Tampere University

Joni Tepsa

3D PERCEPTION SYSTEM DESIGN FOR HEAVY-DUTY FIELD ROBOTS

Master's Thesis
Faculty of Engineering and Natural Sciences
Examiners: Professor Jouni Mattila
Researcher Pekka Mäenpää
December 2021

ABSTRACT

Joni Tepsa: 3D Perception System Design for Heavy-Duty Field Robots
Master's Thesis
Tampere University
Master's Programme in Automation Engineering
December 2021

Autonomous driving and mobile robotics are achieving significant milestones, and perception systems are in an essential role for their versatile data and various applications. The objectives of this thesis were to find out whether it is possible to implement a 3D perception system for heavy-duty field robots using a stereo camera, software environment Robotic Operating System (ROS), and Jetson Xavier NX Embedded PC and to form design principles for 3D perception systems in heavy-duty field robots.

The heavy-duty field robots operate in various conditions and environments, where they can cause incidents to the surrounding environment. A list of requirements from five categories was formed. The heavy-duty field is the first category, including requirements that the 3D perception system should be robust to vibration and designed to fit the correct manipulation scope. The environmental requirements are that the system should be suitable to marker-less environments and resistant to adverse conditions. Safety standards and system placement are the requirements in the safety category. The real-time category has requirements for choosing algorithms, such as system optimization criteria and calculation architecture. Energy efficiency and system dependability are requirements in the embedded category.

The implemented 3D perception system was developed in C++ and CUDA programming languages to be run on the ROS environment. The whole system was run on the same ROS network from image acquisition to the filtered point cloud. The disparity image calculation was done using semi-global matching, developed by Hernandez-Juarez et al., and the point cloud filtering was done using the open-source Cupoch library. The principal used libraries were OpenCV, Point Cloud Library, and Cupoch.

The implemented 3D system achieved 2,5 frames per second with a 74 degree horizontal field of view. The system was evaluated to fulfil all of the set requirements for heavy-duty field robots that were defined for this application, except soft real-time, system dependability, and safety standards were not met. A set of five design principles were formed covering the ten requirements: 1) perception sensors with their placement are in an essential role during the designing, 2) the target environment should be kept as one of the main design criteria during the development, 3) the relevant safety standards should be used as a guide during the development, 4) the algorithm choosing and tailoring should be done carefully and with time, and 5) one of the main focus areas of a 3D perception system development is to have a dependable system.

In conclusion, the implemented 3D perception system needs more development, especially the observation rate and system dependability, and that more refining is suggested for the design principles through experience and testing.

Keywords: 3D perception, computer vision, heavy-duty field robot, stereo camera, Jetson Xavier NX

TIIVISTELMÄ

Joni Tepsa: Konenäköjärjestelmän suunnittelu työkonerobotteihin
Diplomityö
Tampereen yliopisto
Automaatiotekniikan DI-ohjelma
Joulukuu 2021

Autonominen ajo ja mobiilirobotiikka ovat saavuttamassa merkittäviä virstanpylväitä, ja havainnointijärjestelmät ovat tärkeässä asemassa niistä saatavan monipuolisen tiedon ja erilaisten sovellusten vuoksi. Tämän opinnäytetyön tavoitteina oli selvittää, onko mahdollista toteuttaa 3D-havainnointijärjestelmä työkoneroboteille käyttäen stereokameraa, Robotic Operating System (ROS) -ohjelmointiympäristöä ja Jetson Xavier Nx sulautettu tietokonetta, sekä muodostaa suunnitteluperiaatteet työkonerobottien 3D-havainnointijärjestelmiin.

Työkonerobotit toimivat erilaisissa olosuhteissa ja ympäristöissä, joissa ne voivat aiheuttaa vahinkoa. Vaatimusluettelo muodostuu viidestä kategoriasta. Työkoneet on ensimmäinen kategoria, johon sisältyy vaatimuksia, joiden mukaan 3D-havainnointijärjestelmä on oltava tärinänkestävä ja se on suunniteltava siten, että se sopii oikeaan käsittelylaajuuteen. Ympäristökategoriaan kuuluu, että järjestelmän on sovelluttava merkitsemättömiin ympäristöihin ja kestettävä vaikeat sääolosuhteet. Turvallisuusstandardit ja järjestelmän sijoittelu ovat turvallisuuskategorian vaatimuksia. Reaaliaikakategoriassa on algoritmien valintaa koskevia vaatimuksia, kuten järjestelmän optimointikriteerit ja laskenta-arkkitehtuuri. Energiatehokkuus ja järjestelmän luotettavuus ovat vaatimuksia sulautettujen järjestelmien kategoriassa.

Diplomityössä toteutettu 3D-havainnointijärjestelmä kehitettiin käyttäen C++ ja CUDA ohjelmointikieliä suoritettavaksi ROS-ympäristössä. Koko järjestelmä kuvien ottamisesta suodatettuun pistepilveen ajettiin samassa ROS-ympäristössä. Syvyyskuvan laskennassa käytettiin Hernandez-Juarez ym. kehittämää semi-globaalia sovitusta ja pistepilven suodatuksessa käytettiin avoimen lähdekoodin Cupoch-kirjastoa. Tärkeimmät käytetyt kirjastot olivat OpenCV, Point Cloud Library ja Cupoch.

Toteutettu 3D-järjestelmä saavutti 2,5 havainnointia sekunnissa 74 asteen vaakasuuntaisella näkökentällä. Järjestelmän arvioitiin täyttävän kaikki työkoneroboteille asetetut vaatimukset, jotka oli määritelty tätä sovellusta varten, paitsi että pehmeä reaali-aika, järjestelmän luotettavuus ja turvallisuustandardit eivät täyttyneet. Muodostettiin viisi suunnitteluperiaatetta, jotka kattavat kymmenen vaatimusta: 1) havainnointiantureilla ja niiden sijoittelulla on tärkeä rooli suunnittelussa, 2) kohdeympäristö olisi pidettävä yhtenä tärkeimmistä suunnittelukriteereistä suunnittelun aikana, 3) asianmukaisia turvallisuusstandardeja tulee käyttää ohjeena kehityksen aikana, 4) algoritmin valinta ja räätälöinti olisi tehtävä huolellisesti ja ajan kanssa sekä 5) yksi 3D-havainnointijärjestelmän kehittämisen pääpainoalueista on oltava luotettava järjestelmä.

Johtopäätöksenä voidaan todeta, että toteutettua 3D-havainnointijärjestelmää on kehitettävä edelleen erityisesti havainnointitaajuuden ja järjestelmän luotettavuuden osalta, ja että suunnitteluperiaatteita ehdotetaan hiottavaksi edelleen kokemuksen ja testauksen avulla.

Avainsanat: 3D havainnointi, konenäkö, työkonerobotti, stereokamera, Jetson Xavier NX

PREFACE

This thesis has been a unique project, and it was more interesting than I had anticipated, while the writing was at the same time tedious and exciting. The implementation gave me a lot of new skills, which I hope I can use in the future. Alongside this thesis, my student career ends, and the new challenges begin in the work-life.

I want to thank Jouni Mattila for guiding and offering the thesis topic and Pekka Mäenpää for reviewing it. It has also been a pleasure to work in Jouni's research team during this last year.

Big thank you also to my friends and family who have supported me during my whole studies, giving me motivation and inspiration when needed.

I hope the reader finds valuable information from this thesis to be used in engineering projects and research.

Tampere, 27th December 2021

Joni Tepsa

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AM	Amplitude modulation
AOI	Area of Interest
API	Application Programming Interface
ASIC	Application-specific integrated circuit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSP	Digital Signal Processor
FMCW	Frequency-modulated continous-wave
FOV	Field of view
FPGA	Field Programmable Gate Array
FPS	Frames per second
GPIO	General purpose I/O
GPS	Global Positioning System
GPU	General Processing Unit
GUI	Graphical user interface
LiDAR	Light detection and ranging
MEMS	Microelectromechanical systems
OPA	Optical phased-array
PCL	Point Cloud Library
Radar	Radio detecting and ranging
ROS	Robotic Operating System
SGM	Semi-global stereo matching
SLAM	Simultaneous Localization and Mapping
UDP	User Datagram Protocol
b	Baseline of stereo camera
B	Sweep bandwidth

c	Speed of light in a vacuum
d	Disparity value of a pixel
d_{max}	maximum disparity value in SGM algorithm
f	Focal length of a lens
f_1	Focal length of left camera
f_2	Focal length of right camera
f_b	Beat frequency
f_{bd}	Beat frequency during down-ramp
f_{bu}	Beat frequency during up-ramp
f_d	Doppler frequency
n	Refractive index
P_1	penalty for small disparity changes in SGM algorithm
P_2	penalty for large disparity continuities in SGM algorithm
Q	Reprojection matrix
r_{total}	path direction count in SGM algorithm
R	Range from a laser source to an object.
$\vec{T}$	Translation vector
T_s	Laser wave period
T_x	x-component of the translation vector
x_1	x-coordinate of a pixel in left image
x_2	x-coordinate of a pixel in right image
X_c	x-coordinate of a point in stereo system coordinates
X_h	x-coordinate of a pixel in homogeneous coordinates
X_w	x-coordinate of a point in world coordinates
y_1	y-coordinate of a pixel in left image
y_2	y-coordinate of a pixel in right image
Y_c	y-coordinate of a point in stereo system coordinates
Y_h	y-coordinate of a pixel in homogeneous coordinates
Y_w	y-coordinate of a point in world coordinates
Z	Depth value of a pixel
Z_c	z-coordinate of a pixel in stereo system coordinates
Z_h	z-coordinate of a pixel in homogeneous coordinates

Z_w	z-coordinate of a point in world coordinates
δ	Principal point in a image plane
δ_1	Principal point of a left image plane
$\delta_{1,x}$	x-coordinate of the principal point of a left image plane
$\delta_{1,y}$	y-coordinate of the principal point of a left image plane
δ_2	Principal point of a right image plane
$\delta_{2,x}$	x-coordinate of the principal point of a right image plane
τ	Time-of-flight

1. INTRODUCTION

The last couple of years have been revolutionary for autonomous driving and mobile robotics in computer vision and perception systems, achieving significant milestones. It could be said that autonomous driving has entered the pre-industrialization phase with innovations from various car manufacturers. The first Light detection and ranging (LiDAR) sensor has been fitted to a fully commercialized car, Audi A8L 2019, (Li and Ibanez-Guzman 2020) and the computer vision market share is expected to have a 10,8 % compound annual growth rate in the upcoming few years. Being worth 11,35 Billion US dollars in the year 2020 (Rajput 2021).

The output of a perception system can be divided into three main categories: 1) physical description, 2) semantic description, and 3) intention prediction. The first consists of features easily measurable, for example, pose, velocity, and shape. The information about the category of objects is in the semantic description. The last main category, intention prediction, informs the behaviour of objects. (Li and Ibanez-Guzman 2020) Given these points, a perception system is a versatile sensor system that can give much information about the surrounding environment and thus, has become a crucial part of mobile robotics.

A perception system has many applications in robotics. Starting from the navigational requirement, that the robot must be able to localize itself, using Simultaneous Localization and Mapping (SLAM) or other navigation algorithms, and typically these algorithms use perception data in the localization (Stachniss 2009). The data from the perception system is used to avoid obstacles in autonomic guidance to continue effective navigation. In heavy-duty human-robot collaboration applications, the perception sensor is a vital element, estimating the pose of the manipulator and detecting humans working alongside the robot in order to allow autonomous operation and avoid hazards (Liang et al. 2019).

The first research objective is to implement a 3D perception system that could be integrated into a heavy-duty field robot allowing autonomous functions using a stereo camera, Jetson embedded PC, and Robotic Operating System (ROS). The system should fulfil the requirements described in the background section of this thesis. Another aim of this work is to form design principles for 3D perception systems in heavy-duty field robots. That would serve as a design guide for upcoming researchers to simplify and straighten the design process of a computer vision system for mobile robots working in heavy-duty industries.

To achieve all research objectives of this thesis, the following research questions seek to be answered:

1. Is it possible to implement a 3D perception system using a stereo camera, Jetson embedded PC, and ROS, which meets the requirements set to heavy-duty field robots?
2. What must be considered when designing a 3D perception system for a heavy-duty field robot?
3. What are the design principles of a 3D perception system for heavy-duty field robots?

The thesis is structured into three main sections. The first section, containing chapters 2 and 3, serves as the background for the methods used in this work. The chapter 2 introduces the theory of perception sensors, stereo cameras and laser scanners, widely used in computer vision. In the chapter 3 the requirements that a 3D perception system should meet in heavy-duty field robots are described. The second section is the materials and methods, describing the implementation of a 3D perception system, the second research objective, in the chapter 4. The last main section presents the results of the research objectives in the chapter 5. In the end, is the conclusions chapter 6.

2. SENSORS IN 3D PERCEPTION SYSTEM

Sensors are a fundamental part of a 3D perception system. They are the tools that allow computer programs to observe the surrounding environment and transfer that data to the controllers of the robots. Which furthermore allows increasing the level of autonomy and safety to workers. The most commonly used sensors in computer vision are monocular cameras and laser scanners. Nowadays, the variety of sensors achieving state-of-the-art results in computer vision has increased immensely. New variations of the traditional sensors have been commercialised, for example, Luminar Hydra LiDAR, which is explicitly designed for autonomous cars requiring extra-long observation range (Luminar 2021).

This section presents the theory and background of stereo cameras and laser scanners. There are many different laser scanners suitable for computer vision. Two of them are discussed in more detail, LiDAR and 2D laser scanners, due to their excellent track record in 3D perception applications. Also, the scope of this thesis focuses on heavy-duty field robots. Therefore, two commonly found LiDAR models in robot applications are in focus, spinning LiDAR and solid-state LiDAR.

2.1 Stereo Camera

Stereo cameras have been one of the fundamental perception sensors in mobile robotics for a while. Stereo camera setups consist of two similar cameras mounted parallel to each other, with a known distance between them, known as the baseline. This sensor setup simulates the human vision, known as binocular vision, responsible for depth perceiving for humans (Davies 2012). The stereo camera utilises this same principle to observe depth from the surrounding environment, which can be processed to 3D point clouds, presenting the environment as points in a three-dimensional coordinate system.

Figure 1 shows an ideal stereo camera setup, where both cameras observe the same general point located at world coordinates (X_w, Y_w, Z_w). The word ideal in this context means that the setup is perfectly undistorted, aligned, and measured. Also, the two image planes that the cameras form are exactly coplanar with each other, and the optical axes of the cameras are perfectly parallel. The optical axis is a ray from the centre of the camera through the principal point δ in the image plane. In ideal stereo camera setup, also the focal lengths, left f_1 and right f_2, of the two cameras are identical and the principal points, left δ_1 and right δ_2, are calibrated to precisely the same pixel coordinates. Lastly, the images are assumed to be row-aligned to each other. (Kaehler and Bradski 2017) Naturally, the ideal stereo camera setup is only possible in theory. All stereo cameras suffer from distortions, imperfections, and geometrical defects. These shortcomings are solved by calibrating the stereo camera, which undistorts and rectifies the images and aligns the optical axes to be as parallel as possible and the image planes as coplanar as possible. However, the theoretical details of the calibration are out of the scope of the thesis. Therefore left out, and only the working principle is shortly explained.

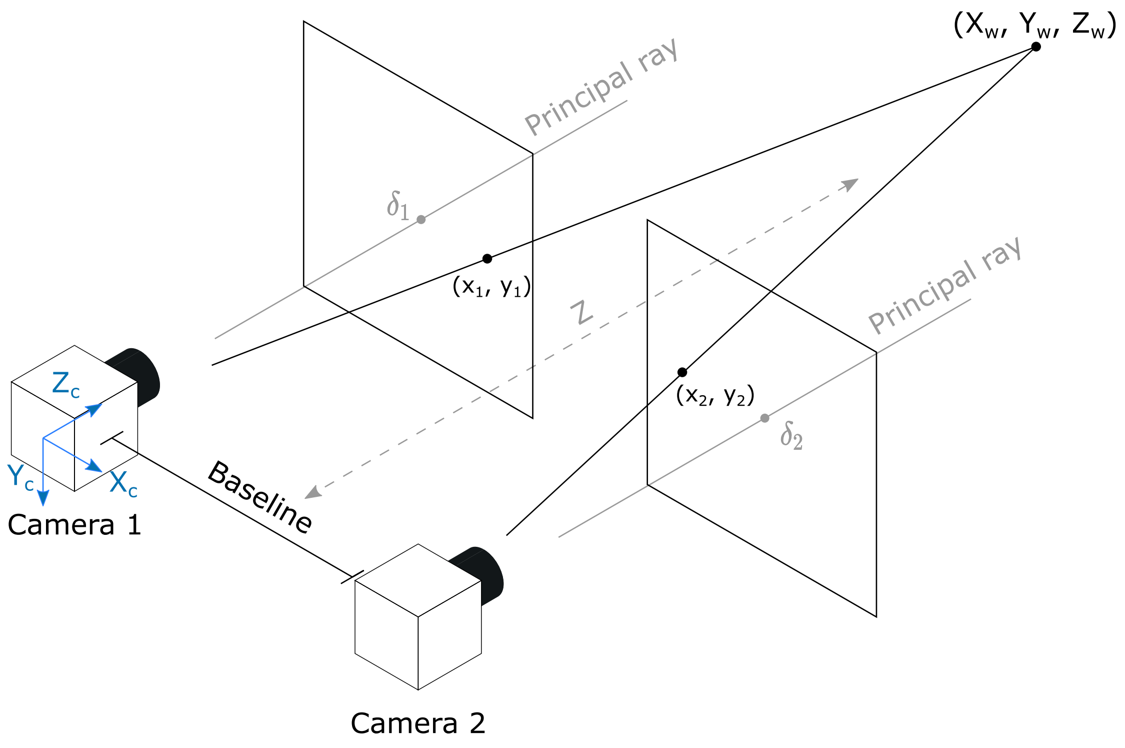


Figure 1. *Stereo vision setup principle.*

The cameras are parallel, and the distance between them is the baseline. Every time the surrounding environment is observed, it is digitised to an image, which means that the 3D points are projected into 2D locations on the image plane. The image planes can be seen in the front of the cameras in Figure 1. The general point is shown in the left image plane in location (x_1, y_1) and correspondingly in location (x_2, y_2) in the right image plane. To calculate the depth of a general point, the pixels representing the general point in both image planes must be corresponding to each other, which means that both pixels point to the same location (Davies 2012). Henceforth, the points in this example are assumed to be corresponding points. Searching and matching points to be corresponding in two image planes is out of the scope of this thesis and not discussed. With two corresponding points, the depth value of a pixel Z can be calculated, using triangulation, with the following equation

$$Z = \frac{b \cdot f}{x_1 - x_2}, \tag{1}$$

where b is the baseline of the stereo camera, f is the focal length of the lens, x_1 is the x-coordinate of the general point in the left image plane, and x_2 is the x-coordinate of the general point in the right image plane (Davies 2012). The difference in row-aligned coordinates is also known as disparity d , which is only the difference of the x-coordinates of two points in two image planes $x_1 - x_2$, divider in (1).

