DEPTH SENSING TECHNOLOGIES OVERVIEW,” [Online]. Available: https://www.framos.com/en/products-solutions/3d-depth-sensing/depth-sensing-technologies. [Accessed 17 09 2022].

[10] D. Oxtoby, S. Fallah, M. Dianati, O. Y. Al-Jarrah, E. Arnold and A. Mouzakitis, “A Survey on 3D Object Detection Methods for Autonomous Driving Applications,” *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* vol. 20, no. 10, pp. 3782-3795, 2019.

[11] L. You and I.-G. Javier, “Lidar for Autonomous Driving,” *IEEE SIGNAL PROCESSING MAGAZINE ,* pp. 50-60, 2020.

[12] Forbes, “Velodyne’s $500 Velarray Solid-State Lidar Goes Into Production In 2021,” [Online]. Available: https://www.forbes.com/sites/samabuelsamid/2020/11/13/velodyne-announces-500-velarray-h800-lidar-production-in-2021/?sh=6e6291f6e8f3. [Accessed 03 08 2022].

[13] Velodyne Lidar, “Velabit,” [Online]. Available: https://velodynelidar.com/products/velabit/. [Accessed 03 08 2022].

[14] Lucid vision labs, “Understanding vision technology,” [Online]. Available: http://thinklucid.cn/tech-briefs/understanding-image-sensors/. [Accessed 21 08 2022].

[15] B. Gary and K. Adrian, Learning OpenCV, Sebastopol: O’Reilly Media, 2008.

[16] R. Owens, “Camera calibration,” [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT9/node2.html. [Accessed 18 09 2022].

[17] photographylife, “What is Lens Distortion?,” [Online]. Available: https://photographylife.com/what-is-distortion. [Accessed 18 09 2022].

[18] E. Trucco, Introductory Techniques for 3-D Computer Vision, New Jersey: Prentice-Hall Inc., 1998.

[19] Teledyne, “Field of View and Angular Field of View,” [Online]. Available: https://www.princetoninstruments.com/learn/camera-fundamentals/field-of-view-and-angular-field-of-view. [Accessed 18 09 2022].

[20] photographylife, “What is aperture in photography,” [Online]. Available: https://photographylife.com/what-is-aperture-in-photography. [Accessed 18 09 2022].

[21] First Principles of Computer Vision , “Youtube - Depth of Field | Image Formation,” [Online]. Available: https://www.youtube.com/watch?v=v5OE90eVIXo&t=137s&ab_channel=FirstPrinciplesofComputerVision. [Accessed 18 09 2022].

[22] D. Chotrov, Z. Uzunova, Y. Yordanov and S. Maleshkov, “Mixed-Reality Spatial Configuration with a ZED Mini Stereoscopic Camera.,” Unknown, 2018.

[23] G. Gerig, “Image Rectification (Stereo),” [Online]. Available: http://www.sci.utah.edu/~gerig/CS6320-S2013/Materials/CS6320-CV-F2012-Rectification.pdf. [Accessed 18 09 2022].

[24] University of Waterloo, “Direct Methods in Visual Odometry,” [Online]. Available: http://wavelab.uwaterloo.ca/slam/2017-SLAM/Lecture14-Direct_visual_inertial_odometry_and_SLAM/slides.pdf. [Accessed 18 09 2022].

[25] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2022.

[26] MathWorks, “Machine Learning Onramp,” [Online]. Available: https://de.mathworks.com/learn/tutorials/machine-learning-onramp.html. [Accessed 18 09 2022].

[27] J. Kämpfer, C. Olbrich and M. Lapschin, “Projektarbeit im Rahmen der Veranstaltung Aktive Systeme in der Fahrwerkstechnik - Implementierung einer

Verkehrszeichenerkennung auf einem Raspberry Pi für ein 1:8 Fahrzeugmodell," Hochschule für Angewandte Wissenschaften Hamburg, Hamburg, 2021.

[28] Taxibot, "IAI's TaxiBot® Granted Certification for Operation With the Airbus A320 Aircraft Family," [Online]. Available: https://www.taxibot-international.com/certification-airbus-a320. [Accessed 18 09 2022].

[29] Schiphol airport, "Sustainable taxiing and the Taxibot," Schiphol airport, Sustainable taxiing and the Taxibo, -.

[30] AIRBUS S.A.S. Customer Services Technical Data Support and Services, A320/A320NEO AIRCRAFT CHARACTERISTICS AIRPORT AND MAINTENANCE PLANNING, Blagnac Cedex: AIRBUS S.A.S. Customer Services Technical Data Support and Services, 2014.

[31] Adobe, "A guide to wide-angle lenses," [Online]. Available: https://www.adobe.com/creativecloud/photography/discover/wide-angle-lens.html#:~:text=A%20wide%2Dangle%20lens%20has,angle%20lens%20in%20their%20kit.. [Accessed 18 09 2022].