The perceived depth or disparity can be used in the perception system as they are, as either depth or disparity images, in which each pixel contains depth or disparity information correspondingly. However, the disparity of the pixel can be transformed to 3D coordinates in the stereo coordinate system (X_c, Y_c, Z_c), allowing to know the distance of an object in all dimensions, not just depth, and the creation of 3D point clouds. The stereo camera must be calibrated to form the mapping between disparity and 3D coordinates. There are a few different techniques to calibrate a stereo camera. 2D metric information based calibration is one of them, and it is presented here shortly (Zhang 2000). The idea is to observe a checkerboard, a board containing a chessboard looking pattern with specified square length, in various locations and poses in both cameras simultaneously. An example checkerboard is shown in Figure 2. From the images, the corners of the squares of the checkerboard are detected, and by knowing the length of the square side in the 3D scene, the camera calibration parameters can be formed (Kaehler and Bradski 2017)

From this point on, it is assumed that the stereo camera is calibrated to use the camera calibration parameters, which can be divided into intrinsic and extrinsic parameters. (Davies 2012; Heikkila and Silven 1997) A point in two dimensions can be presented in three dimensions, when the image plane coordinates of the point and the disparity value, between the stereo image pair, is known, with the following equation

$$\begin{bmatrix} X_h \\ Y_h \\ Z_h \\ W \end{bmatrix} = Q \cdot \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix}, \tag{2}$$

where Q is reprojection matrix, (X_h, Y_h, Z_h, W) are homogeneous coordinates of a point in three dimensions (Kaehler and Bradski 2017). The reprojection matrix Q is formed

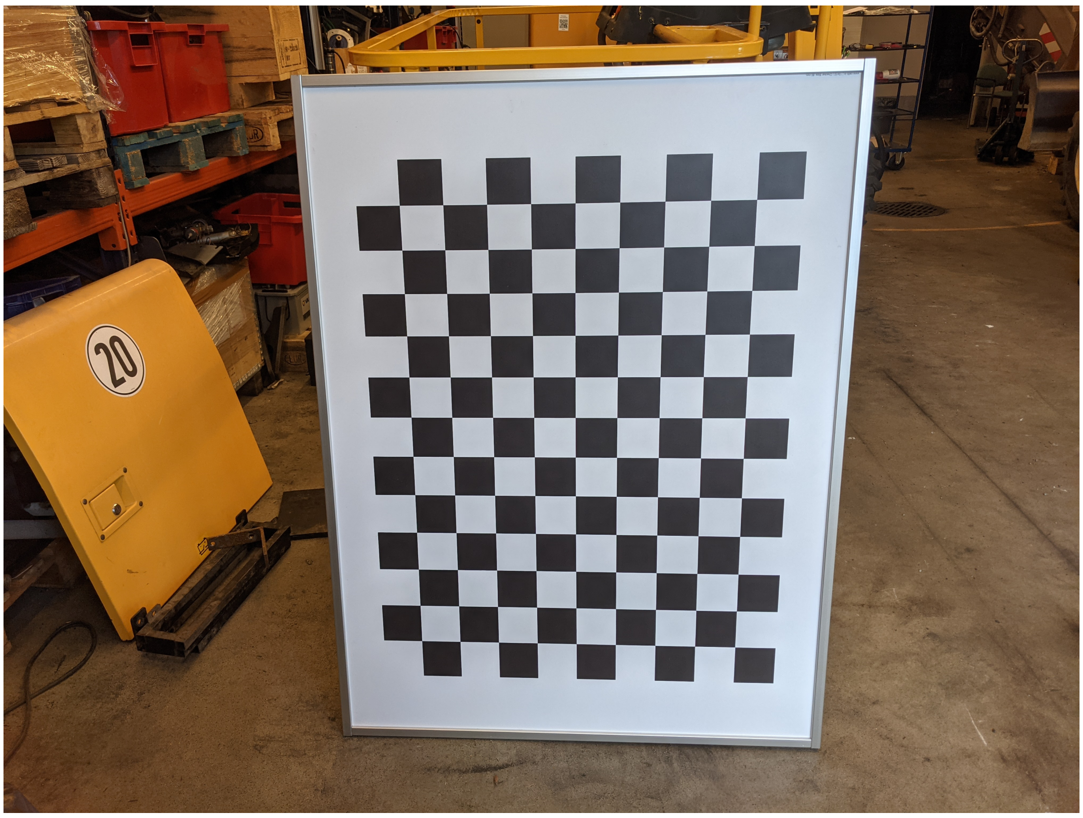


Figure 2. *Checkerboard made of an aluminium sheet containing black-and-white chessboard pattern, used to calibrate cameras.*

from the left camera intrinsic matrix elements

$$Q = \begin{bmatrix} 1 & 0 & 0 & -\delta_{1,x} \\ 0 & 1 & 0 & -\delta_{1,y} \\ 0 & 0 & 0 & f_1 \\ 0 & 0 & -\frac{1}{T_x} & \frac{\delta_{1,x}-\delta_{2,x}}{T_x} \end{bmatrix}, \tag{3}$$

where $\delta_{1,x}$ is principal point x-coordinate in left image plane, $\delta_{1,y}$ is principal point y-coordinate in left image, $\delta_{2,x}$ is principal point x-coordinate in right image plane, and T_x is x-component of translation vector $\vec{T}$, vector representing translation from the object coordinates to the stereo coordinate system. The lower-right corner element is zero, when the principal rays intersect at infinity, $\delta_{1,x} = \delta_{2,x}$. (Kaehler and Bradski 2017) From the equation (2), the stereo coordinate system 3D coordinates (X_c, Y_c, Z_c) are formed by dividing the homogeneous coordinates with W

$$(X_c, Y_c, Z_c) = (\frac{X_h}{W}, \frac{Y_h}{W}, \frac{Z_h}{W}). \tag{4}$$

The accuracy of the depth map is dependant on the length of the baseline between the cameras. A higher baseline gives better accuracy in depth observation and a larger depth estimation range. At the same time, the difficulty of finding corresponding points from the stereo pair increases, leading to badly matched points. (Davies 2012) The observable depth of a stereo camera is inversely proportional to pixel disparity value (Kaehler and Bradski 2017), from equation (1) and the disparity value is represented as an integer. Resulting in a problem that the observable depth is discretised, compared to the continuous representation that humans see. The discretised depth gives a varying depth accuracy. For example, lowering the high disparity value by one increases the depth value slightly. In contrast, lowering the low disparity value by one increases the depth value significantly. Due to the discretisation, there can be several meters between two disparity values when turned into depth values. Stereo camera as perception sensor relies a lot on the stereo calibration, which is prone to noise in the stereo pair images and rounding errors during the calibration process (Kaehler and Bradski 2017). If the stereo calibration is not good enough, the observed 3D points are not corresponding to the real 3D scene, resulting in faulty observation of the scene. The disparity measurement in a stereo camera is dependant on how light reflects from the surface of an object. In other words, there must be a difference in observed visible light to get a disparity (O' Riordan et al. 2018). This becomes a problem in certain scenarios, like when the object surface might have a shadow from another object, creating a difference in light reflection and, thus, a false disparity value. Another problematic scenario is when an observable object has a reflective metal surface, which leads to no visible light change even though the surface geometry changes (O' Riordan et al. 2018).

2.2 Laser scanners

LiDARs were first used in the airborne industry mapping Earth's ground, topographic mapping from aeroplanes and helicopters. The technology came during the '50s. It was widely adopted at the end of the 1980s, when Global Positioning System (GPS) was available to non-military usage, allowing precise location of the aeroplane or helicopter. The airborne laser scanning inspired the usage of laser scanners in other industries, and thus, the terrestrial laser scanners have come to autonomous machines and robots. (Carter et al. 2012; Vosselman and Maas 2010) In scope of this thesis, only the terrestrial laser scanners are discussed.

LiDAR is a remote sensing technology that uses laser scanners to produce precise and dense georeferenced spatial information of the surrounding environment. This information can be produced to accurate 3D representations of an object with shape and surface features for engineering purposes. (Carter et al. 2012) LiDAR systems use laser scanners to emit a focused beam of light to the surrounding environment and measure the time it takes from the emit of a beam to the reflections of the same beam coming back to the sensor. One of the advantages of LiDAR systems is that they can use several types of light, ultraviolet, visible and infrared light; all are possible (Horaud et al. 2016). Using different lights allows observing differences in objects, even though they look identical to the human eye. The LiDAR technology is based on the same principle as Radio detecting and ranging (Radar) is, but LiDAR uses discrete pulses of laser light when Radar uses radio waves. (Carter et al. 2012; Nature 1943) The core principle of how LiDAR measures the distance that the laser beam has travelled is based on light transit time estimation, which can be done in numerous ways, but the two primarily used ones in current LiDARs are the time-of-flight calculation and the Frequency-modulated continous-wave (FMCW) approach (Lee 2019; Vosselman and Maas 2010).

In time-of-flight calculation, the fundamental property is the given constant velocity for a light wave when the medium is known. The medium needs to be known because the velocity of a light wave is dependant on it. The distance travelled by the light wave (Vosselman and Maas 2010), from a laser source to an object and from the object to a light detector, can be calculated from the following equation

$$R = \frac{c}{n}\frac{\tau}{2}, \tag{5}$$

where R is the range from laser source to the object, c is the speed of light in a vacuum, n is refractive index, corrects the speed of light to match the given medium, and τ is time-of-flight (Vosselman and Maas 2010) A illustration how the light pulse travels from a laser to a light detector is presented in the Figure 3.

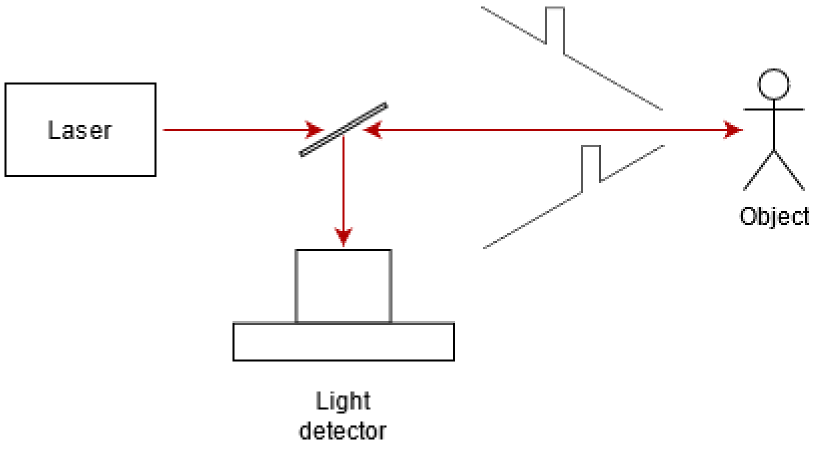


Figure 3. *Light transit time estimation principle using time-of-flight calculation.*

The FMCW idea is the same as in time-of-flight calculation; measure some difference in laser wave after reflecting from an object, see Figure 3. On the contrary, the FMCW approach differs from the time-of-flight calculation, so that in the FMCW approach, a constant laser wave is sent, whereas, in the time-of-flight approach, only short laser pulses are sent. The frequency of the constant laser wave is modified in a chosen pattern

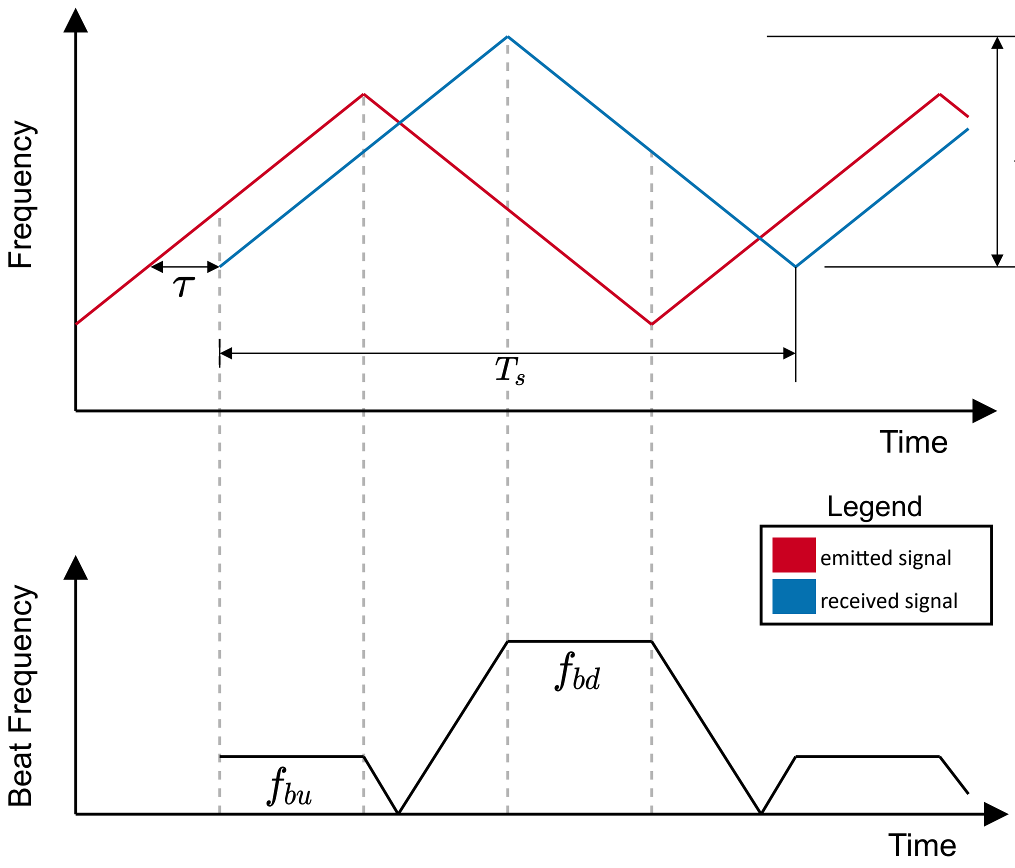


Figure 4. *FMCW system measurement principle and transmitted-received signal using triangular chirp modulation adapted from (Başarslan and Yaldiz 2017; Sabet 2016; Vosselman and Maas 2010)*

to measure the differences in beat frequency. (Vosselman and Maas 2010) The FMCW approach measuring principle using beat frequency is illustrated in Figure 4.

Figure 4 illustrates the triangular pattern in the frequency modulation, which is one of the two patterns used in the FMCW system, while the other is a sawtooth chirp pattern. The latter one is not discussed since it can not measure the velocity of an object if the object is moving (Sabet 2016), which is not beneficial in robot applications. Therefore, the following equations apply only for the triangular chirp modulation. The range R from the laser source to the object can be calculated in the FMCW approach, comes from the following equation:

$$R = \frac{cT_s}{4B}(f_{bd} + f_{bu}), \tag{6}$$

where T_s is wave period, also known as sweep time, B is sweep bandwidth, f_{bd} is beat frequency during down-ramp, and f_{bu} is beat frequency during up-ramp. (Başarslan and

Yaldiz 2017) The beat frequencies during down-ramp f_{bd} and up-ramp f_{bu} are defined as

$$f_{bu} = f_b - f_d, \tag{7}$$

$$f_{bd} = f_b + f_d, \tag{8}$$

where f_b is beat frequency and f_d is Doppler frequency (Başarslan and Yaldiz 2017)

Market research done by Lee (2019) on ten leading companies in the LiDAR industry concluded that most of the state-of-the-art commercialised products use time-of-flight calculation, one used FMCW approach, and one used Amplitude modulation (AM). In contrast, Vosselman and Maas (2010) concluded that time-of-flight systems would be limited by the point rates, under 50 000 points in a second, and the continuous wave modulation systems, FMCW and AM, would be more accurate but limited to short operating ranges, maximum of 100 meters. According to the datasheets provided by the leading companies on their products and research by Lee (2019), it can be concluded that the used technique on light transit time estimation does not anymore limit how many points a LiDAR system can observe in a given time frame. A notable difference between these two distance measurement techniques is that the FMCW approach can directly measure both the distance and velocity of an object. In contrast, in time-of-flight systems, the velocity of an object is an estimation from several laser pulses (Li and Ibanez-Guzman 2020). In the eye-safety standard IEC 60825, the power of laser transmitters is restricted to a maximum limit so that they do not cause damage to the human eye, which in contrast limits the maximum range of LiDARs. The FMCW approach uses less power to illuminate the object due to the continuous illumination. Thus, it holds better against the eye safety standards than time-of-flight systems and has more diversity in the wavelength used. (Li and Ibanez-Guzman 2020)

Passive remote sensing technologies, sensors that do not emit beams of light themselves but only measure radiation from other sources, such as the sun, depend on the surrounding environment conditions. This dependency means that these systems cannot be used during nighttime, cloudy weather or badly illuminated environments. (Carter et al. 2012) Since LiDAR systems use active remote sensing technology; this deficiency does not apply. However, one disadvantage is common for both remote sensing systems. Adverse conditions, like snow, rain, and haze, significantly affect how the light travels in the air, seen in increased transmission loss. The increment results from the laser photons being scattered around and absorbed by particles in the air. (Carter et al. 2012; Li and Ibanez-Guzman 2020)

The main features that distinguish different LiDAR sensors from each other are 1) the method how the laser beam is pointed to the observable area, 2) the method how the flight-time of the beam is calculated, and 3) on what frequency the laser beam is set (Lee 2019). In the scope of this thesis, the first method has the most impact since it affects the observable area of the 3D perception system and the frequency of how often the environment can be observed. Therefore, the discussion of different LiDAR sensors is introduced in the following sub-chapters based on the beam-steering technology.

2.2.1 Spinning LiDAR

The modern LiDAR industry was born when Velodyne sold the prototype spinning LiDARs, Velodyne HDL-64E, to other competitors in the 2007 DARPA Grand Challenge. The LiDAR showed outstanding performance in the competition, even so that five out of the top six teams had used that Velodyne LiDAR. (Lee 2019; Li and Ibanez-Guzman 2020) The core idea in spinning LiDARs is to rotate the laser sensors around many times a second to point the laser beams to every direction horizontally while at the same time changing the laser direction vertically. An illustration of the components in spinning LiDAR and the working principle is illustrated in Figure 5.

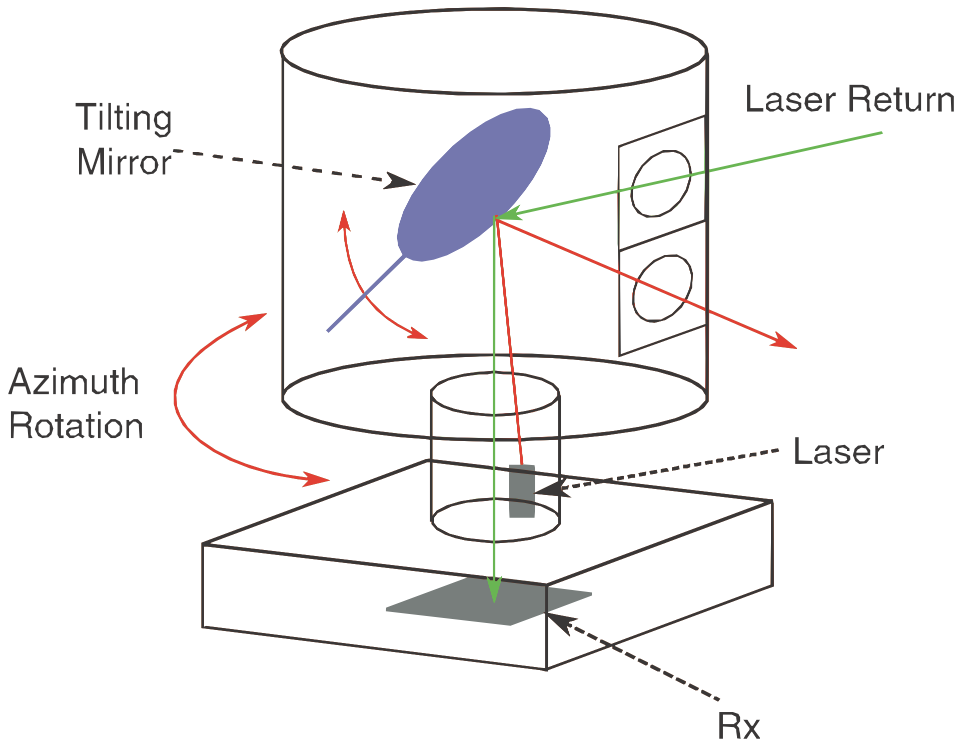


Figure 5. *Mechanical spinning LiDAR working principle illustration (Li and Ibanez-Guzman 2020). © 2020 IEEE*

The most noticeable advantage of spinning LiDARs is the 360° horizontal Field of view (FOV), which allows broad coverage of the surrounding environment, leaving horizontally no blind spots in an ideal environment. The head, on which the laser sources are attached, is rotated using typically a controllable electric motor (Li and Ibanez-Guzman 2020) In Figure 5, this rotation is shown as Azimuth Rotation. At the same time that the head is rotating around its axis, the laser beams are steered vertically using a tilting mirror, which angle is controlled by an electric motor creating the vertical FOV of the

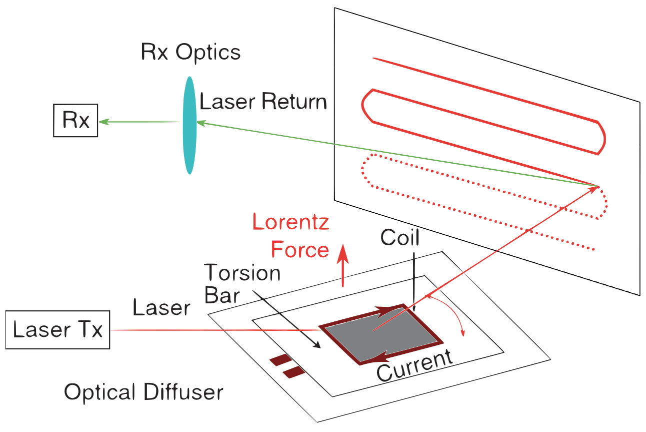
LiDAR. In Figure 5, the used mirror controlling system is an embedded nodding-mirror system. (Li and Ibanez-Guzman 2020) The laser source is located in the not rotating part of the LiDAR, as well as the laser receiver.

Although the mirror control systems provide a cost-efficient way to increase the vertical FOV, they also add one moving mechanism to the system. To reduce the probability of the mechanism to break the leading LiDARs use several laser beams to reduce moving range (Li and Ibanez-Guzman 2020). The lowest amount of laser sensors typically in a spinning LiDAR is 16; for example, Velodyne Puck models and the state-of-the-art LiDARs have even up to 128 laser sensors. The price of LiDARs goes hand in hand with how many laser sensors they have, 16 being the cheapest and 128 most expensive. (Lee 2019) There are at least two different designs of how the individual laser and detectors can be packed. In Velodyne LiDARs the lasers and detectors are individually packed, resulting in as many chips as lasers. In contrast, Ouster has designed all lasers to be packed into a single chip and receivers on a second chip. The two designs also use different lasers; the Ouster unified design uses vertical-cavity surface-emitting lasers sensors, which allow the unified design of integrating several sensors into the same chip. The Velodyne LiDARs use edge-emitting laser sensors, which need each individual chip to operate. (Lee 2019)

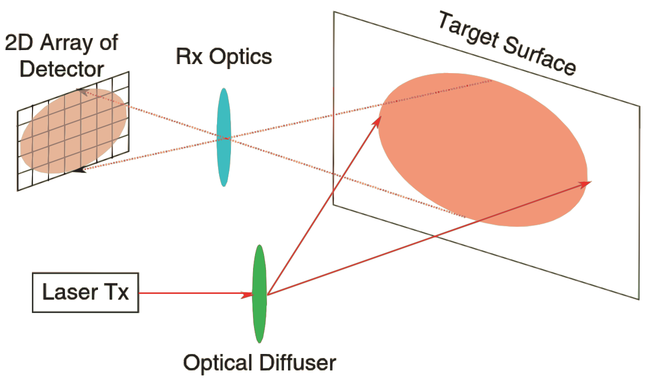
Spinning LiDARs are fragile to harsh conditions, for example vibration (Li and Ibanez-Guzman 2020), otherwise they are resistant to adverse conditions. A drawback that has been criticised among spinning LiDARs is that the sensors have a lot of moving parts, and thus, the price of the sensors is high, and the long-time reliability is not certain (Lee 2019). For example, Velodyne HDL-64E LiDAR spins the head up to 900 rpm around its axis to achieve an observation rate of 15 Hz. The long-time reliability is a problem in spinning LiDARs. (Horaud et al. 2016) Due to the structure of the spinning LiDAR, it has large inertia of the rotating head, which limits the maximum observation rate. Additionally, the rotating head uses a lot of energy, which increases the power consumption of the sensor. (Yoo et al. 2018) Spinning LiDARs tend to be quite expensive, and to see them in used widely in production might be still in the future. The leading manufacturers Velodyne and Ouster in spinning LiDARs are selling their sensors in the price category of around 10 000 US dollars, without taxes (Ouster 2021; Team 2021) Although, as the industry grows, the prices of these sensors are expected to go down and allow more opportunities to use them in commercialised products.

2.2.2 Solid-state LiDAR

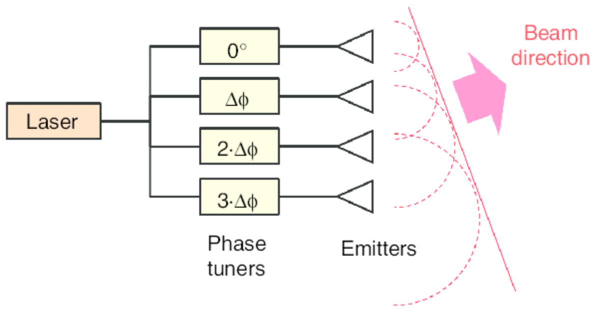
Whereas spinning LiDARs has several moving parts, the opposite, solid-state LiDARs, by definition, do not have moving parts, except they might have moving micro-mirrors (Li and Ibanez-Guzman 2020). Furthermore, in general, spinning LiDARs are bulky and have high power consumption, whereas the solid-state LiDARs are more compact and have smaller weight and power consumption (Yoo et al. 2018). The general principle in the solid-state LiDARs is that the laser source is stationary, and the path of the laser beam is controlled. Figure 6 shows different solid-state LiDARs with different laser beam steering techniques.



(a)



(b)



(c)

Figure 6. *Different solid-state LiDARs with different laser beam steering techniques. (a) Microelectromechanical systems steering LiDAR (Li and Ibanez-Guzman 2020) ©2020 IEEE, (b) Flash LiDAR (Li and Ibanez-Guzman 2020) ©2020 IEEE, and (c) Optical phased-array LiDAR (Heck 2016)*

The state-of-the-art commercialised solid-state LiDARs use three different kinds of laser beam steering techniques (Lee 2019), shown in Figure 6. The first one is Microelectromechanical systems (MEMS) steering, in which a single laser beam is steered predefined path using a mirror, which redirects the path of the laser beam. For example, company AEye uses this technology in their solid-state LiDAR. (Lee 2019; Li and Ibanez-Guzman 2020) Even though the solid-state LiDAR has a moving mirror-like spinning LiDAR, it is still categorised as solid-state LiDAR.

Figure 6 (a) illustrates one MEMS mirror controlling strategy, which uses two forces to rotate a mirror embedded on a chip an electromagnetic force, more specifically Lorentz Force, and an elastic force from a torsion bar. These forces are opposite forces of each other, allowing control in both directions. The mirror is attached to a dual-axis, allowing 2D movement. (Li and Ibanez-Guzman 2020)

The second laser beam steering technique is flash LiDAR, seen in Figure 6 (b). In this steering technique, a single laser is spread to the observable area simultaneously, similar to how cameras work. The single laser source is aimed at an optical diffuser, which spreads the laser to a plane surface simultaneously. The received laser pulses are detected with an array of photodiodes through receiving optics. (Li and Ibanez-Guzman 2020) The flash LiDAR does not have a single moving component and, thus, is more fitting to the solid-state LiDAR category than the MEMS steering LiDAR.

Then there is the last of the state-of-the-art laser beam steering techniques, Optical phased-array (OPA) LiDAR. The working principle of these LiDARs is shown in Figure 6 (c). This technique is not so common as the other steering techniques; for example, only one company, Quaenergy, has done LiDAR with this technology in market research by Lee (2019). The core idea in this technique is to change the direction of the laser beams using phase modulation. The phase tuners can change the speed of light and, when the laser is passing through a lens and by suitably adjusting the speed of light, the beam direction can be changed to steer the laser to the wanted angle. (Li and Ibanez-Guzman 2020) For example, steering the laser beam perpendicular to the emitter array, all emitters must be in sync. To steer angle to the left, in Figure 6 (c) to the top of the image, the emitters on the left side are behind in phase compared to emitters on the right side. (Lee 2019)

Achieving full 360° horizontal FOV with solid-state LiDARs is harder than with spinning LiDARs. For example, using Luminar's solid-state LiDAR, which has 120° horizontal FOV, achieving the 360° horizontal FOV, four of these sensors would need to be integrated through sensor fusion. One typical problem for flash LiDARs is related to reaching long measurement ranges. When the laser beam is passed through the optical diffuser, the light of the laser is split among all points, which can result in too low power to get a signal back from an object. (Li and Ibanez-Guzman 2020) Same range problem has been noticed in OPA LiDARs (Lee 2019). Despite solid-state LiDARs not having moving parts, they are still not immune to vibration and shocks (Druml et al. 2018), leaving them also prone to harsh conditions like the spinning LiDARs are.

2.2.3 Safety laser scanners

The safety laser scanners are 2D laser scanners designed to provide safety for humans working around or collaborating with robots and other autonomously moving machines. The scanners were initially developed to replace the pressure safety maps used in factory automation during the 1990s (Aamodt 2014), and now they have also started to become available to outdoor usage and, thus, can be considered as a potential perception sensor to mobile robots. What differs these types of sensors from the previously mentioned different types of laser scanners and the correct usage of the term safety laser scanner is that these types of sensors must meet various standards, for example, ISO standards 13849 and 62998, and also IEC standards 61496 and 61508 (Keyence 2021; SICK 2021).

The core working principle in safety laser scanners is to emit laser pulses and calculate distances to objects using time-of-flight calculation, same as earlier mentioned LiDARs, which was discussed in more detail at the beginning of chapter 2.2. To identify whether an object is too close to the sensor, the scanner defines two field zones, protective and warning (Aamodt 2014). An illustration of these zones is presented in Figure 7. The protective field is the innermost zone, illustrated in red in Figure 7; when an object is observed in this zone, the scanner initiates a safety stop, which stops all potentially harmful motion of the system. The outer zone is the warning field zone, illustrated in yellow in Figure 7. In this type of field zone, the sensor does not raise any stopping signals; instead, it only raises a warning signal. For example, a light turns on, or a warning sound is played. (Aamodt 2014)

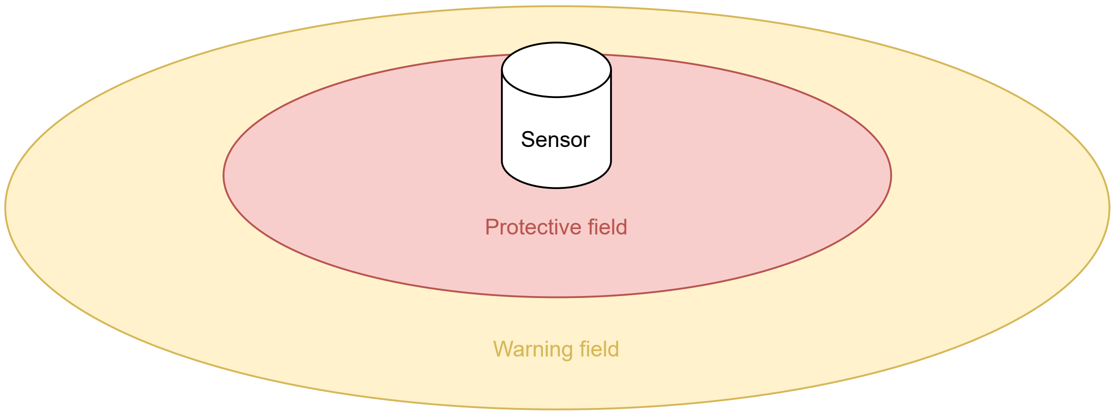


Figure 7. *Safety laser scanner field zones. The red-coloured zone is the protective field zone, and the yellow-coloured zone is the warning field zone.*

Similarly, as in spinning LiDARs, the safety laser scanners offer excellent horizontal FOV. For example, SICK outdoorScan3 has a 275° horizontal FOV. Due to the harsh conditions in outdoor applications, the safety laser scanners are also well protected against adverse conditions. Even though the safety laser scanners provide a reliable and accurate observation of the environment, the range of the protective field zone is somewhat limited. For example, the SICK outdoorScan3 has only a four-meter range in the protective field zone. (SICK 2021) The protective field range is not the same as the whole sensor range, but raising a flag in the warning zone is not guaranteed the safety of the system, and thus, the protective field range is more significant. These sensors typically use redundant signals to ensure the operation and reliability of the sensor.

2.3 Summary of perception sensors

In this chapter, three different types of perception sensors were discussed, and problems regarding them were introduced. It is not possible to shortly represent all pros and cons of these sensors in writing due to the scope of this thesis. Therefore, the following Table 1 is represented as broadly as possible different properties of 3D perception sensors and how the discussed sensors can correspond to these. The feasibility of a sensor to a property is indicated by the colour of the cell, where green is very suitable, yellow is satisfactory, and red is passable.

***Table 1.** Summary of 3D perception properties and fitment of different sensors to these. Green is very suitable, yellow is satisfactory, and red is passable.*

	Stereo Camera	Spinning LiDAR	Solid state LiDAR	Safety laser scanner
Horizontal FOV	🟨	🟩	🟨	🟩
Vertical FOV	🟩	🟨	🟨	🟥
Maximum range	🟨	🟨	🟩	🟥
Minimum range	🟨	🟩	🟩	🟩
3D point density	🟩	🟨	🟨	🟥
Weather resistance	🟨	🟨	🟨	🟩
Observation rate	🟩	🟥	🟨	🟩
Long-time durability	🟩	🟨	🟩	🟩
Harsh conditions durability	🟨	🟨	🟨	🟩
Power consumption	🟩	🟥	🟨	🟩
Observation reliability	🟨	🟨	🟨	🟩
Safety standards	🟥	🟥	🟥	🟩

Table 1, the FOVs are divided to horizontal and vertical, measured as per one sensor. The observation range is divided into minimum and maximum to emphasise the differences in extremes of the observable depth range. 3D point density refers to how many observable points the sensor can observe in a given area. Weather resistance measures how well the sensor can perform in adverse weather conditions. Observation rate is observations in a second. Long-time durability describes how long the sensor is expected to work before mechanical failures. Harsh conditions durability refers to whether the sensor can tolerate harsh conditions, for example, vibration and dust. Power consumption measures how much power the sensor uses to function. Observation reliability is related to what probability the sensor produces false positives. Safety standards mean whether the sensor obeys safety standards.

Overall, the stereo camera is pretty capable in all properties described in Table 1, but it requires good calibration of the stereo camera and additional peripherals, like protective cases. Spinning LiDAR offers mediocre performance with excellent horizontal FOV but requires much power, and the observation rate is limited. Solid-state LiDAR is a good compromise in almost all properties and has a significant maximum observation range. The safety laser scanner is the only one obeying the safety standards while also offering good durability in several categories. However, the limited maximum range makes it limited only to specific scenarios.

3. REQUIREMENTS FOR 3D PERCEPTION SYSTEM

In this chapter, the design process of a 3D perception system for heavy-duty field robots is discussed from the view of requirements for the system. The view includes the requirements the system must meet and why these requirements exist. The main areas how the requirements can be divided are 1) heavy-duty field, 2) real-time devices, and 3) embedded system requirements, each of them discussed as to their sub-chapter. At the end of this chapter is a summary of the requirements. Potential solutions to these requirements are discussed in chapter 5.2, 3D perception system design principles, where the design principles are also formed.

3.1 Heavy-duty field requirements

The scope of this thesis focuses on heavy-duty field robots, more precisely mobile robots, that have both controllable bases and manipulators. These robots are designed to be operated on construction sites, for example, in road worksites or building worksites and other demanding environments such as mines. Heavy-duty robots use hydraulic actuators to do labour-intensive and potentially dangerous actions, resulting in increased worker safety and productivity versus operators using similar machines. The robots are commonly powered by diesel engines, although the diesel engines are starting to get replaced by electric motors to cut carbon emissions as the technology advances. For example, Volvo has launched the same excavator in both electric and diesel configurations with exact specifications (Volvo 2021). A common trend is to automatise existing machines that have been operated by operators before since heavy-duty machines tend to be expensive. They also have long a life expectancy, and usually, there is much room to retrofit the equipment needed to run the machine autonomously. For example, a company called Built Robotics makes retrofit-kits to excavators to achieve autonomous trench digging (Robotics 2021) In the following Figure 8, on the left is a photograph of an excavator, which is a heavy-duty machine that can be deployed as a heavy-duty field robot. On the right is a picture of a Sandvik loader, which is a commercialised heavy-duty field robot with autonomous driving and loading (Sandvik 2021).

Mobile and industrial robots can have a macro-micro manipulation architecture. However,

Figure 8. *(a) Photograph of a heavy-duty field machine. (b) Photograph of a heavy-duty field robot.*

it is more common in mobile robots, and in this thesis, it is assumed to be applied to all mobile robots. The robot consists of two parts, macro and micro, being independently controlled in this architecture. The macro part is responsible for the major movements of the robot, typically in industrial robots, to steer the micro part close to the machinable object. When the macro part has driven the micro part close enough, it activates, and the macro part deactivates. The micro part performs precise motions, like drilling holes to the object with small tolerances. (Feng et al. 2015) In some applications, the macro and micro parts might be controlled together. The mobile robot macro-micro architecture resembles the industrial robot one with slight differences. In mobile robots, the macro part is responsible for driving the robot base close to the desired object, then the micro part, for example, a manipulator, can make the desired action to the object. An illustration of both robot types in a macro-micro manipulation architecture is presented in Figure 9.

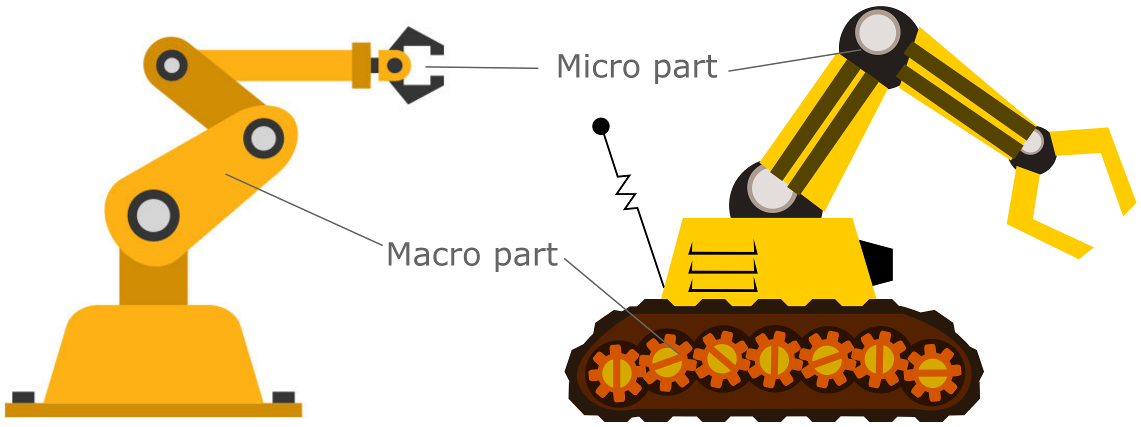


Figure 9. *Example industrial and mobile robot with macro-micro manipulation architecture adapted from (Devianart 2021; Pixabay 2021a).*

In Figure 9, on the left is an example industrial robot with macro-micro manipulation architecture. In these types of robots, the macro part is the robotic arm attached to the ground, and the micro part is attached to the end of the macro part with the desired tool. On the right is an example mobile robot with the same architecture, where the macro part is the moving base of the robot and the micro part is the robotic arm attached to the macro part.

Observation of the surrounding environment is a crucial part of a mobile robot's macro and micro scopes. The scenarios can drastically differ between these two scopes, resulting in different requirements for the perception sensor in different scopes. As an example scenario, (Wang et al. 2018) implemented a 3D perception system, using camera and laser sensors, to control a 5-DOF hydraulic manipulator to fill holes with emulsion explosives. The manipulator was attached to a heavy vehicle, but the vehicle was not controlled. The camera with laser scanners was proven suitable for this purpose, but these sensors would not be suitable if the vehicle were to be driven autonomously. The used laser sensors were just one point lasers giving depth to a point they are pointing, and there was only one camera. Comprehensively observing the surrounding environment with these sensors is problematic, leading to a partial observation of the environment and, thus, unsafe robot operation. For example, a safety laser scanner would be better suited for controlling the macro part but unsuitable for the hole filling. Given these points, choosing the perception sensor according to the manipulation scope is vital.

Generally, heavy-duty field robots are prone to vibration due to the big diesel engines powering the machines. Another vibration source is the continuous tracks in tracked robots. The vibration can be harmful to the perception sensors. In stereo cameras, the vibration can be seen in the rectified images. Due to the vibration, the rectification might not be successful and cause row miss-alignment, leading to problems finding corresponding points from the stereo image pair and resulting in false positives in the disparity image. Furthermore, the LiDARs are not immune to vibration either. The rotation mechanism of spinning LiDAR is vulnerable to vibration potentially causing damage (Li and Ibanez-Guzman 2020), and in the solid-state LiDARs the mirror controlling mechanisms without resonance are prone to vibration and shocks (Druml et al. 2018). For these reasons, the potential vibration must be considered when choosing a perception sensor to 3D perception system in a heavy-duty field robot.

3.1.1 Environmental requirements

The environment conditions where a heavy-duty field robot operates can drastically differ; the robot can be inside or outside, the weather changes according to seasons, and the weather can have rare phenomenons. The weather is not the only dynamical factor in the surrounding environment of heavy-duty field robots; for example, in construction sites, the building components are moved around, and the construction progress changes the environment by placing walls and other stationary obstacles alternating the environment. The environment that is constantly changing and evolving is called an unstructured environment. In these environments, the robots cannot operate on pre-defined paths due to obstacles moving around, and the target objects cannot always be placed in the same pre-defined location. (Feng et al. 2015) Relying on sensors that do not observe the surrounding environment, like GPS and wheel odometry, is not sufficient for safe robot navigation and, therefore, perception sensors are necessary.

The mobile robot pose, position and orientation, to the surrounding environment is typically estimated using SLAM algorithms, which are prone to position drifting. The pose of the robot, calculated by SLAM, slowly starts to differ from the ground truth. (Stachniss 2009) This can be corrected by using loop closing. In loop closing, the algorithm takes features from the surrounding environment to estimate the robot pose and uses them to correct the position drift. There are two categories how features that can be categorised: known objects and landmarks. Known objects can be pre-defined markers or coding in the environment, what the loop closing is defined to look for. (Stachniss 2009) Common marker type is an ArUco marker; it is a black-and-white square marker containing binary code, coded with the colours, enclosed with black border (Garrido-Jurado et al. 2014). A surrounding environment with markers is called a marker-based environment. The other feature type is landmark, which is previously measured position of some interesting point of the environment (Stachniss 2009). Correspondingly, an environment without markers is called a marker-less environment. The marker-based environments are problematic for heavy-duty field robots because they operate in unstructured environments, where there might not be a possibility to place markers. Even if possible, it causes more work to workers around the robot, reducing the level of autonomy, and the durability of the markers is limited in adverse conditions. (Feng et al. 2015) While the marker-less environment is a more challenging problem for SLAM and loop closing, the usage of markers should be opted out; hence, the 3D perception system should be suitable to marker-less environments.

As mentioned earlier, a 3D perception system in a heavy-duty field robot must be resistant to even the more demanding weather conditions, referred here as adverse conditions. These are conditions, which either produce or have the potential to produce harmful or interfering effects to the 3D perception system, resulting in malfunctions or disturbance in the system (Insider 2021). Adverse conditions are, for example, fog, rain, snow, shocks, dust, and loose objects. Most LiDAR sensors are protected against weather conditions. However, due to the nature of the laser beam, adverse conditions can cause disturbance in the observation of the surrounding environment. Michaud et al. (2015) study shows that several LiDAR sensors were not operating very well in snow conditions. The impact of the snowflakes caused the observation range of the LiDAR sensors to be limited to under 10 meters. One reason for the poor performance is that the snowflakes cause extra echoes of the laser beam to be returned. Their study used the Hokyo sensor, which can give two false-positive measurements from snowflakes due to the ability of the sensor to return up to three echoes. (Michaud et al. 2015) Snowstorm is not the only adverse condition where the LiDAR sensors have problems. A study by Kutila et al. (2018) concluded that LiDARs are affected by fog and rain significantly, and even up to 50 % reduction in observation range was measured. When a laser beam is steered to fog, the signal weakens a lot due to reflection, scattering, and absorption of fog particles (Kutila et al. 2018). Some adverse conditions, like shocks and loose objects, can cause mechanical damage to perception sensors in collisions. Moreover, dust and excess water can cause a short circuit due to accruing inside the casings onto the electronics. Therefore, the 3D perception system sensors should be as well as possible enclosed and, thus, protected against adverse conditions. For example, cameras are not usually well protected, whereas the LiDARs are and usually need extra casings. As noted, the 3D perception system should be resistant to adverse conditions.

3.1.2 Safety requirements.

As the heavy-duty field robots are deployed outside in an unstructured environment, the robot cannot be caged. Thus, the safety aspects of the design must be excellent to have safe interactions between heavy-duty field robots and other workers. The work sites where the heavy-duty field robots are working are dangerous, and the rate of workplace accidents is relatively high (Liang et al. 2019). U.S Bureau of Labor Statistics has reported that from all fatal accidents during 2003-2010 in road construction sites, a little over half were collisions with a vehicle or mobile equipment driven by operators. One of the significant reasons were that a worker was struck by a vehicle was because the vehicle was backing and the operator did not see the worker. (Pegula 2013) Indeed, the safety of the heavy-duty field robots must be ensured before allowing them to operate on worksites with other workers and kept as the baseline for human-robot collaboration. Hence, the importance of reliable observation of the surrounding environment using 3D perception sensors can not be stressed enough.