[32] Basler, "a2A4096-30ucPRO - Basler ace 2," [Online]. Available: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace2/a2a4096-30ucpro/. [Accessed 18 09 2022].

[33] Basler, "Kowa Lens LM6HC F1.8 f6mm 1" - Lens," [Online]. Available: https://www.baslerweb.com/en/products/lenses/fixed-focal-lenses/kowa-lens-lm6hc-f1-8-f6mm-1/.

[34] Basler, "Basler Lens C23-3520-5M-P f35mm - Lens," [Online]. Available: https://www.baslerweb.com/en/products/lenses/fixed-focal-lenses/basler-lens-c23-3520-5m-p-f35mm/. [Accessed 18 09 2022].

[35] Python.org, [Online]. Available: https://www.python.org/downloads/. [Accessed 25 08 2022].

[36] Pip Documentation, “Installation,” [Online]. Available: https://pip.pypa.io/en/stable/installation/. [Accessed 25 08 2022].

[37] Git, “1.5 Getting Started - Installing Git,” [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git. [Accessed 18 09 2022].

[38] Jupyter / IPython Notebook Quick Start Guide, “1. What is the Jupyter Notebook,” [Online]. Available: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html. [Accessed 08 25 2022].

[39] Homebrew, “Install Homebrew,” [Online]. Available: https://brew.sh/. [Accessed 18 09 2022].

[40] C. Olbrich, Konzeptentwicklung, Bewertung und Implementierung einer Umfelderkennung im AUDEx-Projekt, Hamburg: HAW Hamburg, 2022.

[41] Tensorflow 2 Object Detection API tutorial, “Installation,” [Online]. Available: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html#downloading-the-tensorflow-model-garden. [Accessed 25 08 2022].

[42] Nicknochnack, “Github.com - TFOD course,” [Online]. Available: https://github.com/nicknochnack/TFODCourse. [Accessed 18 09 2022].

[43] Tensorflow.org, [Online]. Available: https://www.tensorflow.org/. [Accessed 25 08 2022].

[44] Github, “Tensorflow Model Garden,” [Online]. Available: https://github.com/tensorflow/models. [Accessed 25 08 2022].

[45] Github, “Coco API,” [Online]. Available: https://github.com/cocodataset/cocoapi. [Accessed 25 08 2022].

[46] Github, “Protocol buffers,” [Online]. Available: https://github.com/protocolbuffers/protobuf/releases. [Accessed 25 08 2022].

[47] Github, “TensorFlow 2 Detection Model Zoo,” [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. [Accessed 18 09 2022].

[48] Open Source Computer Vision, “Camera Calibration and 3D Reconstruction,” [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a . [Accessed 18 09 2022].

[49] C. Rackwitz, “OpenCV Forum,” [Online]. Available: https://forum.opencv.org/t/bad-disparity-map-with-sgbm-algorithm/8209/5. [Accessed 18 09 2022].

[50] Technische Universität München, “Stereo Vision I: Rectification and Disparity,” [Online]. Available: https://campar.in.tum.de/twiki/pub/Chair/TeachingWs11Cv2/3D_CV2_WS_2011_Rectification_Disparity.pdf. [Accessed 18 09 2022].

[51] G. Föll, “Stereo Vision: Vergleich verschiedener Algorithmen zur Lösung des Korrespondenzproblems,” Hochschule für Angewandte Wissenschaften Hamburg, Hamburg, 2010.

[52] M. Gschwindt, “Bachelorarbeit Integration eines Echtzeit-Algorithmus zur Objektrekonstruktion aus Stereo- Bilddaten in eine Simulationsumgebung zur Nahbereichsnavigation im Orbit,” Technische Universität München, München, 2016.

[53] Open Source Computer Vision, “cv::ximgproc::DisparityWLSFilter Class Reference,” [Online]. Available: https://docs.opencv.org/3.4/d9/d51/classcv_1_1ximgproc_1_1DisparityWLSFilter.html. [Accessed 18 09 2022].

[54] BHawk, “Stackoverflow.com:How to interpret Disparity value,” [Online]. Available: https://stackoverflow.com/questions/44184676/how-to-interpret-disparity-value. [Accessed 18 09 2022].

[55] Melles Griot, “Fundamental Optics,” *CVI Melles Griot 2009 Technical Guide,,* vol. 2, no. 1, 2009.

[56] FRAMOS, “DEPTH SENSING TECHNOLOGIES OVERVIEW,” [Online]. Available: https://www.framos.com/en/products-solutions/3d-depth-sensing/depth-sensing-technologies. [Accessed 03 08 2022].

[57] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby and A. Mouzakitis, “A Survey on 3D Object Detection Methods for Autonomous Driving Applications,” in *TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, IEEE , 2019.

[58] J. Ibanez-Guzman and Y. Li, “Lidar for Autonomous Driving,” *IEEE SIGNAL PROCESSING MAGAZINE,* no. July2020, 2020.

[59] J. Chouinard, “How to use Wget: Install, Commands and Examples (Mac & Windows),” [Online]. Available: https://www.jcchouinard.com/wget/#Download_Wget_on_Windows. [Accessed 18 09 2022].

8 Appendix

Derivation of depth of field

Using the notation of Figure 11, the blur circle for an object at a closer distance to the camera than the plane of focus is, using similar triangles [18] [55]([21]):

$$\frac{b}{D} = \frac{|i_1 - i|}{i_1} \quad \rightarrow \quad b = D\frac{|i_1 - i|}{i_1} \tag{30}$$

The lens equation for the point in focus is:

$$\frac{1}{F} = \frac{1}{i} + \frac{1}{Z} \quad \rightarrow \ i = \frac{Z * F}{Z - F} \tag{31}$$

And for the point closer to the camera

$$\frac{1}{F} = \frac{1}{i_1} + \frac{1}{Z_1} \quad \rightarrow \ i_1 = \frac{Z_1 * F}{Z_1 - F} \tag{32}$$

Bringing equations (31) and (32) to the form of equation (30):

$$i_1 - i = \frac{Z_1 * F}{Z_1 - F} - \frac{Z * F}{Z - F} = \frac{F^2}{(Z_1 - F)(Z - F)} * (Z - Z_1) \tag{33}$$

Substituting the denominator in equation (30) for (32) and the enumerator for (33) yields:

$$b = D * F * \frac{(Z - Z_1)}{Z_1 * (Z - F)} = \frac{F^2}{N}\frac{(Z - Z_1)}{Z_1(Z - F)} \tag{34}$$

Analogously, the blur circle for the objects farther from the camera than the plane of focus is:

$$b = \frac{F^2}{N}\frac{(Z_2 - Z)}{Z_2(Z - F)} \tag{35}$$

The depth of field is determined from equations (34) and (33) yielding:

$$Z_2 - Z_1 = \frac{2ZF^2bN(Z - F)}{F^4 - b^2N^2(Z - F)^2} \tag{36}$$

Derivation epipolar geometry

To find the geometric relationship between the two cameras in Figure 13, i.e., matrices t and R, a mathematical approach called the *epipolar constraint* is used. It uses two observations:

The vectors $\vec{t}$, $\overrightarrow{Q_l}$ and $\overrightarrow{Q_r}$ all lie in the same epipolar plane. A normal vector to this plane can be described, for instance, using the cross product:

$$\vec{n} = \vec{t} \times \overrightarrow{Q_l} \tag{37}$$

Furthermore, the normal vector must be perpendicular to vectors $\overrightarrow{Q_l}$ and $\overrightarrow{Q_r}$. Hence, the dot-product must equal zero:

$$\overrightarrow{Q_r} \cdot \vec{n} = 0 \ and\ \overrightarrow{Q_l} \cdot \vec{n} = 0 \tag{38}$$

To provide a better fundamental understanding of these operations in later implementation chapters, the matrix notation will be used instead of vector notation. Equation (38) becomes:

$$Q^T n = 0 \ , where\ Q^T = [x\ y\ z\ ] \tag{39}$$

Equation (37) can be written using matrix notation as:

$$\vec{n} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} X_{c,l} \\ Y_{c,l} \\ Z_{c,l} \end{bmatrix} \tag{40}$$

To provide a system of equations that includes the required translation matrix t, Q_r is expressed as $Q_l - t$ in equation (39) and the normal vector is substituted for (37), which yields the *coplanar constraint*:

$$(Q_l - t)^T (t\ x\ Q_l) = 0 \tag{41}$$

The second observation is that the point described in matrices Q_l and Q_r has the same coordinates when viewed from the other camera, after coordinate transformation between the cameras (defined for world-to-camera transformation as $Q_c = R(Q_w - t)$), which is used for the second constraint:

$$Q_r = R(Q_l - t) \tag{42}$$

Since $R^{-1} = R^{T}$, as mentioned previously, equation (42) can be modified to:

$$R^{-1}Q_r = R^{-1}R(Q_l - t) \quad \rightarrow \quad (Q_l - t) = R^{-1}Q_r \tag{43}$$

Substituting eq. (41) for (43) yields:

$$Q_r^T R t Q_l = 0 \tag{44}$$

Which is:

$$\begin{bmatrix} X_{c,r} \; Y_{c,r} \; Z_{c,r} \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} X_{c,l} \\ Y_{c,l} \\ Z_{c,l} \end{bmatrix} = 0$$

The product of $R\ t$ is called the essential matrix E. It relates two points in three dimensional coordinates. However, the camera coordinates of the left and right camera in equation (44) are not known during the camera calibration process. The known coordinates are the image coordinates u and v for both cameras.