There are many standards related to autonomous machines, worksites, and industries that set guidelines and requirements for autonomous applications in different scenarios. The literature review by Tiusanen et al. (2020), summarises a lot of different standards, like ISO, IEC, and others, for autonomous machines. Even though the paper focuses on different industries, many of the standards are applicable to heavy-duty field robots regardless of the industry. The primary industries were mining, harbour business, earth-moving, agricultural, forestry, construction, and general industry (Tiusanen et al. 2020). The following Table 2 presents different standards along their main objectives related to heavy-duty field robots.

The scope of this thesis is not to go through each of the mentioned standards in Table 2 in detail, instead to show what standards there are related to heavy-duty field robots. General things that the standards define are risk assessment, system safety concepts, measurement and evaluation of systems, safety criteria, guidance to safe usage, performance levels for safety functions, failure scenarios, and requirements for the absence of a driver (Tiusanen et al. 2020). Figure 10, by Tiusanen et al. (2020), shows how the standards in their review, also mentioned in Table 2, are related to each other.

Table 2. *Summary of safety standards related to heavy-duty field robots and perception sensors, with their primary objectives.*

Standard	Objective
ISO 3691-4	Safety requirements for driverless industrial trucks, including automated guided vehicle, autonomous mobile robot, and unmanned forklift (ISO 2020).
ISO 10975	Safety requirements for autonomous driving systems in agriculture industry used machines (ISO 2009).
ISO 13482	Safety requirements for collaborative robots in personal care (ISO 2014).
ISO 13849	Safety requirements for safety-related parts of control systems (ISO 2015).
ISO 16001	Object detection and visibility aid systems evaluation and performance testing on earth-moving machines (ISO 2017).
ISO 17757	Safety requirements for autonomous and semi-autonomous earth-moving or mining machines on a system-level (ISO 2019a).
ISO 18646	Defines methods to measure and evaluate robotic performance criteria (ISO 2016).
ISO 18497	Design principles and safety requirements for highly automated machines and vehicles in agricultural industry (ISO 2018d).
ISO 19014-1	Determination of performance levels for earth-moving vehicles safety-related parts (ISO 2018a).
ISO/PAS 21448	Reliability evaluation for the safety of the intended functionality in road vehicles, with the option for artificial intelligent based applications (ISO 2019b).
ISO 25119	Functional safety requirements for safety-related parts of control systems in agriculture vehicles (ISO 2018b).
ISO 26262	Functional safety requirements of electrical systems for automotive industry (ISO 2018c).
IEC 60204	Safety requirements for electrical machinery in industrial (IEC 2016).

IEC 61508	Defines four safety integrity levels for automatic protection systems and guides how to apply, design, deploy and maintain them (IEC 2010).
IEC 62061	Safety requirements for safety-related control systems for machines (IEC 2021).
IEC 62267	Safety requirements in autonomous urban guided transport in high-level (IEC 2009).
IEC TS 62998-1	Safety requirements for safety-related sensors and systems in human protection (IEC 2019).

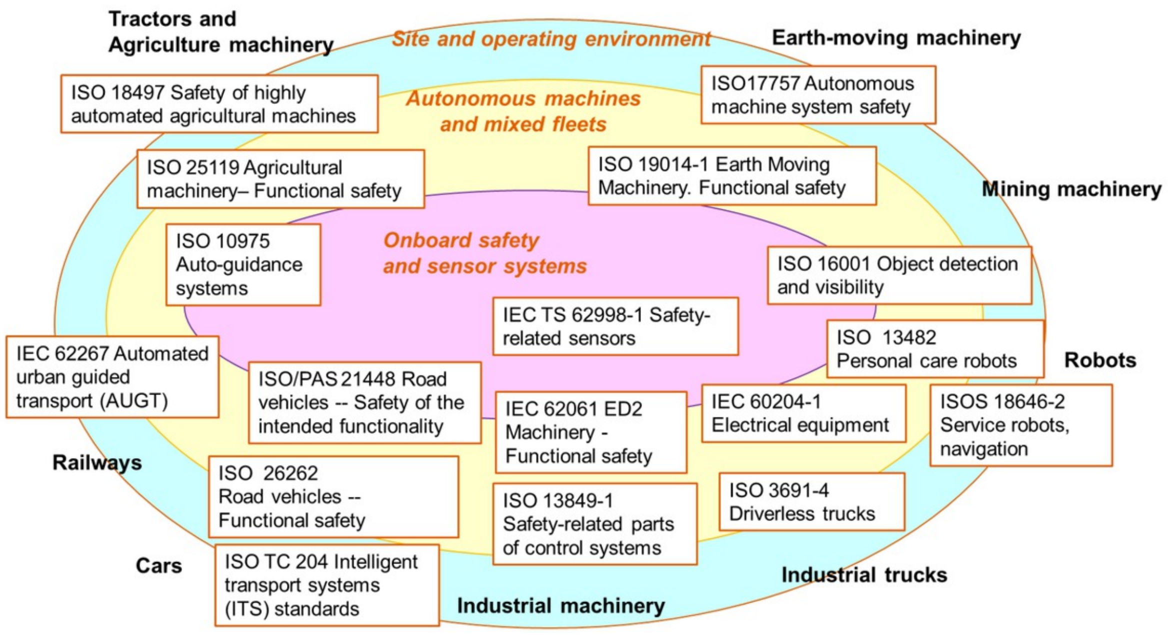

Tractors and Agriculture machinery
Site and operating environment
Earth-moving machinery
ISO 18497 Safety of highly automated agricultural machines
Autonomous machines and mixed fleets
ISO17757 Autonomous machine system safety
ISO 25119 Agricultural machinery– Functional safety
ISO 19014-1 Earth Moving Machinery. Functional safety
Mining machinery
ISO 10975 Auto-guidance systems
Onboard safety and sensor systems
ISO 16001 Object detection and visibility
IEC TS 62998-1 Safety-related sensors
ISO 13482 Personal care robots
Robots
IEC 62267 Automated urban guided transport (AUGT)
ISO/PAS 21448 Road vehicles -- Safety of the intended functionality
IEC 62061 ED2 Machinery - Functional safety
IEC 60204-1 Electrical equipment
ISOS 18646-2 Service robots, navigation
Railways
ISO 26262 Road vehicles -- Functional safety
ISO 13849-1 Safety-related parts of control systems
ISO 3691-4 Driverless trucks
Cars
ISO TC 204 Intelligent transport systems (ITS) standards
Industrial machinery
Industrial trucks

Figure 10. *Different ISO and EAC standards related to heavy-duty field robots (Tiusanen et al. 2020) ©2020 De Gruyter Open Access, under Creative Commons Attribution 4.0*

The main concerns with sensors are related to the performance levels defined in ISO-13849-1 (ISO 2015). The performance level is a discrete indication what is the probability of dangerous failure per hour for a perception sensor (Tiusanen et al. 2020). From the perception sensors described in Chapter 2, only the safety laser scanner has a good performance level. Bad performance levels have been kept as a bottleneck for the development of autonomous vehicles.

The current safety standards are pretty new, and they have been used for a short time, which was indicated by Tiusanen et al. (2020). They also mentioned two problems relating to the current standards. First, one is the gap between the requirements in standards and the state-of-the-art technology, resulting in a problem of integrating already existing designs to the safety standard requirements. The latter is more related to how the safety standards are formed, which shows how the standards are mainly for machine manufacturers and not for workers in the worksite. (Tiusanen et al. 2020) Regardless of these problems, the used 3D perception system in heavy-duty field robots should obey the obligatory safety standards and follow good practices from related standards.

The observation area is one of the fundamental properties of a 3D perception system. It defines the area of the surrounding environment that the system can observe. The observation area can be described with several features, of which the two important ones are the perceived depth and the FOV. The perceived depth means how far the system can observe from the sensor in a straight line. Each robot has a different application, and thus, the perceived depth should be fitted to meet the requirements that the application specifies. At the same time, different sensors have different depth ranges, meaning that not each sensor can fulfil the requirements. The FOV can be divided into horizontal FOV and vertical FOV, both affecting how the system can observe the surrounding environment. The horizontal FOV defines the horizontal area that the system can see, usually an angle of a cone of a circle. In chapter 2, it was shown that different perception sensors have different FOVs, resulting in different observation areas. The vertical FOV, similarly to the horizontal FOV, describes a part of an observation area but in the horizontal direction. It does not have such an impact on the ground area, but the height difference between robot and ground must be considered when choosing a perception sensor. Some of the sensors have a low vertical FOV, which can cause blind spots near the robot. The vertical FOV is not very good on spinning LiDARs, and it is typically between 30-40°, resulting in a large blind spot area close to a heavy-duty field robot, like an excavator, if LiDAR is mounted on top of the cabin or base. Figure 11 illustrates how close a spinning LiDAR can observe when it is inserted into the middle of the base of the excavator. The LiDAR is mounted 32 centimetres above the top of the back of the excavator to make use of the whole 15° vertical FOV so that the laser beams do not hit the excavator. The area close to the robot, under 5 meters, can not be observed fully. Therefore, the obstacle with a height of half a meter is entirely inside the blind spot showing how significant the vertical FOV has on the observation area.

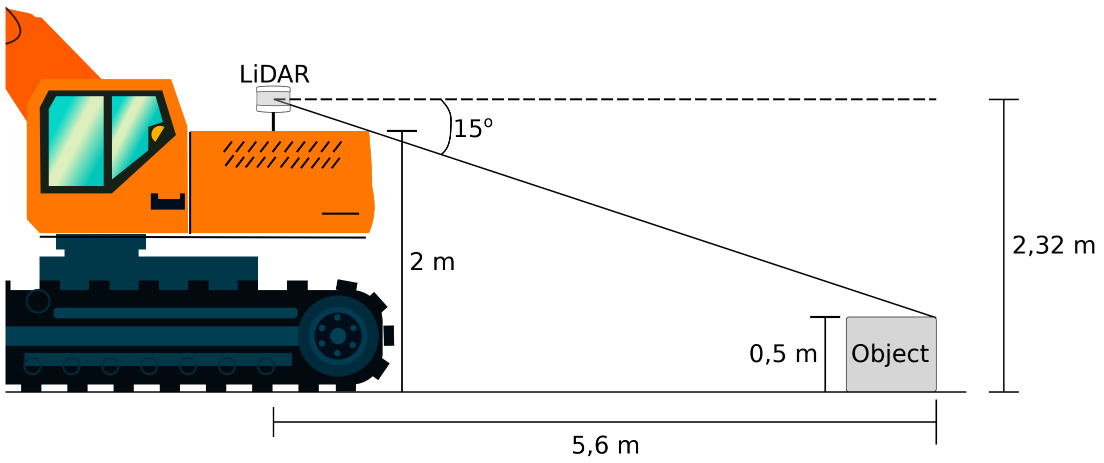


Figure 11. *Vertical observation area of a spinning LiDAR, when attached to the middle of the back of an excavator. Excavator adapted from (Pixabay 2021b). The excavator measurements adapted from (JCB 2021).*

The heavy-duty field robots tend to be big and have large bases and manipulators, causing problems in observing the surrounding environment reliably most of the time when only one perception sensor is used. For example, if there is an excavator with only one LiDAR in the back of the excavator, see Figure 11, the cabin causes a big blind spot towards the front of the excavator. The blind spots are an enormous concern for the safety of the workplace. It has been noticed that blind spots around equipment cause a lot of accidents (Liang et al. 2019). Given the concern about blind spots forming and too narrow observable areas, the 3D perception system should take these into consideration.

3.2 Real-time requirements

The amount of data that a 3D perception system has to process every time an observation about the surrounding environment is made is enormous. In addition, to get a reliable and valuable continuous perception, the system should process the data in a couple of tens of milliseconds, introducing many challenges in the computational and software capability alongside data transferring limits. The term *real-time* is from here on defined as a soft real-time, meaning that the calculation is not done on a guaranteed time and being late for the deadline is not catastrophic. Another property of real-time is that the 3D perception system can process each observation fast enough that the observation rate of the system is at least 5 Hz. Therefore, usually, either special hardware or software optimisation is needed to get real-time perception systems, excluding those applications, where the robot speed is the limiting factor and faster processing time does not benefit anyway (Davies 2012).

The first strategy to improve the data processing rates is the earlier mentioned special hardware. Previously, some exciting strategies to get faster processors have been used, like gallium arsenide semiconductor devices, Josephson junction devices, and optical processing elements (Davies 2012). More common special hardware used to improve the data throughput are presented in Table 3, reprinted from (Davies 2012). It is not a comprehensive table but a good reference for different devices for perception systems. The speed properties in Table 3 are a little outdated but still show the relationship between different hardware and how they compare in speed. The devices in Table 3 can be categorised as reasonable and flexible for developing the perception system and good and inflexible for the commercialised perception systems. PC, Field Programmable Gate Array (FPGA), and General Processing Unit (GPU) can be included in the first category suitable for the development since they are easily reprogrammed and have a fast development time. The other category, for commercialised systems, belongs to the following devices: MP, Application-specific integrated circuit (ASIC), VLSI chip, and GPU. All of these, except the GPU, need a long development time to get the best performance out of them and, therefore, belong to the second category. In conclusion, GPUs are the state-of-the-art hardware devices for perception systems currently.

Table 3. *Hardware devices used in perception systems adapted from (Davies 2012)*

Device	**Function**	**Summary of Properties**
PC	*Personal computer or more powerful workstation*: complete computer with RAM, hard disk and other peripheral devices. Would need an embedded (restricted) operating system in a real-time application.	• Fast • Extremely flexible • Should be envisioned as a software device
MP	*Microprocessor*: single chip device containing CPU + cache RAM. The core element of a PC.	• Fast • Extremely flexible • Should be envisioned as a software device

Table 3 – continued from previous page

Device	**Function**	**Summary of Properties**
FPGA	*Field programmable gate array*: random logic gate array with programmable linkages; may even be dynamically reprogrammable within the application. The latest devices have flip-flops and higher level functions already made up on chip, ready for linking in; some such devices even have one or more MPs on board.	• Fast • Extremely flexible • Should be envisioned as a hardware device, commonly slaved to a Digital Signal Processor (DSP) • Can be a software device if controlled by on-chip MPs
ASIC	*Application-specific integrated circuit*: contains devices such as Fourier transforms, or a variety of specific SP or vision functions. Normally slave to a MP or DSP.	• Very fast • Inflexible (flexibility sacrificed for speed) • Should be envisioned as a slave software device
VLSI	*Custom VLSI chip*: this commonly has many components from gate level upwards frozen into a fixed circuit with a particular functional application in mind. (Note, however, that the generic name includes MPs, DSPs, although we shall ignore this possibility here.)	• Fast • Inflexible • Should be envisioned as a hardware device
GPU	Graphics processing unit on a PC or other workstation. Although designed for computer games and other graphics applications, GPUs are able to provide valuable functionality for computer vision.	• Very fast • Substantial power requirements

Reprinted from Computer and Machine Vision: Theory, Algorithms, Practicalities, 4th edition, E.R. Davies, Chapter 26 - Real-Time Hardware and Systems Design Considerations, pp. 756-757, Copyright (2012), with permission from Elsevier.

In the last century, GPU technology has advanced in significant steps and using GPUs in scientific calculation has superseded application-specific hardware mentioned earlier. The strategy using GPUs to improve the calculation speed is massive parallelisation that GPUs are capable of because of the vast amount of calculation cores. In parallelisation, the idea is to invoke as many processors as possible to work on the same task simultaneously. In contrast, in special hardware, the one processor tried to do the same tasks in sequential order as fast as possible (Davies 2012). The requirement to parallelise a calculation task is that the output of one processor can not depend on the output of the other processors running simultaneously. Typical perception system data types suitable for parallelisation are images and point clouds. The split of a calculation task can be done either by splitting the task into smaller tasks that can be run at several processors, called algorithmic parallelism, or splitting the data into smaller parts that can be run at several processors, called data parallelism (Davies 2012). Later one is more common in perception systems, which has led to the focus on increasing the data processing throughput, having moved from special hardware to special software.

There is not only one way to do certain things, like stereo matching and point cloud filtering, and thus, there are many different algorithms for the same perception system step. Each of the algorithms has advantages and disadvantages, and one should be able to decide between these. Davies (2012) presented a list of optimisation parameters for system optimisation aimed to help researchers to compare algorithms, which are :

1. Sensitivity;
2. Accuracy;
3. Robustness;
4. Adaptability;
5. Reliablity;
6. Speed;
7. Cost.

The previous listing is not precise, and not always different algorithms can not be distinguished from each other easily. However, the idea is to guide researchers on comparing algorithms and choosing the correct one. The literature review, by Hernandez-Juarez et al. (2016), showed that there exist considerable differences in the speed of different algorithms calculating the same Semi-global stereo matching (SGM). The implementation in the same paper showed significant improvement in the calculation time due to parallelisation and software optimisations. The common shortcoming of many perception algorithms is that they are tested on ideal datasets during development, and the critical review is left out. For example, a vision algorithm should tackle the following problems:

1. Noise;
2. Background clutter;
3. Occlusions;

4. Object defects and breakages;
5. Optical and perspective distortions;
6. Nonuniform lighting and its consequences;
7. Effects of stray light, shadows, and glints. (Davies 2012)

Given these points, the algorithm in the 3D perception system should be chosen based on the system optimisation criteria and critically reviewed against non-ideal data features.

The current trend is to use GPUs as the accelerator in 3D perception systems to achieve real-time, alongside special software to parallelise all potential calculation steps. The modern GPUs must be used together with a more traditional Central Processing Unit (CPU), which has led to a new branch of programming, called GPU programming. It is a technology for programmers to write code specifically run on the GPU with massive parallelisation. GPU programming is hardware-dependant, meaning that different GPU manufacturers have specified different coding languages and Application Programming Interfaces (APIs) for programmers to use with their devices. At the moment, there are three computing platforms to use with GPUs: Compute Unified Device Architecture (CUDA), OpenCL, and OpenACC. (Morelo 2017) CUDA is the first one invented of the three and is only available to GPUs manufactured by Nvidia. OpenCL is an open and royalty-free standard supported by several different GPU manufacturers (Cook 2012), most noticeably the GPUs manufactured by AMD are supporting this standard. OpenACC is a newer platform released in 2015, with a mission to be a widely portable platform for as many machine configurations as possible, with different operating systems and devices CPUs and GPUs (Morelo 2017).

Nvidia has been one of the leading GPU manufacturers for the last century and even market leader recently in PC GPU sales (Mujtaba 2021). In addition, Nvidia launched the CUDA GPU programming platform already in the year 2007 (Cook 2012), which has led to the massive popularity of using CUDA as the GPU programming platform for researchers and engineers. Nvidia has also launched a broad range of embedded devices with capable GPUs, Jetson embedded devices, increasing the popularity of CUDA even more. Therefore, CUDA is the chosen computing platform to be discussed in more detail.

The objective of CUDA during its launch was to allow programmers to code GPUs without the need to learn shader languages. CUDA is based on the C language and can be thought of as an extension to C with an option to run the code on a GPU. More specifically, in the development phase, the coder can target the code to run either on the host CPU or the device GPU. (Cook 2012) As the CUDA is only specified for Nvidia GPUs, the working principle of them is gone through next.

The main architecture of GPUs is that they contain several streaming multiprocessors, similar to CPU cores. Then each streaming multiprocessor contains a certain amount of

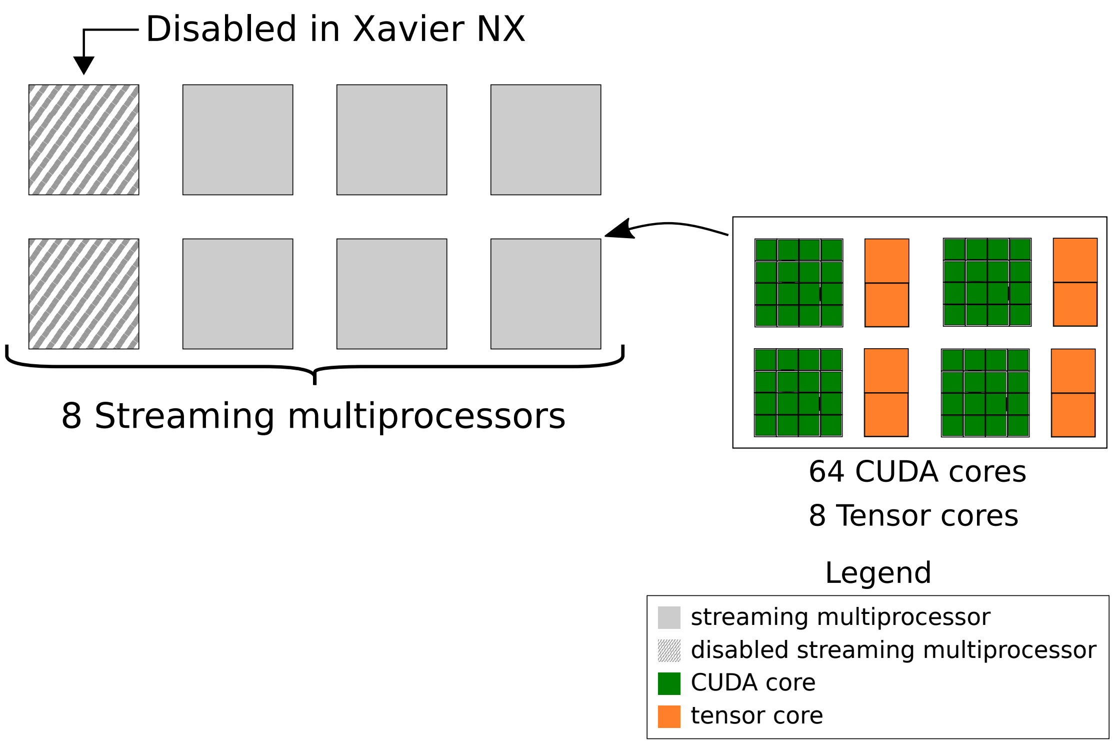


Figure 12. *GV10B GPU architecture in Jetson Xavier NX, with two disabled streaming processors.*

streaming processors. In Nvidia GPUs they are called CUDA cores, and there are eight or more of them in one processor. The Nvidia GPUs also have Tensor cores. (Cook 2012) CUDA core is a parallel processor, which performs one single value precision multiply-accumulate operation every GPU clock cycle (GamerNexus 2021). Tensor core is a more advanced version of CUDA core capable of doing one matrix multiply-accumulate operation every GPU clock cycle, which gives Tensor cores more computational capability than CUDA cores in deep learning and artificial intelligence applications, which are matrix-based calculations. The Jetson Xavier NX GPU architecture is presented in Figure 12. It is based on the GV10B GPU, which is the same one as in Jetson Xavier AGX. The difference in GPUs between these two Jetson models is that in Xavier NX, two of the eight streaming processors are disabled to reach the target CUDA core count set by Nvidia. It has six streaming multiprocessors each containing 64 CUDA cores and eight tensor cores in 4 texture mapping units, resulting in 384 CUDA and 48 tensor cores. (Techpowerup 2021)

The CUDA cores work in parallel to each other, and that is how GPUs achieve the speed-up compared to CPUs. Both CUDA and Tensor cores are calculation cores, meaning that they can not fetch or decode instructions (Ravi 2020). Therefore, as mentioned earlier, a GPU must always be paired with CPU giving the instructions to GPU cores. In earlier, the Nvidia GPU architecture was discussed. The parallelisation of a calculation step and how the individual threads are divided into CUDA cores are gone through. Firstly, a thread is the core principle of parallel computation. It is a sequence of single tasks to be calculated one at a time. Threads can have tasks from one to several tens of tasks. Threads run inside a kernel, which in GPU computing terms is a function, meaning that each thread is identical and runs the same computational steps as the others, but on the different data part of the dataset. (Cook 2012) Individual threads are contained to thread blocks before instructed to streaming multiprocessors on the GPU. The thread blocks are not individually sent to the GPU; instead, the thread blocks are formatted to a grid of thread blocks. (Cook 2012) This is done by the programmer in the code, and the importance of the shape of the grid is discussed later. An example of how threads can be divided into a grid of thread blocks is shown in Figure 13. Numbers represent the thread index starting from 0 and going to 255, totalling 256 threads. In that example, 256 threads are divided into eight blocks, each containing 64 threads and forming a grid of thread blocks. The maximum amount of threads per block is limited, depending on the CUDA compute capability (Cook 2012).