The right camera's projective transformation from equation (7) becomes:

$$Q_{c,r}^T = {M^{-1}}^T Z_{c,r} \; [u_r \; v_r \; 1]$$

Inserting $Q_{c,r}^T$ in (44) becomes:

$$Z_{c,r}[u_r \; v_r \; 1]{M_r^{-1}}^T \; E \; M_l^{-1} Z_{c,l} \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = 0 \tag{45}$$

The depth of the point relative to both cameras ($Z_{c,r}, Z_{c,l}$) cannot be zero, since this would indicate that this point lies in the centre of projection, which is inside of the cameras. Therefore $Z_{c,r} \neq 0$ and $Z_{c,l} \neq 0$. Hence, the remaining terms in equation (45) must be zero:

$$[u_r \; v_r \; 1] \; {M_r^{-1}}^T E \; M_l^{-1} \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = 0 \tag{46}$$

The product of the matrices in equation(46) relating the left camera's image coordinates to the ones of the right camera is called the fundamental matrix F with $F = \; {M_r^{-1}}^T E \; M_l^{-1}$. Equation (46) becomes

$$[u_r\ v_r\ 1]\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}\begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = 0 \qquad (47)$$

Taxibot geometry requirements

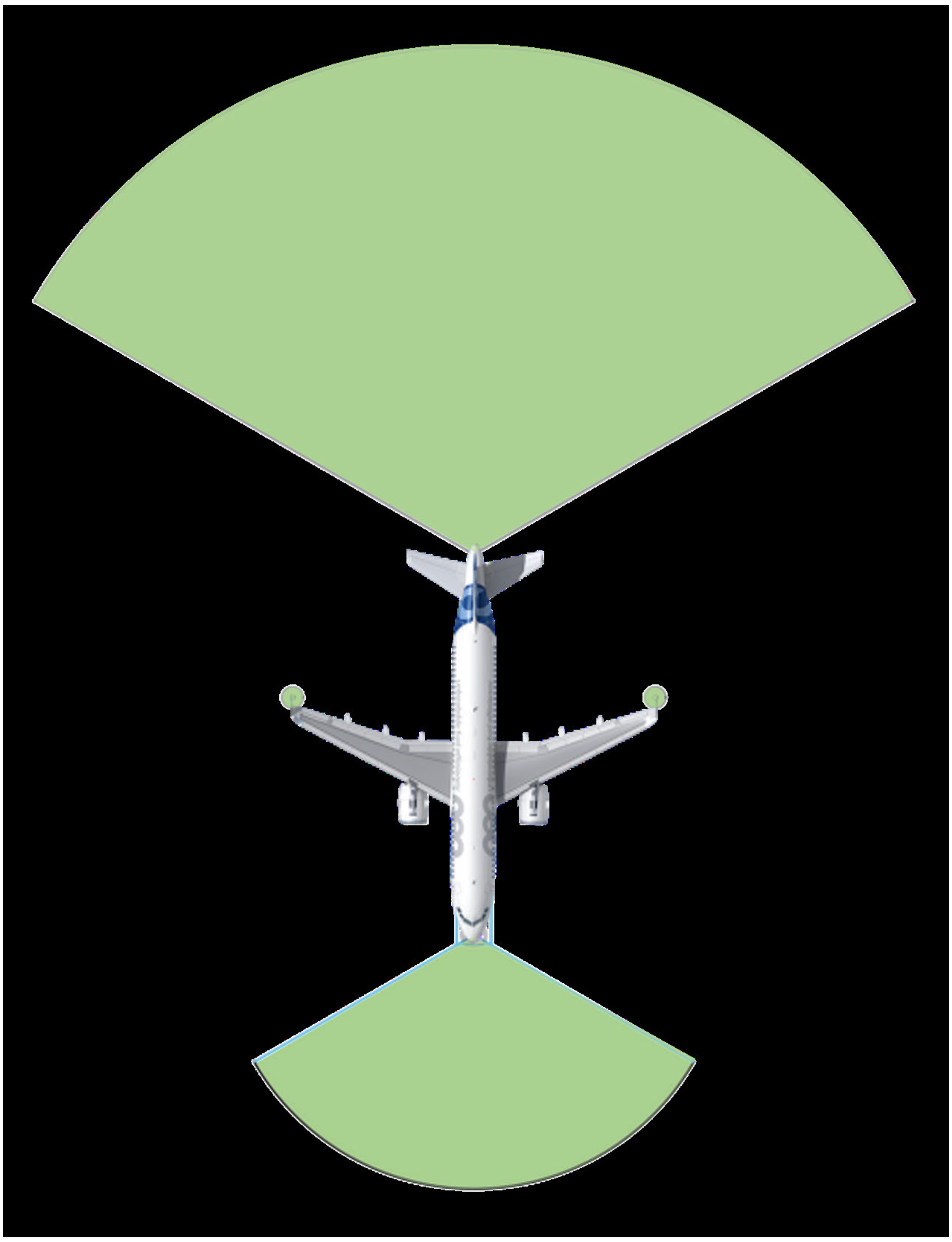


Figure 54: Geometric requirements defined in [6]

Taxibot FOV configuration

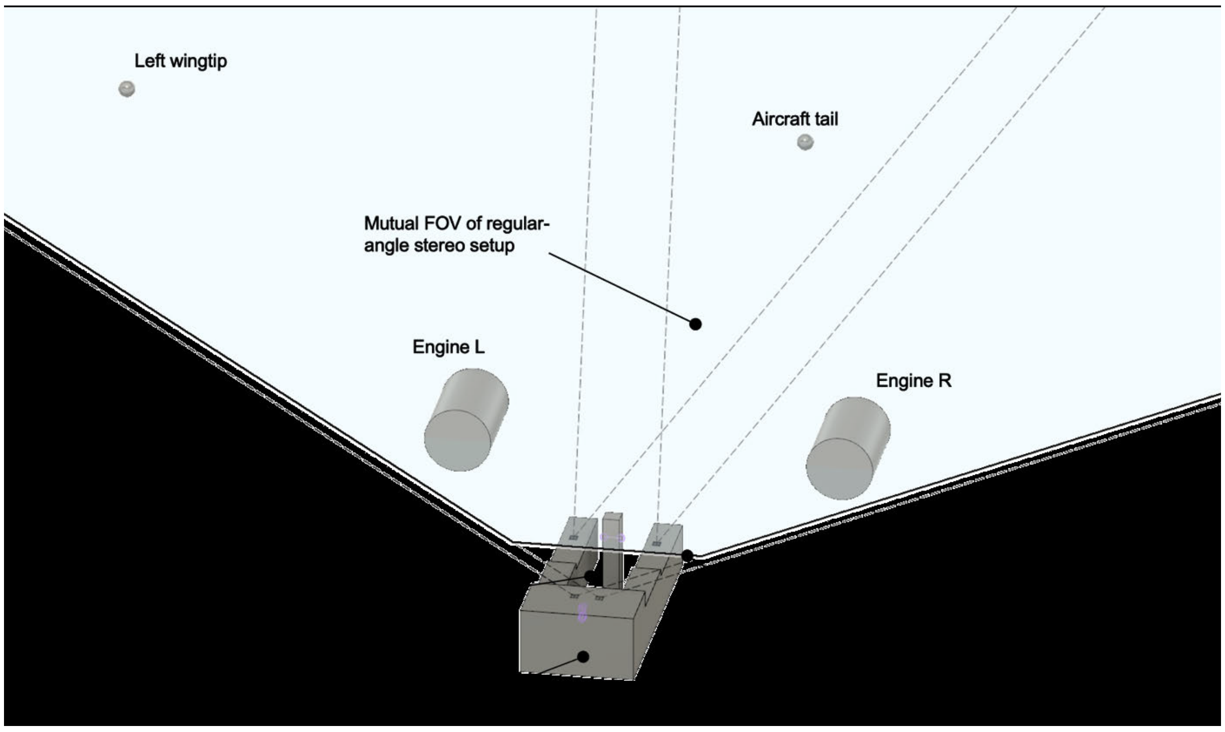


Figure 55: Frontal FOVs Taxibot

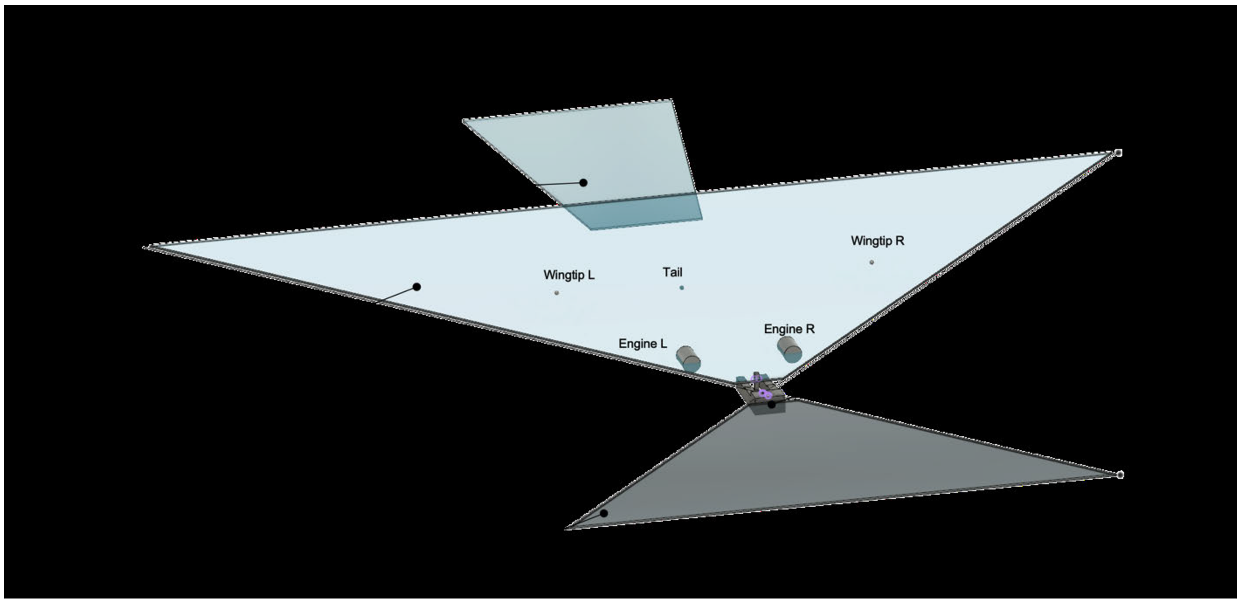


Figure 56: Forward and backward DOF

Hardware prototype

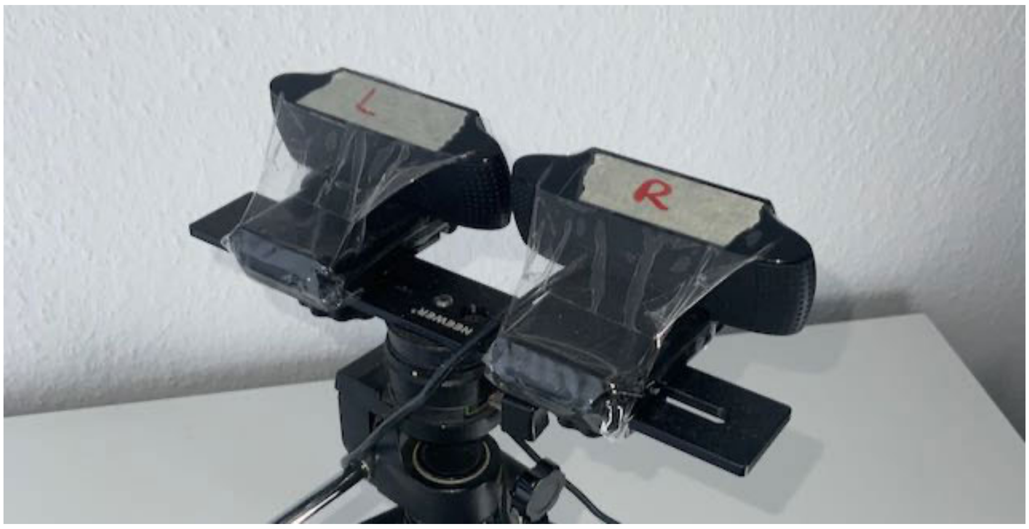


Figure 57: Top view of hardware prototype

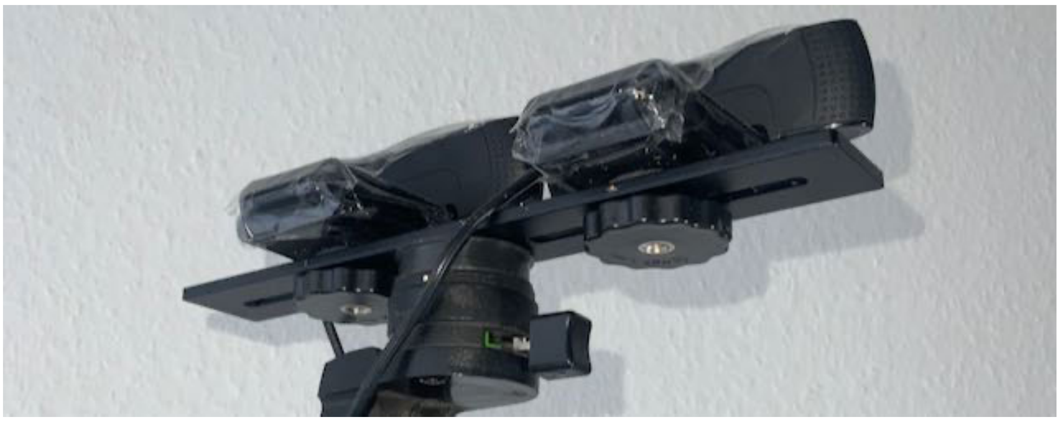


Figure 58: Bottom view of hardware prototype

Hardware cost calculation

Table 6: Estimation of hardware costs

Number	Component	Cost p.p	Cost sum
6	Basler ace 2 - a2A4096-30ucPRO	€ 1.399,00	€ 8.394,00
4	Kowa Lens LM6HC	€ 549,00	€ 2.196,00
2	Basler Lens C23-3520	€ 99,00	€ 198,00
Total			€ 10.788,00

Table 7: Hardware used for stereo vision

Device	Characteristics
MacBook Pro (macOS Monterey)	3,1 GHz Dual-Core Intel Core i5, 16 GB RAM