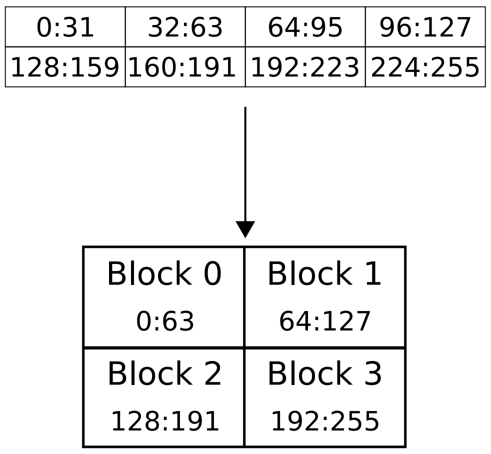


Figure 13. *Example kernel splitting 256 threads to 8 thread blocks, each containing 64 threads, and together forming a grid of thread blocks.*

The grid of thread blocks is divided equally to all available streaming multiprocessors, called block scheduling. The block scheduling strategy has not been confirmed by Nvidia. However, the GPU tries to achieve even usage of all streaming multiprocessors meaning that the blocks are probably assigned in ascending number order and assigned one at a time to a streaming multiprocessor, then the next one to the following streaming multiprocessor and so on. (Cook 2012) Resulting that each streaming multiprocessor is scheduled with the same amount of thread blocks if the total thread block count is divisible by the number of streaming multiprocessors. The amount of thread blocks a single multiprocessor can be scheduled with is limited, and it varies between different CUDA compute capabilities (Cook 2012). Continuing the example kernel process from Figure 13, the following Figure 14 shows how a previously formed grid of thread blocks is divided into streaming multiprocessors. In total, four thread blocks are divided into two streaming multiprocessors, giving each two blocks of threads to execute on their CUDA cores.

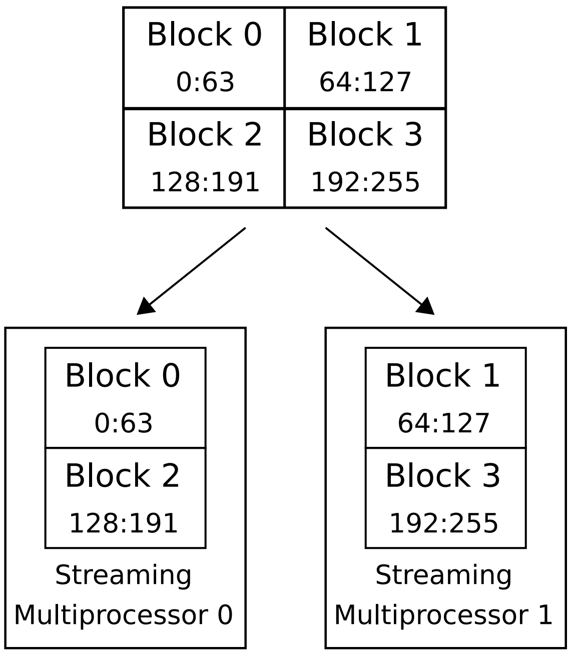


Figure 14. *Example kernel scheduling a grid of thread blocks to two streaming multiprocessors.*

The basic unit of execution on the GPU are warps. Warp is a group of threads that are executed at the same time, one warp contains 32 threads, but Nvidia might change it in the future. Because GPU uses warps, when assigning threads to CUDA cores, the thread blocks must be divided into threads. One warp must be finished execution before the next warp can be loaded to CUDA cores. (Cook 2012) As previously, Figure 15 continues the example kernel operation showing how the scheduled thread blocks are divided into warps of threads. Each thread block is divided into two warps containing 32 threads in this example.

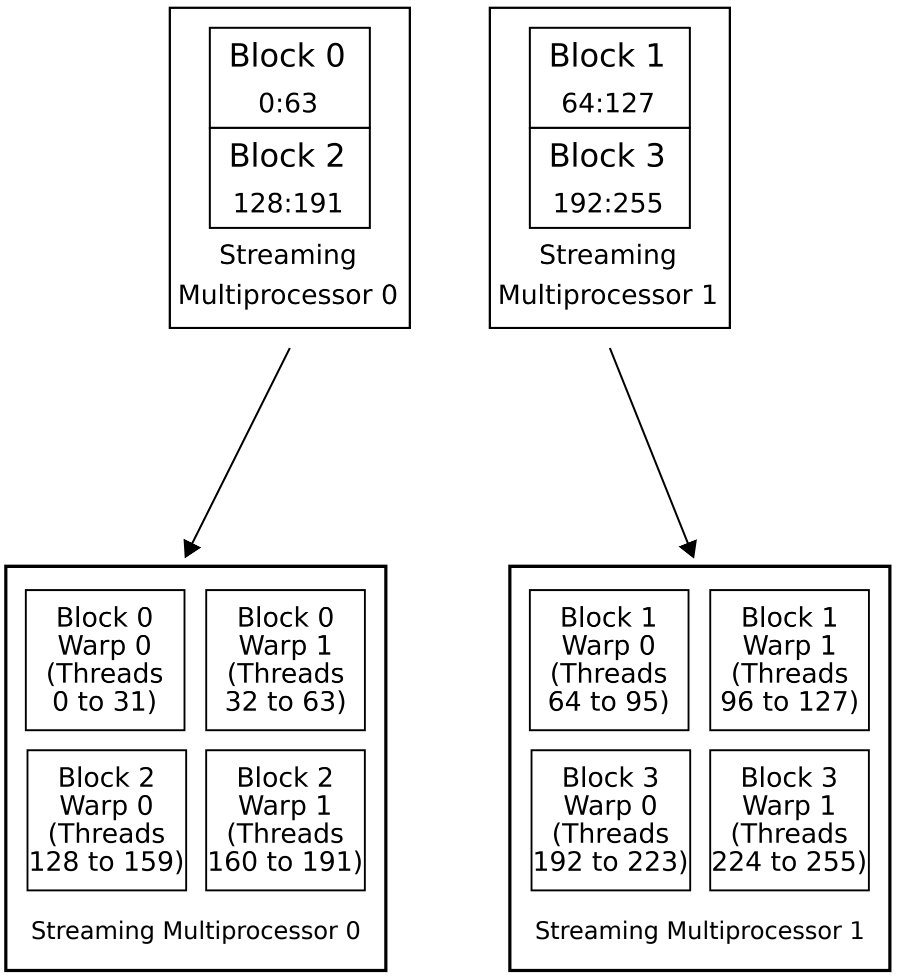

Block 0
0:63
Block 2
128:191
Streaming Multiprocessor 0
Block 1
64:127
Block 3
192:255
Streaming Multiprocessor 1
Block 0 Warp 0 (Threads 0 to 31)
Block 0 Warp 1 (Threads 32 to 63)
Block 2 Warp 0 (Threads 128 to 159)
Block 2 Warp 1 (Threads 160 to 191)
Streaming Multiprocessor 0
Block 1 Warp 0 (Threads 64 to 95)
Block 1 Warp 1 (Threads 96 to 127)
Block 3 Warp 0 (Threads 192 to 223)
Block 3 Warp 1 (Threads 224 to 255)
Streaming Multiprocessor 1

Figure 15. *Example kernel dividing thread blocks to warps of threads to execute them on CUDA cores.*

The streaming processors have warp schedulers; the amount depends on the CUDA compute capability that assigns the warps to be executed on the CUDA cores in an arbitrary order. One warp scheduler can schedule one warp in one scheduling cycle on available CUDA cores. The executed warp then writes the result to the memory and frees the resources for the next warp. (Cook 2012) Figure 16 continues the example kernel process and shows how the warp schedulers assign the warps to be executed on the CUDA cores on two scheduling cycles. This example assumes that it only takes one scheduling cycle time for a warp to execute. Both active warps in both streaming multiprocessors take the same amount of time to execute, and there are no other delays, like instruction fetching. The active warp is inside the rectangle representing 32 CUDA cores, and the warps to be executed are waiting inside the rectangle captioned as a streaming multiprocessor. Finished warps are not illustrated.

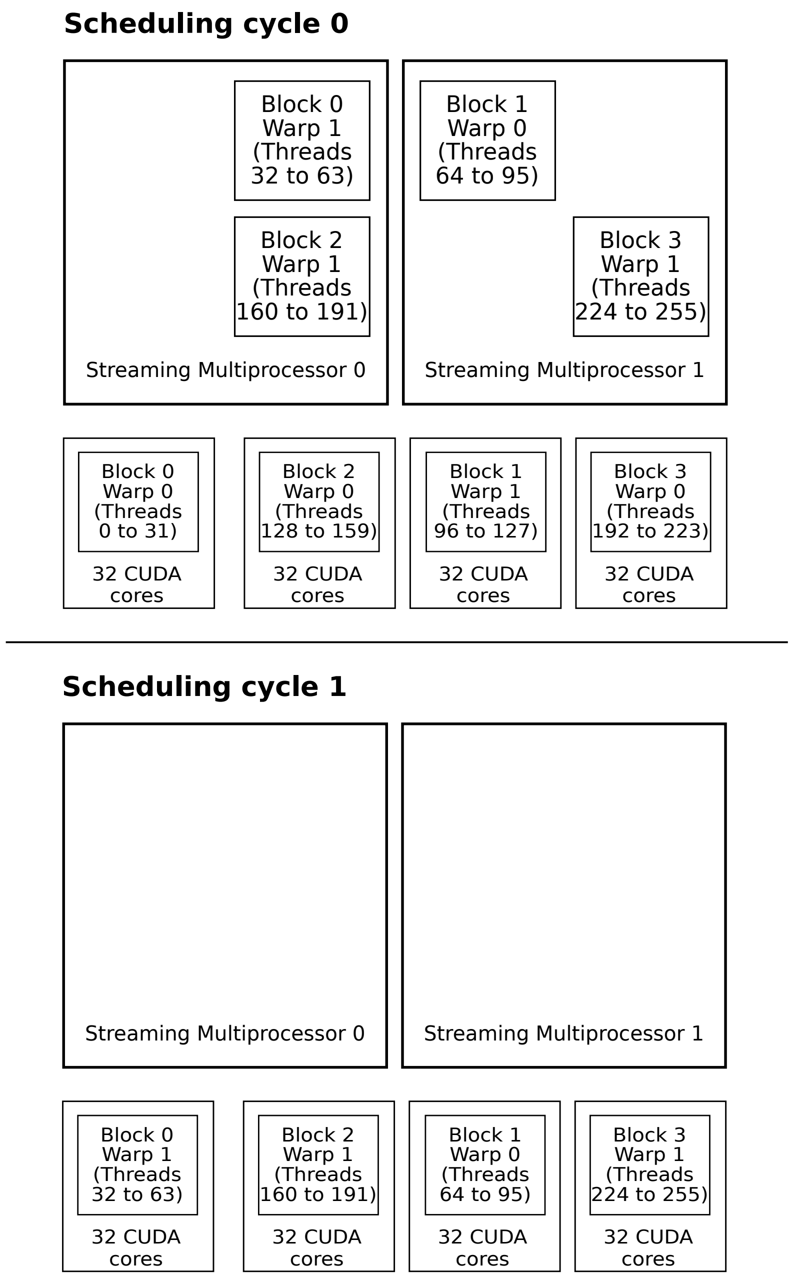

Scheduling cycle 0
Block 0 Warp 1 (Threads 32 to 63)
Block 2 Warp 1 (Threads 160 to 191)
Streaming Multiprocessor 0
Block 1 Warp 0 (Threads 64 to 95)
Block 3 Warp 1 (Threads 224 to 255)
Streaming Multiprocessor 1
Block 0 Warp 0 (Threads 0 to 31)
32 CUDA cores
Block 2 Warp 0 (Threads 128 to 159)
32 CUDA cores
Block 1 Warp 1 (Threads 96 to 127)
32 CUDA cores
Block 3 Warp 0 (Threads 192 to 223)
32 CUDA cores
Scheduling cycle 1
Streaming Multiprocessor 0
Streaming Multiprocessor 1
Block 0 Warp 1 (Threads 32 to 63)
32 CUDA cores
Block 2 Warp 1 (Threads 160 to 191)
32 CUDA cores
Block 1 Warp 0 (Threads 64 to 95)
32 CUDA cores
Block 3 Warp 1 (Threads 224 to 255)
32 CUDA cores

Figure 16. *Example kernel all 256 threads calculated in two scheduling cycles using two streaming multiprocessors.*

Several minor details exist that the programmer needs to be acknowledged to get the best performance out of the CUDA computation platform. The first one is related to the memories of the devices. The memory accessible to CPU and GPU are not unified, resulting in that GPU cannot access parameters on the memory of the CPU. (Cook 2012) The particular memory problem is solved by copying the needed data from the CPU to GPU and vice versa, but it creates overhead to the system. Excess amount of data copying should be avoided to achieve better performance. As explained earlier and shown in Figure 13, the threads to be calculated are arranged to a grid of thread blocks, which size is specified by the programmer. The size is specified as the width of the grid in threads and the height of the grid in rows of thread blocks. Suppose the programmer does not map the grid correctly according to the available streaming multiprocessors with specified CUDA cores and the total thread count. In that case, the kernel is not achieving full potential, called the poor memory coalescing problem, which can lead to an order of a five times increase in the execution time. (Cook 2012) The important thing is that there should be no deficiency in the warp sizes, meaning that each warp should have 32 threads, a current warp size that might change in the future. If the grid width is not multiple of the warp size, the CUDA pads the width of the grid to one larger multiple of the warp size, or the warps are not full, both slowing down the execution time. (Cook 2012) The shape of the grid of thread blocks is also one aspect of the design that can cause a performance drop in the kernel. The grid of thread blocks is scheduled to streaming multiprocessors starting from the top left corner of the grid, going row by row from left to right, scheduling one block to one streaming multiprocessor, as seen in Figure 14. Now, if the grid of thread blocks width is not multiple of the streaming multiprocessor count, the result is that one of the streaming multiprocessors is scheduled with more thread blocks than the others. Since the GPU is using warp-based execution, leaving more warps to one streaming multiprocessor than the others, resulting that in the last scheduling cycle, only one of the streaming multiprocessors is working, and the others are idle, resulting in performance downgrade. (Cook 2012)

As seen earlier, the architectural differences between GPUs and CPUs are pretty significant. Moreover, the CUDA coding is not as straightforward as traditional CPU coding. In summary, CPUs excel at running fewer complex tasks in lesser parallelisation, and GPUs excel at running a huge amount of simple tasks at the same time using massive parallelisation. (Cook 2012) When designing a 3D perception system, the developer should know the differences between these two architectures and decide which one suits better the calculation step to take full advantage of the available hardware, as the goal is to achieve a real-time system. Usually, many tasks in a 3D perception system depend on each other, allowing only sequential execution.

3.3 Embedded system requirements

A 3D perception system can be categorised as an embedded system. An embedded system is a product containing information processing systems, such as microprocessors embedded into the product. Together, they from an enclosed product. (Marwedel 2021) The term embedded system is still comprehensive, and a 3D perception system can also be categorised as a cyber-physical system, which is an extension for embedded system term. A Cyber-physical system utilises computational and physical components to integrate both processes into one system. (Marwedel 2021) How this relates to the design of a 3D perception system in heavy-duty field robots is the common characteristics, challenges, and possibilities for both general embedded systems and 3D perception systems. This chapter focuses on the common ground between these two system types and introduces some insight into what products can be found from the market.

First, the requirements for both systems is discussed. One of the essential requirements is that the systems should be dependable, which can be defined that the system has a low probability of failing and that if it fails, it does not cause any harm. If the system fails, it has an impact on the surrounding environment since they are connected. (Marwedel 2021) This impact can be harmful, and therefore, they must be dependable. Dependability is related to many aspects of a system that are: security, confidentiality, safety, reliability, repairability, and availability (Marwedel 2021). The second requirement is that both systems should use resources efficiently. Two important aspects of resource efficiency are energy consumption and run-time resources. The energy consumption affects how long the system can be operated if the power comes from a battery. Another aspect of energy consumption is that the reliability of electrical circuits is lowered when the temperatures have increased (Marwedel 2021). Power draw always creates heat, which warms the circuits, potentially leading to problems. Being resource efficient also includes that the system should use all of the available computational power, meaning that optimisations to the system should be done on both software and hardware level (Marwedel 2021).

One of the challenges with 3D perception systems is that the amount of data run through the system every second is high, which leads to a few problems with the data amount. First of all, if all of the data must be transferred somewhere else, the transfer data limit might be too low. For example, two cameras running full HD resolution, 1920 by 1080 pixels, in 24 Frames per second (FPS), each pixel represented as 8 bytes is around one Gigabit per second, which means that the technique to transfer the data has to be up-to-date on the current technology. The problem comes when wanting to get either more pixels or more FPS from the stereo camera, resulting in that the standard Gigabit Ethernet is no anymore sufficient. The second aspect of the enormous data amount is that it must be stored somewhere if one wants to analyse it afterwards. This means that the processor and the drive, which the data is chosen to be written, must be supporting the data throughput. The same stereo setup as earlier, run on ROS using ROS messages generates about 6 Gigabits of data in one minute. Given these points, the data transferring and storing of the 3D perception system should be fitted according to the chosen sensors.

3.4 Summary of requirements

The requirements for 3D perception systems in heavy-duty field robots were discussed from three perspectives: heavy-duty field, real-time, and embedded system. Overall, there are many requirements to be met or guidelines that should be followed. Naturally, each of them cannot always be fulfilled and therefore, trade-offs play an essential role when designing 3D perception systems. All the requirements presented in this chapter are gathered in Table 4. Potential solutions to some of these problems are discussed in chapter 5.

Table 4. *Summary of 3D perception system requirements.*

Index	Requirement
1	The 3D perception system should be designed to fit the correct manipulation scope.
2	The 3D perception system should be robust to vibration.
3	The 3D perception system should be suitable to marker-less environments.
4	The 3D perception system should be resistant to adverse conditions.
5	The 3D perception system should obey the obligatory safety standards and follow the good practices from related standards.
6	The 3D perception system should be placed to get best possible observation area and the least amount of blind spots.
7	The algorithms in 3D perception system should be chosen based on the system optimization criteria and be critically reviewed against non-ideal data features.
8	The calculation steps in the 3D perception system should be fit to the correct calculation architecture.
9	The 3D perception system should be dependable.
10	The 3D perception system should be energy efficient.

4. IMPLEMENTATION

In this chapter, the implementation of the 3D perception system is discussed in detail. The chosen methods are described, and some alternatives are also discussed. The chosen methods consist primarily of open-source libraries compatible with the default ROS installation. The chapter goes first through the system architecture and then the workflow of capturing raw images to outputting filtered point cloud in the same logical order as implemented in the source code.

4.1 Overall Architecture

This chapter discusses the architecture of the implemented 3D perception system. The architecture is divided into hardware and software architectures, and they both have sub-chapters.

4.1.1 Components

The main components of the implemented 3D perception system are the stereo camera and Jetson Xavier NX embedded PC. Specifications of both components are presented next.

Basler acA1920-50gc GigE camera

The stereo camera setup (see Figure 17) consists of two Basler acA1920-50gc cameras mounted parallel to a steel frame with 37 cm as the baseline. To capture an image with both cameras simultaneously, the cameras are connected to each other with a General purpose I/O (GPIO) cable. The left camera sends a signal through the opto-line in the GPIO cable to the right camera, indicating when the right camera should start the exposure. The left camera sends the signal when it starts the exposure. The synchronisation of the camera exposure through a GPIO cable is called hardware triggering. The cameras are powered using standard 12V DC power, using a Hirose 6-pin connector on the cameras with GPIO cable, which at the same time transfers the hardware trigger through the opto-line. The images are transferred using Ethernet to the Jetson Xavier NX embedded PC. To have enough bandwidth on the Ethernet network, only cables passing the Cat6 standards are used.

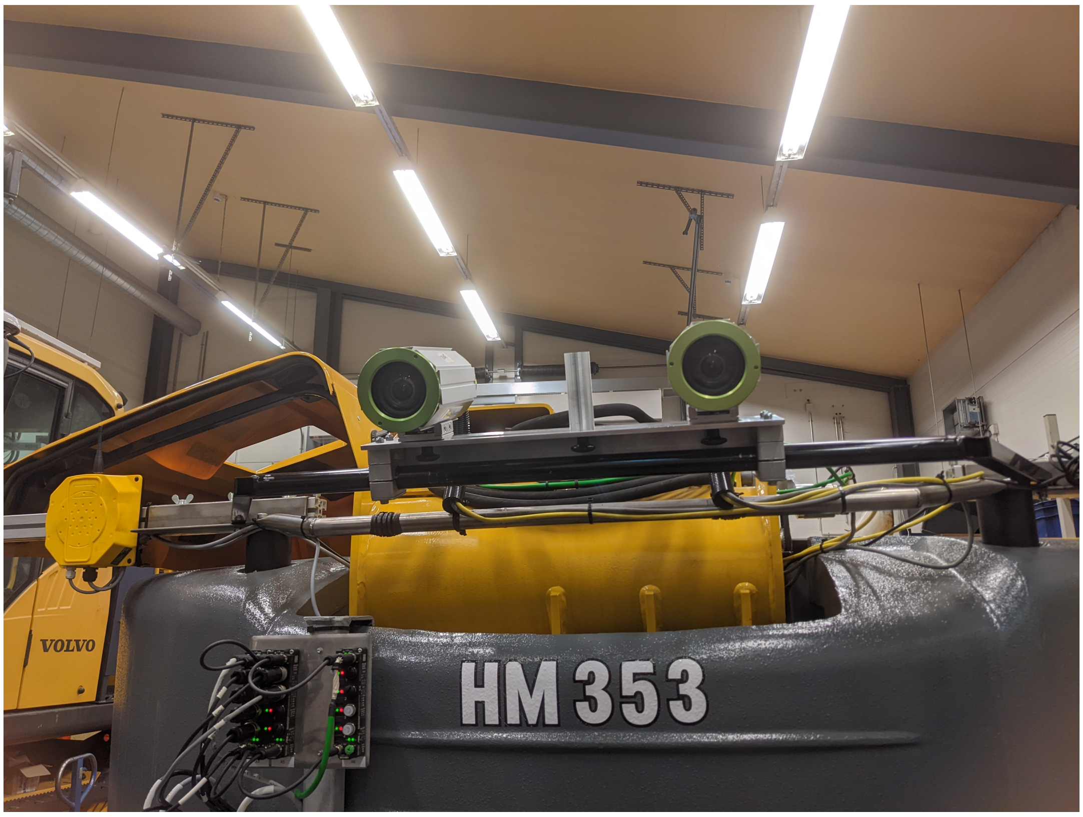


Figure 17. *Photograph of the Basler stereo cameras inside casings in stereo setup mounted onto a steel frame.*

Both cameras are equipped with VS Technology VS-0618H1 lenses. The lenses have manually-adjustable iris range and focus. The specifications of a Basler camera with a VS Technology lens is described in Table 5.

Table 5. *Individual Basler camera specifications.*

	Basler acA1920-50gc with VS-0618H1 lens
Focal Length (mm)	6
Iris Range (F)	F1.8-16
Horizontal Field of View (°)	94.8
Vertical Field of View (°)	77.9
Diagonal Field of View (°)	107
Tv distortion (1")	-0.19%
Maximum Image Resolution (pixels)	1920 (Height) x 1200 (Width)

Table 5 – continued from previous page	
	Basler acA1920-50gc with VS-0618H1 lens
Maximum Frame Rate (FPS)	50
Camera Resolution (MP)	2.3

The camera parameters are stored in two different memory locations, on the memory of the camera and the memory of the Jetson Xavier NX. Both cameras have their configuration set containing parameters stored locally in the individual cameras. The memory architecture of an individual Basler camera setup is shown in Figure 18.

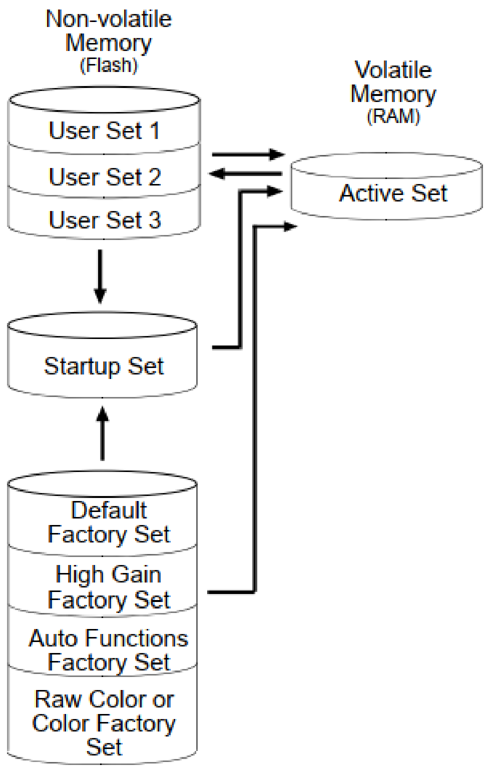


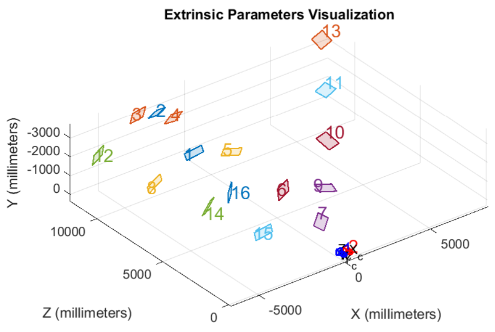
Figure 18. *Basler camera memory architecture, adapted from Basler (2016)*

Each camera has three configurable User sets, which each store one configuration set. The User sets are in the non-volatile memory, which is like a hard drive, meaning that the parameters are not lost if the camera is turned off. Unlike the non-volatile memory, the volatile memory is like random access memory (RAM), meaning that the parameters are stored there are lost every time the camera is turned off. (Basler 2016) Basler offers the option of loading a configuration set to volatile memory every time the camera is turned on, called the Startup set. To use the saved parameters in one of the User sets, it can be set up as a Startup set, resulting in the camera loading the same saved parameters every time. The User sets are configured through the Pylon software, which is only available through a Graphical user interface (GUI). (Basler 2016)

The config file is the other place where some of the parameters are stored. There are also some overlapping parameters in both places. The overlapping parameters get their value from the config file, which loads the parameters to the camera after the User set has been loaded. The Startup set is configured in the config-file, discussed in the earlier section. Both cameras have their own config-file, which uses the YAML format. The parameters from the config-file are loaded during the startup of the ROS node driving the Basler camera from the Pylon ROS driver.

The cameras must be calibrated to calculate the disparity image from the stereo camera setup and a point cloud from that. The calibration of the cameras is done with the same checkerboard as shown in Figure 2. The checkerboard consists of black and white squares of the same size, with each side length being 80 mm. The longer side of the checkerboard consists of 13 squares and is 133 cm in length. The shorter side consists of 10 squares and measures 100 cm in length. The checkerboard must be big enough to allow a good calibration of the cameras when the perceived depth is several tens of meters. With a small checkerboard, it would be impossible to recognise images far away from the checkerboard, which would result in unreliable calibration.

The software used for the calibration of the cameras is MATLAB's Stereo Camera Calibrator App. For the calibration, some calibration data must be gathered. This data consists of raw images containing the checkerboard mentioned above. The checkerboard should be positioned in many different parts of the image, with variations on as many depths and angles as possible to get a good calibration. There is no minimum or a maximum number of image pairs to be fulfilled to get a calibration. However, the MATLAB calibration guide mentions that, as a rule of thumb, there should be between 10 and 20 image pairs in the calibration (MathWorks 2021). The only other information that the app needs is the size of the checkerboard square, which is 80 mm in our checkerboard. In Figure 19, the left image shows different locations of the checkerboard during a calibration session in world coordinates, recognised from the image pairs using calibration matrices, and the right image shows an example photograph used in the calibration of the cameras.



(a) *(b)*

Figure 19. *(a) Output of the MATLAB Stereo Camera Calibrator App showing the 3D coordinates and pose for the checkerboard in the calibration image pairs. (b) Example photograph that was used in the calibration process.*

The Stereo Camera Calibrator App calculates both the intrinsic and the extrinsic camera parameters for both cameras (MathWorks 2021). A reliable indicator of how good the calibration success is can be found in the reprojection errors calculated by the application. An example output of the reprojection errors is shown in Figure 20.

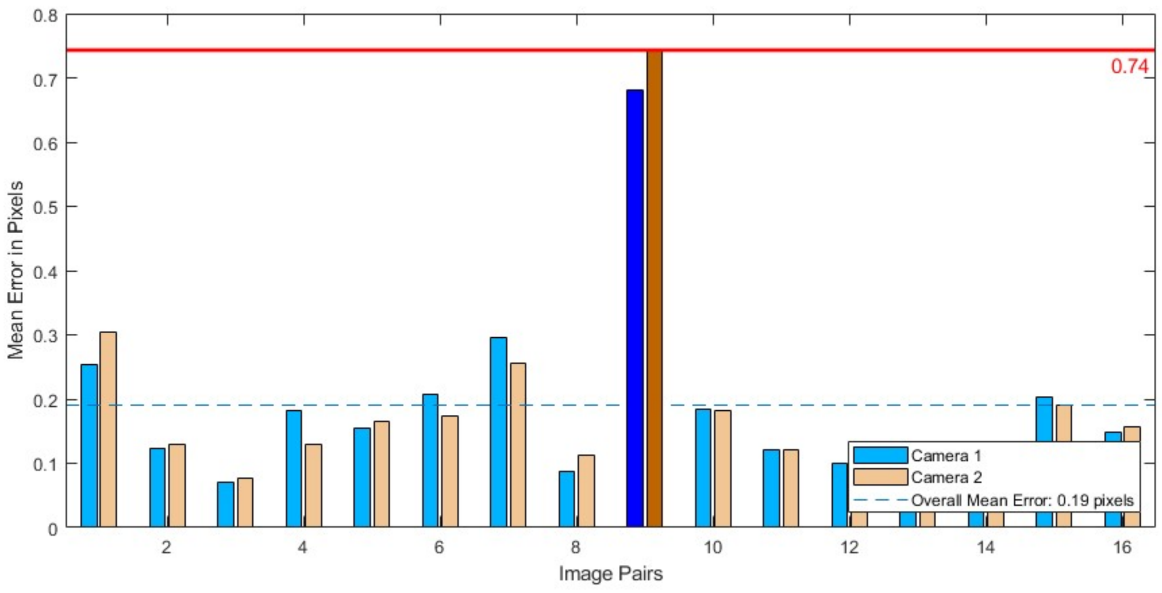


Figure 20. *Example reprojection error output in MATLAB's Stereo Camera Calibrator App.*

The output from the app is a *.mat-file*, which contains all the possible parameters that the app calculates. Since this format is specific to the MATLAB environment only, it must be converted to be compatible with OpenCV, which at the same time makes it compatible with ROS, which uses OpenCV. The end file format is YAML. In the 3D perception system workflow, this is done using MATLAB script together with C++ OpenCV API.

The compatible calibration config files are given to the Pylon ROS driver, which would then output grayscale rectified images from the camera. The other option is to give the config files to ROS image_proc nodes, which is discussed further in chapter 4.3.

OpenCV also has a stereo camera calibration capability, which is also integrated into ROS nodes. Although for the 3D perception system, the MATLAB Stereo Camera Calibrator App is used for its familiarity and the GUI, the OpenCV workflow would most likely give as good a camera calibration.

Jetson Xavier NX

The main computing hardware on the implemented 3D perception system is a Jetson Xavier NX, specifically the developer kit version. The other version of the same computing capability is the system-on-module version of the Xavier NX, and to be fully operational, it would need a carrier board. It belongs to the Nvidia's Jetson family, which are power efficient and small embedded system-on-modules designed to handle demanding deep learning and artificial intelligence calculation. (Nvidia 2021) The developer kit version has everything that a regular PC has, and operation of it is easy. The following Table 6 is presented the technical specifications of the Jetson Xavier NX developer kit embedded PC.

Table 6. *Technical specifications of Jetson Xavier NX developer kit (Nvidia 2021).*

GPU	Nvidia Volta architecture with 384 Nvidia CUDA cores and 48 Tensor cores
CPU	6-core Nvidia Carmel ARM v8.2 64-bit CPU 6MB L2 + 4 MB L3
DL Accelerator	2x NVDLA Engines
Vision Accelerator	7-way VLIW Vision Processor
Memory	8 GB 128-bit LPDDR4x @ 51.2 GB/s
Storage	microSD, M.2 SSD
Video Encode	2x 4K @ 60 | 6x 1080p @ 60 | 14x 1080p @ 30 (H.265/H.264)
Video Decode	2x 4K @ 60 | 4x 4K @ 30 | 12x 1080p 30 (H.265) | 2x 4K @ 30 | 6x 1080p @ 60 | 16x 1080p @30 (H.264)
Camera	2x MIPI CSI-2 DPHY lanes

Connectivity	Gigabit Ethernet, M.2 Key E (Wi-Fi/BT included), M.2 Key M (NVMe)
Display	HDMI and Display port
Others	GPIO, SPI, UART, I^2C, I^2S
Mechanical dimensions	103 mm x 90.5 mm x 34.66 mm

As a standard storage capacity, the Xavier NX uses an SD card. Due to the nature of the perception system, a lot of data has to be either processed or written to the disk, which causes problems with SD cards since they do not have a sufficient write speed. Testing during the development showed that the SD card already has problems under 100 MB/s. The implementable 3D perception system was specified to be run at 24 FPS, both cameras using a resolution of 1920 by 1080 pixels and 8 bits per pixel. Thus, the total data flow will be around 100 MB/s. Therefore, the SD-card was replaced in the Xavier NX by NVMe M.2 SSD-disk, more specifically Samsung 970 EVO Plus SSD. The Samsung SSD should have a write speed of around 3500 MB/s, meaning that the data throughput should not be limited by the storage disk, even having room for increased throughput.

4.1.2 Hardware architecture

The implemented 3D perception system consists of two Basler acA1920-50gc GigE cameras in stereo setup connected using Ethernet to Jetson Xavier NX embedded PC. Jetson NX is used to control the image acquisition and all the processing required to turn the raw images into filtered point clouds. The hardware architecture is represented in Figure 21.

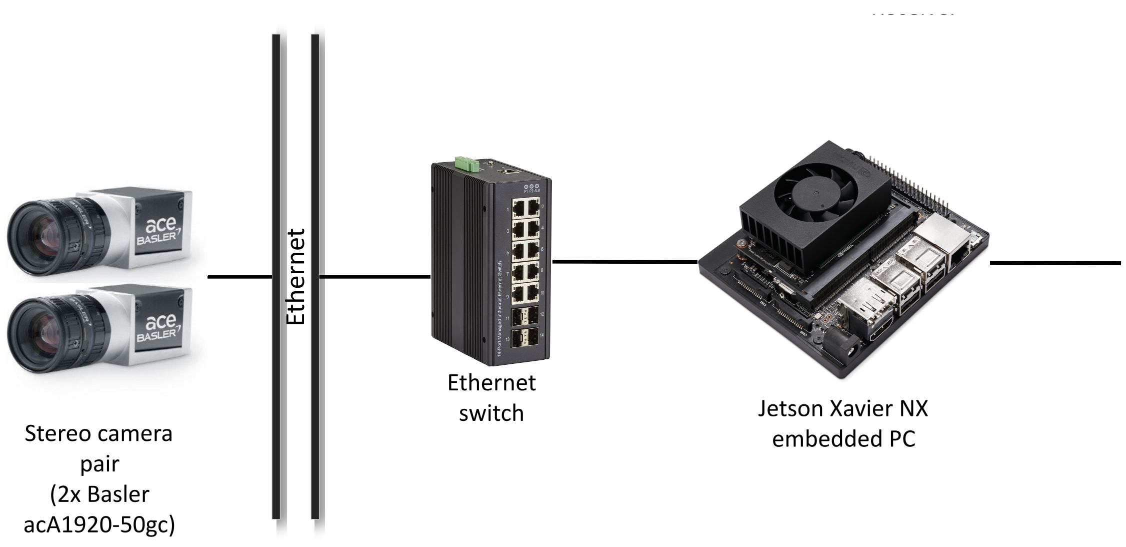

Ethernet
Ethernet switch
Stereo camera pair (2x Basler acA1920-50gc)
Jetson Xavier NX embedded PC

Figure 21. *Hardware architecture of the 3D perception system.*

4.1.3 Software architecture

The software used in the 3D perception system is run on the ROS environment. The system uses several different coding languages, including Python and C++. The system utilises a lot of different open-source libraries such as the well-known OpenCV and Point Cloud Library (PCL), and also some less known libraries such as Cupoch (Shirokuma 2021).

The overall 3D perception system software architecture is shown in Figure 22. The software architecture consists of four different main parts, that are:

1. image acquisition;
2. image rectification & undistortion;
3. depth image creation;
4. point cloud creation;

All the steps of the 3D perception system workflow are calculated and processed on the Jetson Xavier NX embedded PC. The input to the 3D perception system is the raw images provided by the Basler stereo camera setup, and the output is filtered point clouds.

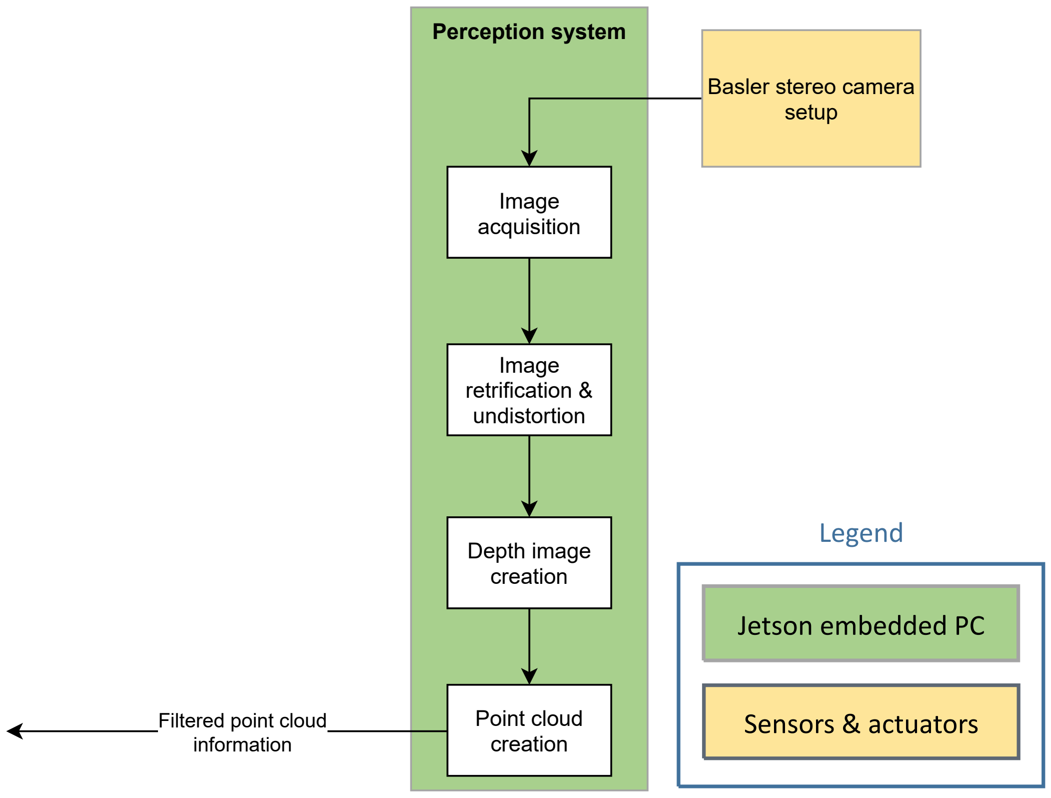


Figure 22. *Software architecture of the 3D perception system.*

In the 3D perception system, part of the workflow has been implemented to be run on the GPU using CUDA. Mainly the heavy computation parts and some functions that were already developed as CUDA code by the open-source library developers. Figure 23 shows the workflow of the 3D perception system and whether the function is implemented to be calculated with the GPU or CPU.