Directory structure

- …
  - BachelorNB
    - envTeleTaxi
    - projectTeleoperatedTaxibot
      - 0_UserInput
        - backgrounds
        - objects
        - scripts
        - trainedModels
      - 1_Calibration
        - individual
          - camL
          - camR
        - stereo
          - camL
          - camR
      - 2_ObjectDetection
        - 1_Preprocessing
        - 2_Tensorflow
          - models
          - protoc
          - workspace
            - annotations
            - images
            - models
            - pre_trained_models
        - 3_Output

Figure 59: Structure of directory

Rectification check

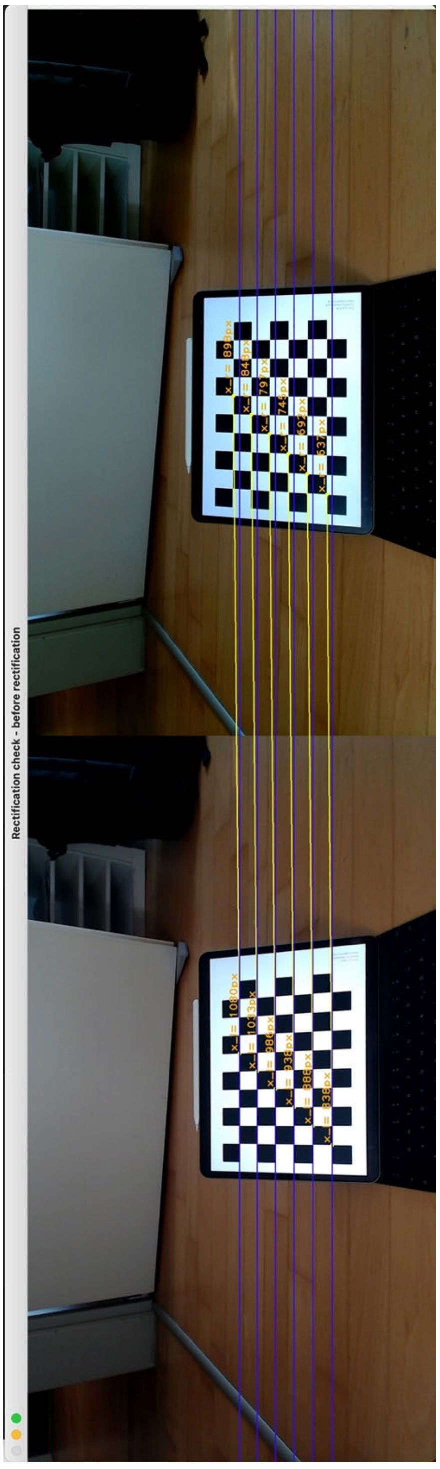
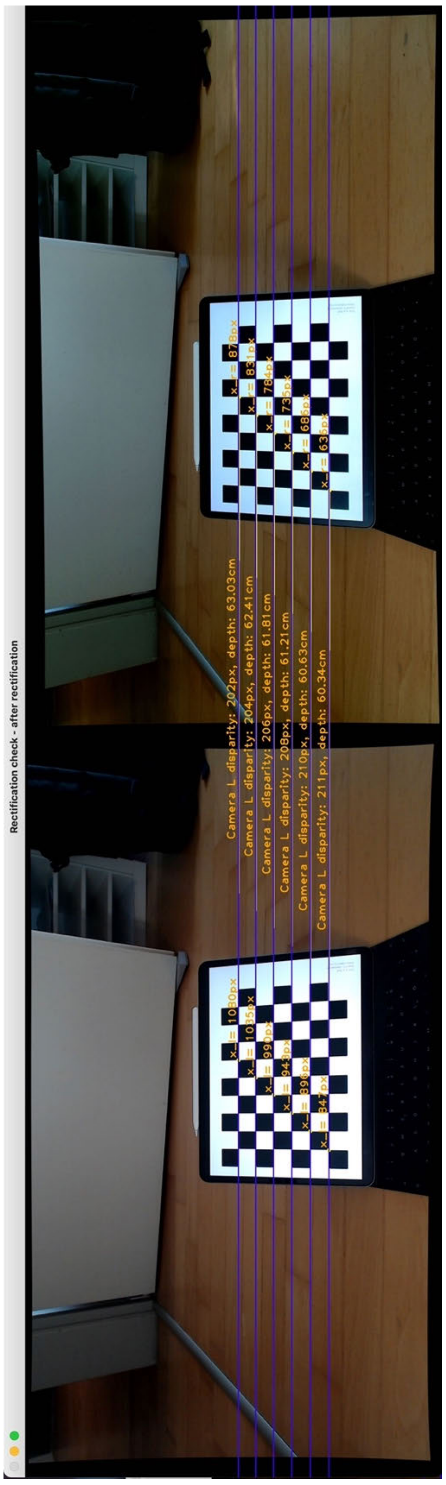