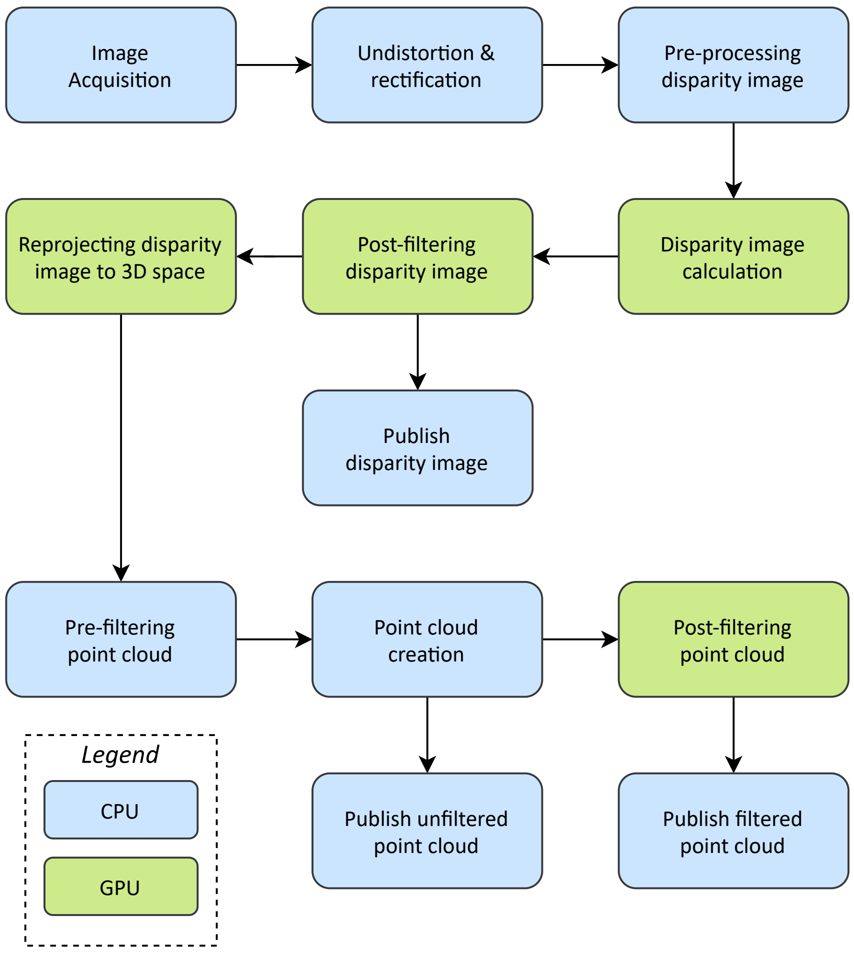


Image Acquisition
Undistortion & rectification
Pre-processing disparity image
Reprojecting disparity image to 3D space
Post-filtering disparity image
Disparity image calculation
Publish disparity image
Pre-filtering point cloud
Point cloud creation
Post-filtering point cloud
Legend
CPU
GPU
Publish unfiltered point cloud
Publish filtered point cloud

Figure 23. *The 3D perception system workflow with colours indicating the processing unit used to calculate the step.*

As mentioned earlier, the 3D perception system is run on the ROS environment, making the system easy to interface with other components of the robot and as well as adding additional calculations. ROS provides a lot of standard nodes for publishing, standard message formats for more effortless data transfer, and options for custom message formats. The ROS environment does not limit how many ROS nodes subscribe to a specific topic. Instead of pipelining the data to only a particular ROS node, the data can be easily distributed to several different ROS nodes. The computation load is also lowered when no data is transferred while there are no ROS nodes subscribed to a topic.

The published topics from the 3D perception system are disparity image and filtered point cloud and the unfiltered point cloud. In some cases, the data must be transferred outside of the ROS environment. which can be done using ROS nodes that send UDP packets to the targeted device using the available network.

The starting and shutting down of the nodes is controlled by using ROS-launch files; when possible, several launch files are combined to simplify the control of the 3D perception system. If it is not specially mentioned, the parameters are controlled through the

ROS-launch files bypassing the parameters to the nodes as arguments. That way, the parameters can be adjusted without recompiling the source code when an adjustment is made. ROS dynamic reconfigure, a ROS GUI for parameter adjusting is enabled for those parameters that can be adjusted during run-time. The rest of the parameters can be adjusted by turning off the node and starting it again with different parameters in the launch file.

The ROS architecture of the 3D perception system is shown in Figure 24. In this Figure, the green rectangles are active ROS nodes, and arrows are published topics, with their image type in parenthesis, to which any ROS node in the ROS network can subscribe. The greyed arrows are published topics, but no ROS node is subscribed to them.

The main libraries used in implementing the 3D perception system are OpenCV, PCL, CUDA, and Cupoch. The ROS version used is Melodic Morenia. The OpenCV library version is 4.5.1, which is compiled with support CUDA functionalities. The PCL version used is 1.8.1. The CUDA version used is 10.2.89, and the CUDA architecture is 7.2. The Cupoch version used is 0.2.2.0.

4.2 Stereo image acquisition

The raw images are acquired from the Basler acA1920-50gc cameras using ROS nodes provided by Basler. These ROS nodes are named *pylon-ros-camera*, and the source code for these ROS nodes can be found on Basler's GitHub page. (Basler 2021) The ROS nodes are interfaced to the Basler software Pylon, which is the formal driver between a computer and Basler cameras. The Pylon software must also be installed in order to be able to use these ROS nodes. Documentation on installing the Pylon software and testing the Pylon ROS driver can be found on Basler's website.

The Pylon ROS driver supports the Pylon software version 5, according to the installation instructions by Basler (Basler 2020), but in the implemented 3D perception system the version 6.2 is used successfully. Some minor modifications had to be done to the CMake-files in the Pylon ROS driver to import the correct Pylon software and the correct OpenCV version when compiling the package.

The underlying library of the Pylon ROS driver is OpenCV, which uses CPU in the image acquisition. There is also a possibility of using different, potentially faster libraries in image acquisition, for example, Gstreamer, which has CUDA support. The Basler Pylon ROS driver was chosen for the 3D perception system for simplicity and shorter development time. The Jetson Xavier NX embedded PC also has dedicated video coder/decoder hardware that could be used for the image acquisition (Nvidia 2021).

Because the amount of data throughput from the cameras is high, the standard User Datagram Protocol (UDP) packet size cannot handle it. For this reason, to use the

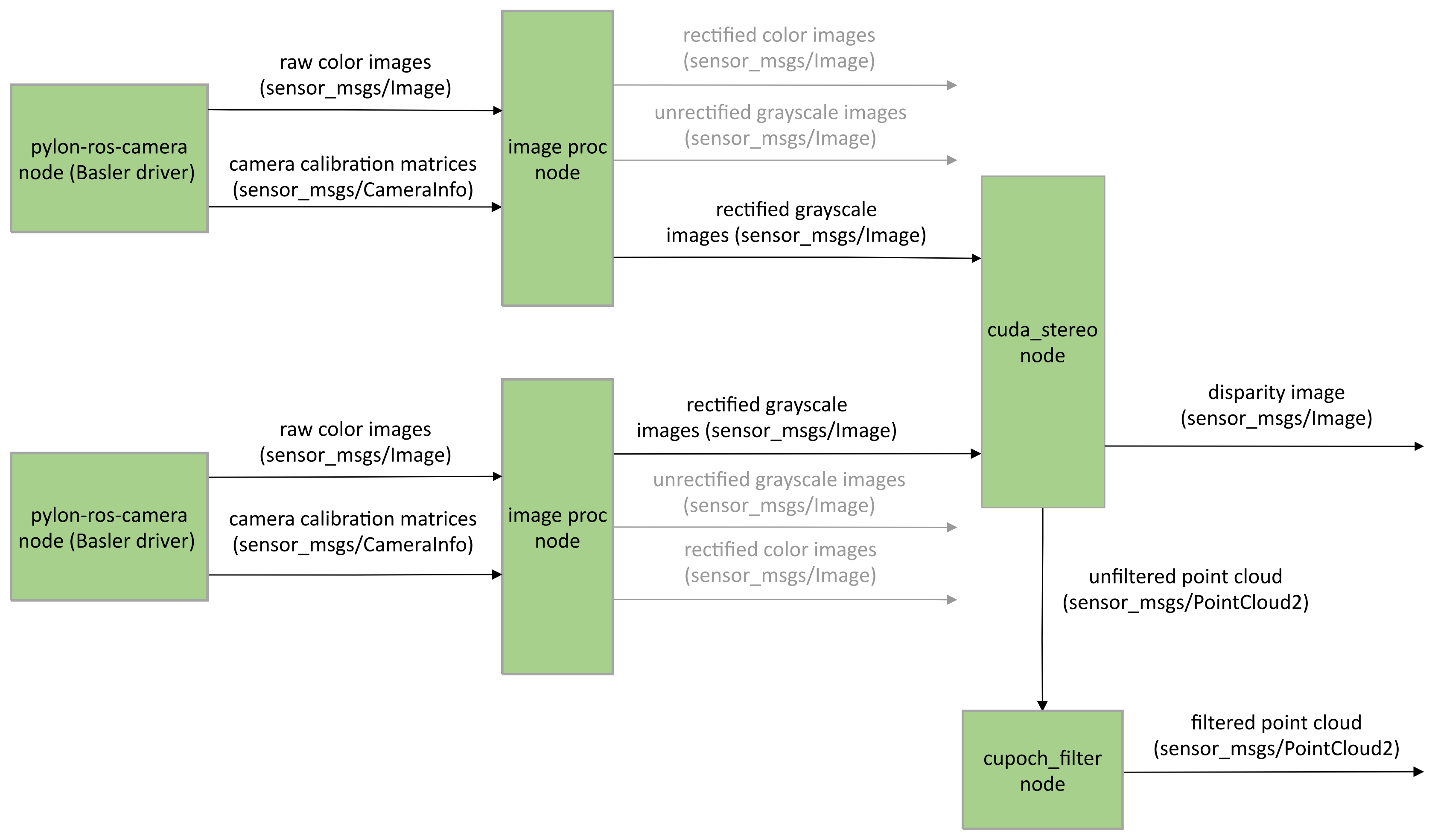

pylon-ros-camera node (Basler driver)
raw color images (sensor_msgs/Image)
camera calibration matrices (sensor_msgs/CameraInfo)
image proc node
rectified color images (sensor_msgs/Image)
unrectified grayscale images (sensor_msgs/Image)
rectified grayscale images (sensor_msgs/Image)
pylon-ros-camera node (Basler driver)
raw color images (sensor_msgs/Image)
camera calibration matrices (sensor_msgs/CameraInfo)
image proc node
rectified grayscale images (sensor_msgs/Image)
unrectified grayscale images (sensor_msgs/Image)
rectified color images (sensor_msgs/Image)
cuda_stereo node
disparity image (sensor_msgs/Image)
unfiltered point cloud (sensor_msgs/PointCloud2)
cupoch_filter node
filtered point cloud (sensor_msgs/PointCloud2)

Figure 24. *The 3D perception system ROS architecture.*

cameras to their full potential, the jumbo frames must be enabled in the receiving PC and the cameras. Jumbo frames reference to Ethernet frames that are over 1500 bytes in size, also called MTU size. The maximum jumbo frame size is usually 9000 bytes, which is also the size used in the 3D perception system. The Basler camera defaults to 1500 bytes in Ethernet frames, and the jumbo frames can be enabled with the Pylon software.

Basler cameras support large Area of Interest (AOI), which is beneficial to get more pixels in the images and, thus, more precise observations of the surrounding environment. However, a larger pixel count equals to larger image size, which causes a problem with the default UDP to receive buffer size in the Jetson Jetpack operating system, the used operating system in Jetson Xavier NX. With large AOI, the image contains more data than the buffer can hold at a time, which results in all UDP segments of an image data not fitting in the buffer simultaneously. When a ROS node makes a read request to a buffer that does not have all UDP segments of an image, it will get a partial image and throws an error. To ensure that the image acquisition works correctly, the maximum UDP receive buffer size must be increased at least to the size of one raw image in the receiving PC.

4.2.1 Hardware triggering

The hardware triggering is set so that in the Basler stereo setup, the left camera sends a signal when to start the exposure to the right camera. The signal is carried in GPIO a line. The way to set up the GPIO physically between the cameras was discussed in chapter 4.1.1. The hardware triggering must be configured in the User set for both cameras on the software side, which is done using the Pylon software.

The Pylon ROS driver does not support the hardware triggering by default in the master branch of the GitHub repository. Therefore, a non-official version of the Pylon ROS driver must be used. The open pull request to the Pylon ROS driver GitHub repository number 46 made by Neel Bhatt, GitHub username Neel1302, implements hardware triggering to the source code. By merging this pull request with the master branch, the hardware triggering can be used with the *pylon-ros-camera* ROS node. (Bhatt 2020)

4.3 Undistortion and rectification of the camera images

The raw images coming from the cameras are not good enough for the disparity image calculation as they are, mainly because the rows are not aligned. To get rectified and undistorted images, the raw images are processed with ROS *image_proc* nodes using the camera calibration matrices presented in chapter 2.1. The calibration matrices are fed to the *image_proc* nodes by the *pylon-ros-camera* nodes that publish *camera_info* topic for each camera, to which the *image_proc* nodes subscribe. The relevant part of the whole ROS architecture to this chapter is shown in Figure 25.

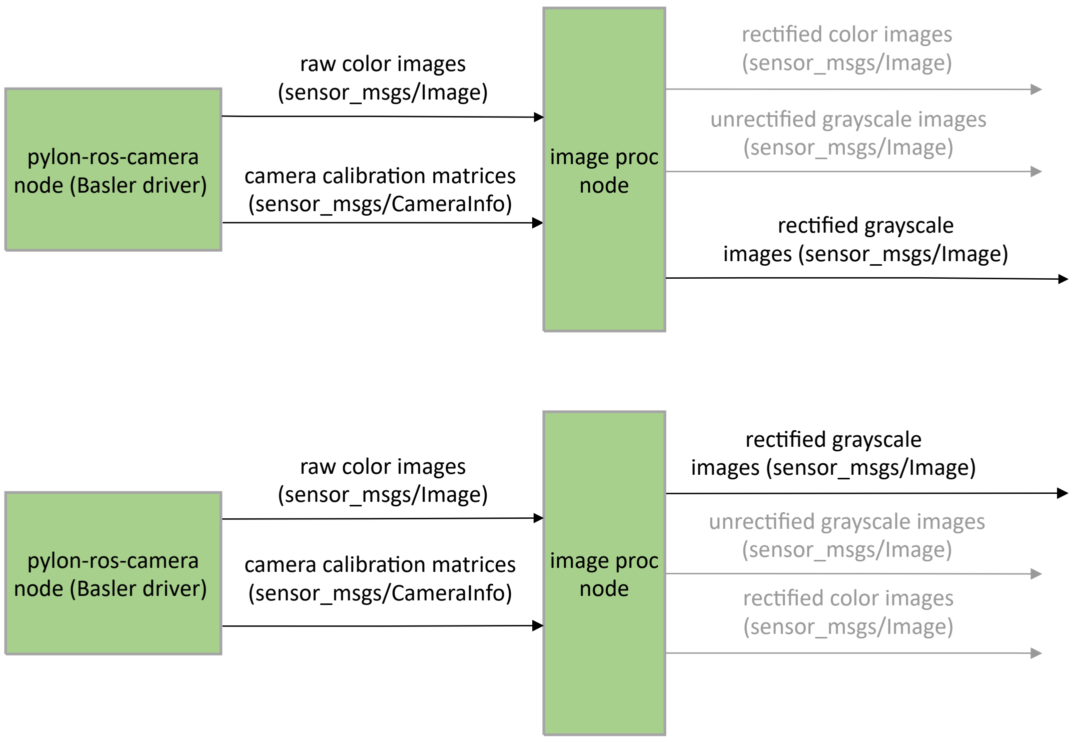

pylon-ros-camera node (Basler driver)
raw color images (sensor_msgs/Image)
camera calibration matrices (sensor_msgs/CameraInfo)
image proc node
rectified color images (sensor_msgs/Image)
unrectified grayscale images (sensor_msgs/Image)
rectified grayscale images (sensor_msgs/Image)
pylon-ros-camera node (Basler driver)
raw color images (sensor_msgs/Image)
camera calibration matrices (sensor_msgs/CameraInfo)
image proc node
rectified grayscale images (sensor_msgs/Image)
unrectified grayscale images (sensor_msgs/Image)
rectified color images (sensor_msgs/Image)

Figure 25. *Image undistortion and rectification part of the 3D perception system ROS architecture.*

Since this step is implemented using standard ROS nodes, the calculation is carried out by the CPU, and there is no option to calculate with the GPU in the *image_proc* node. A red-cyan stereo anaglyph fused from the undistorted and rectified left and right images are shown in Figure 26. Showing the disparity between the images and as well as how well the calibration succeeded. In good calibration, the horizontal objects should be on the same row.

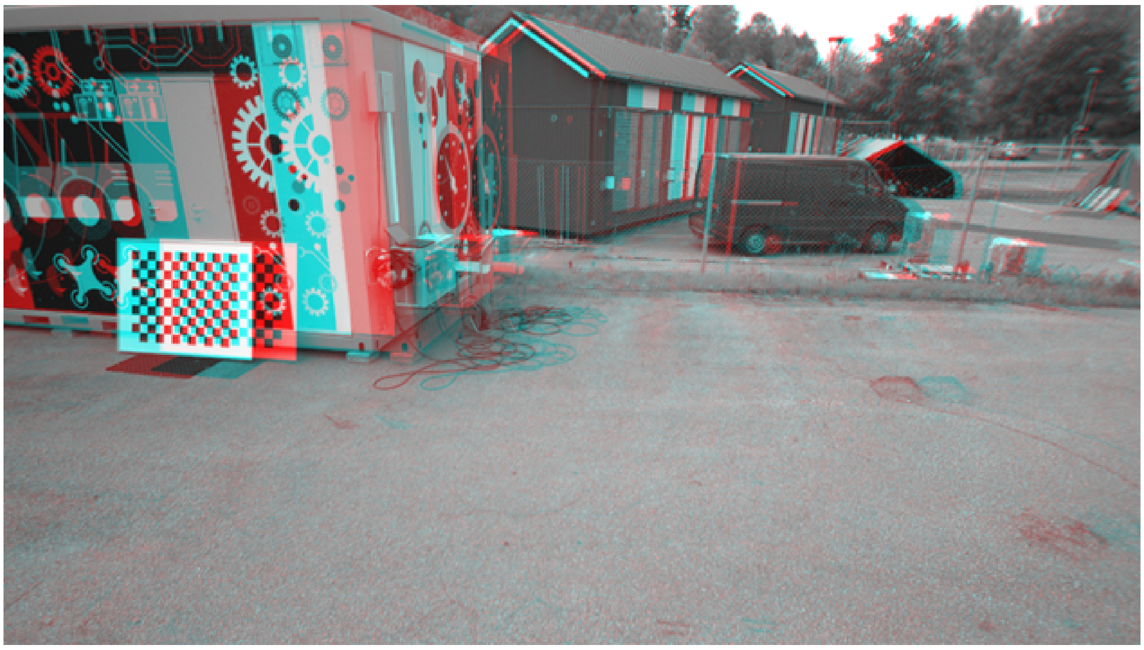


Figure 26. *Example red-cyan stereo anaglyph from the Basler stereo setup, processed with ROS image_proc nodes using the calibration matrices from the MATLAB Stereo Camera Calibrator App.*

An alternative method for the undistortion and rectification would be using the rectified images from the *pylon-ros-camera node*. The downside of using *pylon-ros-camera* nodes to process the raw images is that the colour information is lost since the *pylon-ros-camera* node cannot publish rectified colour images, only grayscale. The *image_proc* node publishes both grayscale and colour rectified images, which can be used, for example, in image recognition.

ROS also has a node called *stereo_image_proc*, which does the same as two *image_proc* nodes, but also calculates the disparity image and point cloud simultaneously. For the data amount that the 3D perception system uses, this node cannot produce even one frame a second since it does all the calculations with the CPU. Therefore, it is not used, but a faster GPU implementation is used instead.

4.4 Disparity image creation

This section presents the implementation of the disparity image calculation. Calculating the disparity image is a process, which can be divided into three parts as follows:

1. pre-processing the disparity image;
2. disparity image calculation;
3. post-filtering the disparity image.

The disparity image calculation is done in the *cuda_stereo* ROS node in the source code.

The relevant part of the whole ROS architecture, Figure 24, is shown in Figure 27. The input to this third step of the 3D perception system workflow is rectified grayscale images for both cameras, and the output is a disparity image.

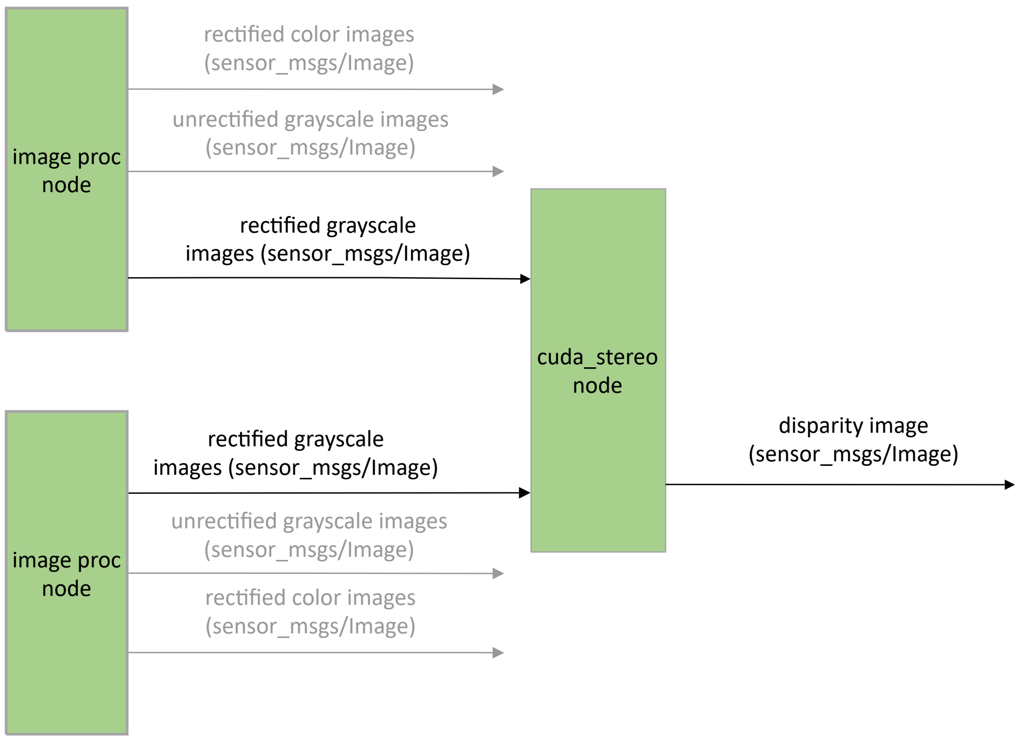


image proc node
rectified color images (sensor_msgs/Image)
unrectified grayscale images (sensor_msgs/Image)
rectified grayscale images (sensor_msgs/Image)
cuda_stereo node
disparity image (sensor_msgs/Image)
image proc node
rectified grayscale images (sensor_msgs/Image)
unrectified grayscale images (sensor_msgs/Image)
rectified color images (sensor_msgs/Image)

Figure 27. *Disparity image creation part of the 3D perception ROS architecture.*

4.4.1 Pre-processing the disparity image

It is beneficial to pre-process the undistorted and rectified images in many applications before calculating the disparity image. Usually, the most beneficial feature is that the calculation time for the disparity image decreases significantly if the image pixel count is lowered. The downside of downsampling the images before the disparity images are that some accuracy can be lost in the resulting disparity, and the overall quality decreases.