Figure 60: Rectification check, original frames



```
Camera L disparity: 202px, depth: 63.03cm
Camera L disparity: 204px, depth: 62.41cm
Camera L disparity: 206px, depth: 61.81cm
Camera L disparity: 208px, depth: 61.21cm
Camera L disparity: 210px, depth: 60.63cm
Camera L disparity: 211px, depth: 60.34cm
```

Figure 61: Rectification check, rectified frames

Sample of images produces with albumentation

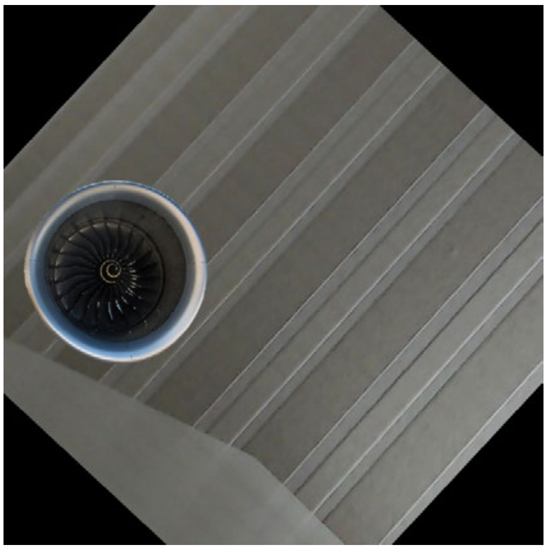


Figure 62: First sample of modified images using Albumentation

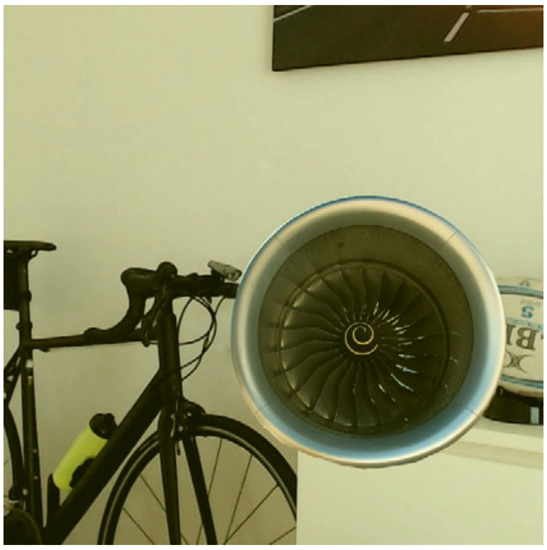


Figure 63: Second sample of modified images using Albumentation

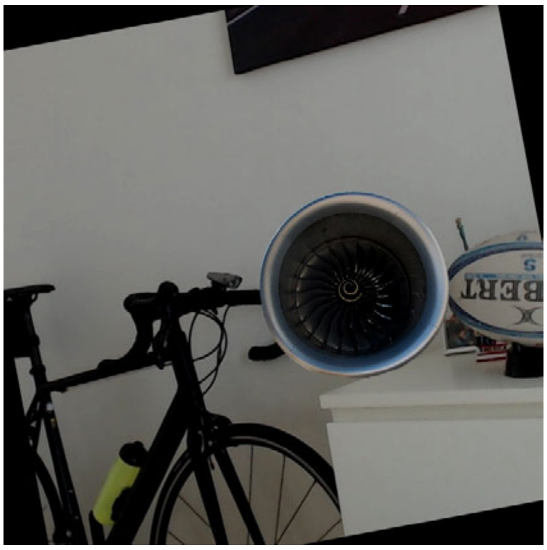


Figure 64: Third sample of modified images using Albumentation

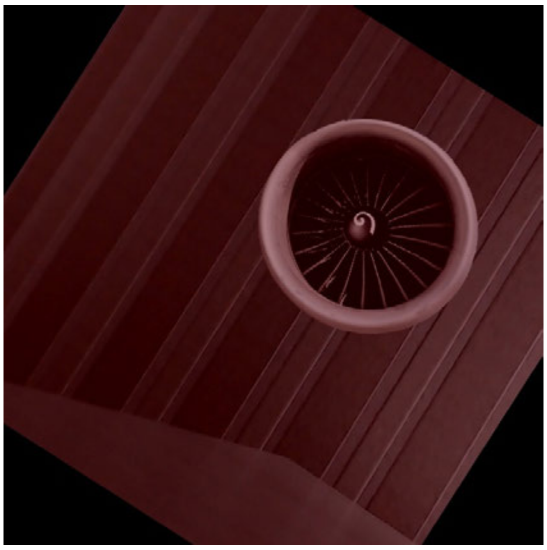


Figure 65: Fourth sample of modified images using Albumentation

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Bakula

Vorname: Niklas Joshua

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzeptentwicklung, Bewertung und Implementierung einer Methode zur bildbasierten Abstandsermittlung im AUDEx-Projekt

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

Hamburg	19.09.2022	[redacted]
Ort	Datum	Unterschrift im Original