In the implemented 3D perception system, the processed images are downsampled 50% before the disparity image calculation. The decreased disparity calculation time benefit outweighs the little worse disparity image quality in the system. The increased frames per second are crucial to get a better overall system since it decreases the time between observations of the surrounding environment. Thus, allowing better tracking of a dynamic environment, which decreases the time from observation of a location change of an object to an actuator change to avoid an object, resulting in a safe overall system. The downsampling is done with OpenCV using the bit exact bilinear interpolation as the interpolation method. This step is done with CPU, even though there exists GPU implementation of the function in OpenCV. The GPU implementation could benefit the overall calculation time. However, copying and downloading the data from GPU might, in total, decrease the calculation time since the next step requires the data to be on the CPU in the current implementation.

4.4.2 Disparity image calculation

There exists a lot of different algorithms for the disparity image calculation. Most of them are based on the SGM (Hirschmuller 2008), like the algorithm that is chosen to be used in the implemented 3D perception system.

The algorithm chosen for the 3D perception system is a slightly modified version of the SGM. The algorithm is based on the GitHub repository by Daniel Hernandez-Juarez, which is a repository containing source code for the paper *"Embedded real-time stereo estimation via Semi-Global Matching on the GPU"* (Hernandez-Juarez et al. 2016). The GitHub repository was forked, and it was modified so that calling the function multiple times in a row is made possible. The algorithm is implemented in CUDA.

Hernandez-Juarez's SGM has four different configurable parameters:

1. penalty for small disparity changes P_1;
2. penalty for large disparity discontinuities P_2;
3. maximum disparity value d_{max};
4. path direction count r_{total}.

The penalties P_1 and P_2 are the main parameters to be configured since it is advisable to keep the maximum disparity value d_{max} as high as possible. The path direction count r_{total} is also constant, chosen on depending how much the decreased computation time is valued against the quality of the disparity image. The penalty P_1 changes how well slanted and curved surfaces are observed. The penalty P_2 acts as a median filter, smoothening the disparity image and removing abrupt changes. (Hernandez-Juarez et al. 2016)

The 3D perception system uses a maximum disparity value of 128 pixels, giving good enough depth while at the same time keeping the ability to see close with the camera. The Hernandez-Juarez's SGM version also has a limitation that the maximum disparity value d_{max} is 128 (Hernandez-Juarez et al. 2016). The path direction count is set to 4 in the perception system. It is chosen to get better quality on the disparity image while keeping the calculation time low, showing similar results to Hernandez-Juarez (Hernandez-Juarez et al. 2016) The algorithm is limited that the AOI pixels must be divisible by 4. Our original AOI is 1920 by 1080 pixels in the undistorted and rectified images, which is downsampled by 50% in the pre-processing phase. Thus, the resulting 960 by 540 pixels AOI is used as input to the SGM, divisible by 4.

Alternative algorithms for the Hernandez-Juares SGM are the OpenCV block-matching and SGM algorithms. There are both CPU and GPU implementations in the OpenCV library for the first one. Unfortunately, there does not exist GPU implementation for the OpenCV SGM algorithm since it seems to produce good results with the CPU implementation. However, it is not suitable for the 3D perception system due to the long execution time. The OpenCV block matching algorithm GPU version was tested during the development, achieving a feasible execution time. However, for unknown reasons, the parameters sent to the function, calculating the algorithm, did not affect the resulting disparity image. Given these points, the OpenCV block matching algorithm was abandoned due to bugs in the source code.

4.4.3 Post-filtering the disparity image

The disparity image itself is rarely good enough as it is and usually needs some post-filtering. If the disparity image contains noise, it will lead to a noisy point cloud, especially sub-pixel disparities that are prone to noise. Since the image disparity values are correlated to the 3D location of the pixel, the potential noise must be filtered out to avoid false positives and other noise in the point cloud.

In the 3D perception system, a median filter is used to post-filter the disparity image. A median filter is a smoothing filter, meaning that when using it, potential weak edges in the images are lost, but moderate Gaussian noise is effectively filtered out. A Gaussian filter could be used to preserve the edges better. However, it might affect how well the noise is filtered out. The median filter used in the perception system is implemented in Hernandez-Juarez's GitHub repository as CUDA kernel, and the same filter is used in the 3D perception system (Hernandez-Juarez et al. 2016).

In the left part of the disparity image, there is some almost static noise due to HernandezJuarez's SGM algorithm used to calculate the disparity image. The noise is not dependent on the environment the cameras are looking in and is always present. Thus, it can be segmented out. The segmentation is done because this noise creates many outliers to the point cloud, decreasing the quality of the point cloud and slowing down the system. The noise is segmented out by creating a new version of the OpenCV Mat-class, holding the disparity image, by indexing part of it out. With this approach, the data is not copied over, so there is no noticeable execution time change. The noise is analysed to be only on the first 70 columns of the disparity image. The segmented part is shown in Figure 28, the left side of the white vertical line.

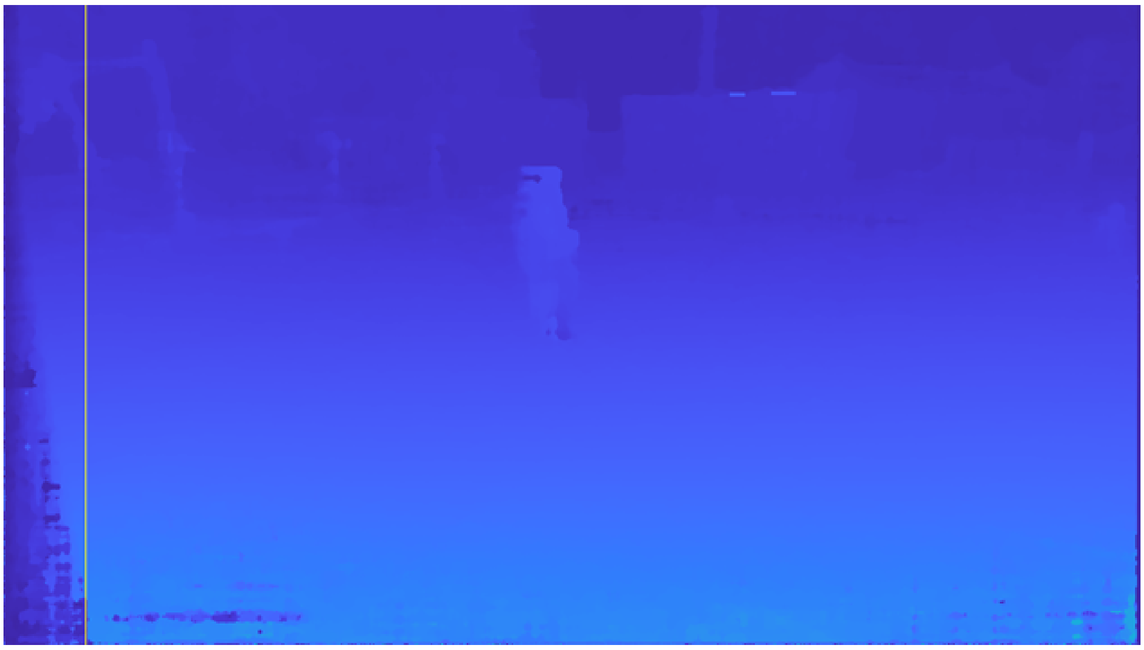


Figure 28. *Example disparity image showing the segmented area on the left of the white line. A colourmap has been applied to the disparity for better visualisation.*

The segmentation has a minor decrease in the horizontal FOV of the 3D perception system. However, the benefits in the quality of the point cloud and execution time outweigh the slight loss in horizontal FOV. It is measured that the segmentation only decreases the left side of the horizontal FOV by 1°.

Alternatively, there exists a lot of different image filters in OpenCV that could be used, but most of them are implemented as CPU code, which would slow down the 3D perception system.

4.5 Point cloud creation

This chapter discussed implementing the point cloud creation from the disparity image in the 3D perception system. Creating the filtered point cloud can be divided into a four-step process as follows:

1. reprojecting disparity image to 3D space;
2. pre-filtering the point cloud;
3. point cloud creation;
4. post-filtering the point cloud.

The input to this last part of the 3D perception system is the disparity image from the earlier step, chapter 4.4 Disparity image calculation. The output is a filtered point cloud that can be then fed, for example, to an autonomous driving function to dodge obstacles. The relevant part of the 3D perception system ROS architecture to this chapter is presented in Figure 29. In the implementation, the point cloud creation described in this section is part of two different ROS nodes, *cuda_stereo* and *cupoch_filter*.

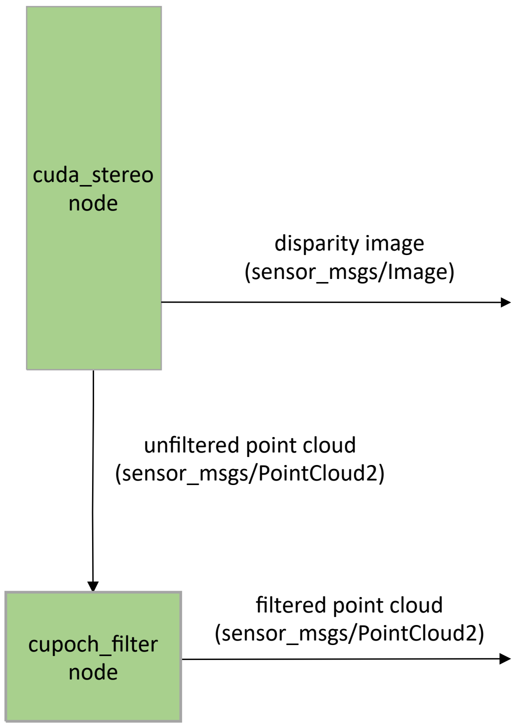

cuda_stereo node
disparity image (sensor_msgs/Image)
unfiltered point cloud (sensor_msgs/PointCloud2)
cupoch_filter node
filtered point cloud (sensor_msgs/PointCloud2)

Figure 29. *Point cloud creation part of the 3D perception system ROS architecture.*

4.5.1 Reprojecting disparity image to 3D space

To create a point cloud from the disparity image, the pixel disparities must be converted to 3D locations. For each pixel in the disparity image, one 3D point can be created. The disparity values can be turned into 3D locations using the camera calibration matrices. The underlying ability to transform is based on epipolar geometry, discussed in chapter 2.1.

In the 3D perception system, the reprojection is done with OpenCV using the CUDA implementation of the function *reprojectImageTo3D*. There also exists a CPU version of the same function in the OpenCV library, which is slower than the CUDA version and, therefore, not used. OpenCV needs the perspective transformation matrix in the reprojection, which is not calculated when using MATLAB's Stereo Camera Calibrator App. Therefore, this matrix must be calculated with *stereoRectify*-function included in OpenCV. In the 3D perception system workflow, this is done in chapter 4.1.1 in the MATLAB script, which converted the MATLAB stereo parameters to YAML-format, which both OpenCV and ROS use.

The output of the *reprojectImageTo3D*-function is the same sized matrix as the input but in three dimensions. Each dimension respectively contains the X-, Y-, and Z-coordinates of each point, which lie on the same row and column as the disparity pixel value in the disparity image.

4.5.2 Pre-filtering the point cloud

Before creating a point cloud, it is beneficial to filter the matrix containing the 3D locations of the points, called pre-filtering of the point cloud. The *reprojectImageTo3D*-function of the previous step does not consider that some of the pixels might have zero or infinite disparity values. That results in 3D locations, where each coordinate can be infinity or zero, points that do not exist in the natural environment.

In the implemented 3D perception system, the coordinate matrix is iterated through with the *inRange*-function included in the OpenCV. With this function, the coordinate matrix is gone through with lower and upper limits for each coordinate. If the point has one of the three coordinates under the lower limit or over the upper limit, it is marked as zero; otherwise, as one, creating a same sized matrix as the coordinate matrix containing only zeros and ones, being a Boolean mask matrix. This matrix is used in the point cloud creation. This step of the point cloud creation is calculated with CPU. Alternatively, this kind of filtering could be done in the post-filtering of the point cloud but doing it before creating the point cloud makes the point cloud creation faster, making the overall 3D perception system faster. The lower and upper limit is -20 meters and 20 meters correspondingly for each dimension in the 3D perception system.

4.5.3 Point cloud creation

The point cloud creation is a simple step of iterating over the coordinate matrix according to the mask matrix created earlier. In this step, the coordinate matrix is converted to ROS message *sensor_msgs/PointCloud2*, containing the 3D locations of the points in float format. The resulting point cloud is called the unfiltered point cloud, published under the same name in the ROS network, as seen in Figure 29. It is not an organised point cloud, so it does not have the same shape as the coordinate matrix and disparity image. A benefit of an unorganised point cloud is the speed of iterating over the data, which is faster than using iterators on an organised point cloud or push backing to a point cloud. The point cloud only contains the x-, y-, and z-coordinates of the points, even though the colour information could also be passed from the rectified colour images to the point cloud. By leaving out the colour information, the calculation is naturally faster. Both parts of the point cloud creation are calculated with CPU.

There is also the possibility of creating a PCL point cloud from the coordinate matrix, but since the post-filtering of the point cloud is not done with the PCL, it is not used in the implemented 3D perception system. There is no ready-made function in the libraries that are used in the 3D perception system, meaning that a significant computation time decrease could be achieved by writing a custom CUDA function for this step in the future.

4.5.4 Post-filtering the point cloud

The point cloud still contains many points, some of which are unnecessary as the same shape can be represented with fewer points. Usually, some noise is shown as outlier points that are not reasonably connected to other shapes of the observed environment. Therefore, some filters are applied to the point cloud to make it cleaner and smaller in size while keeping the shape simultaneously.

The filtering operations in the implemented 3D perception system are done using the Cupoch library. This library is based on the more popular library Open3D, an open-source library that is designed to handle 3D data (Shirokuma, 2021). The Cupoch library idea is to implement the same functions that the Open3D library has, but implementing them in CUDA and thus, improving the speed of the calculations making soft real-time possible. The Open3D has all its functions implemented to be calculated on CPU. The Cupoch library has comprehensively converted many functions from the Open3D. The library has good Python API and support for direct C++ code usage. The author of the Cupoch library has also made the library available as a pip package.

The first filtering operation performed to the unfiltered point cloud is a voxel grid downsampling. This filter aims to reduce the number of points in the point cloud while trying to maintain the shape of the objects, which is done with voxel grids. A voxel grid is a smaller piece of the point cloud that has a shape of a cube in the point cloud space. In the filter, the whole space that the point cloud has is divided into these voxel grids. Then, the points inside that voxel are taken for each voxel, and their centroid is calculated. In the end, only the centroid is returned to the point cloud, thus eliminating other points from that voxel except for the centroid. The voxel grid filtering principle is shown in Figure 30.

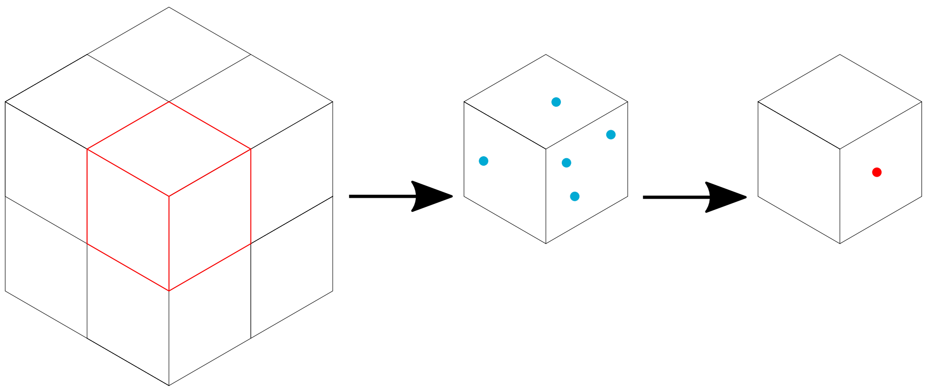


Figure 30. *Voxel grid downsampling principle.*

After filtering the point cloud with the voxel grid filter, the point cloud still usually contains outliers, false positives of the surrounding environment. A statistical outlier removal filter removes these outliers from the point cloud. The idea in the filter is to calculate the mean Euclidean distance from a point in the point cloud to chosen number of neighbouring points. This is done for every point in the cloud. The mean distances are assumed to be on a Gaussian distribution, characterised by a mean value and a standard deviation. Then the outliers are trimmed from the point cloud according to how the points fit the Gaussian distribution. (Rusu et al. 2008) Example pictures of statistical outlier removal filtering effect and mean distances are shown in Figure 31.

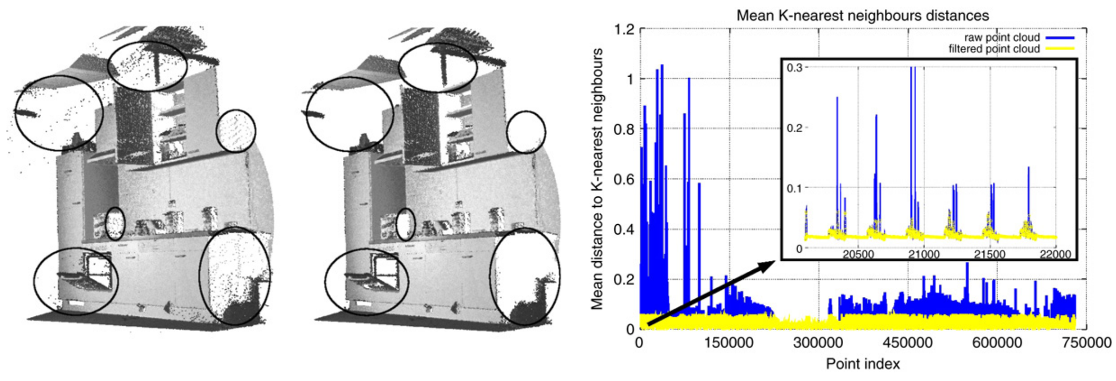


Figure 31. *Example of statistical outlier removal filtering. Left shows kitchen scene with outliers and after filtering. Right shows mean distances before and after filtering. Reprinted from Semantic Knowledge in Robotics, Vol 56, Rusu et al., Towards 3D Point cloud based object maps for household environments, Pages 927-941, Copyright (2008), with permission from Elsevier.*

The usage of both voxel grid downsampling and statistical outlier removal filters is made easy in the Cupoch Python API. The voxel grid downsampling is implemented as an attribute to the core data format of the library *geometry.PointCloud*, which is called with only one parameter: voxel size, which is the length of the side in the voxel cube. The output from the function is the point cloud downsampled by the chosen voxel size given as a parameter to the attribute. The statistical removal filter is also implemented as an attribute of the class *geometry.PointCloud*. This attribute is called with two parameters; the number of neighbours and standard deviation ratio. The number of neighbours signifies how many neighbours are included in calculating the Euclidean mean distance from a point to neighbouring points. The standard deviation sets the threshold on the mean distance distribution, which is based on how the points are trimmed out. The outputs of the statistical removal filter are point cloud with indicated which points are outliers and vector indexes that are inliers. Then, by selecting the indexes from the original point cloud, the result is point cloud only containing the inliers.

In the 3D perception system, the Cupoch library is used as Python 3 library, installed via a pip package. The source code does not have to be compiled with the Jetson, which is not straightforward in many cases but made easy by using the pip package. The ROS Melodic, on the other hand, does not support Python 3 by default, so minor modifications must be done to get Python 3 code running in the ROS environment. The main trick is to make a virtual environment in Python 2 that is running Python 3. To publish the filtered point cloud, the function *cupoch_msg_info_to_pointCloud2* in Cupoch is used to turn the *geometry.PointCloud* to publishable *sensor_msgs/PointCloud2* ROS message, which can be then published with standard rospy functions.

An alternative library that could be used instead of Cupoch would be the PCL. The upside of it would be the broader filter selection. However, the big downside is that it is implemented to run on a CPU, which would significantly eliminate the 3D perception system capability. The PCL also has some functions implemented in CUDA code, but these functions did not correctly work for one reason or another during the development. One alternative to voxel grid downsampling would be pixel jumping, in other words skipping pixels when creating the point cloud. Hence, it has less to calculate and, thus, would be probably faster if correctly implemented. The downside is that it does not keep the shape of the objects so well that the voxel grid downsampling keeps.

5. RESULTS AND DISCUSSION

This chapter discusses the results of the research and implementation of this thesis. The implemented system is evaluated using the heavy-duty field requirements formed in chapter 3, and potential solutions to shortcomings are discussed. The second subchapter answers the two later research questions alongside the research objective of forming design principles for a 3D perception system in heavy-duty field robots. In addition, some general solutions to requirements are presented.

5.1 Evaluation of implemented system

The details about the implemented discussion were presented in earlier chapter 4, and in this subchapter, the implemented system is evaluated against the requirements in Table 1. The requirements are not gone in the ascending number order.

One of the essential requirements for the implemented 3D perception system to fulfil is the real-time requirement. The implemented 3D perception system achieves 2,5 FPS in an unstructured environment when observing the environment randomly. The environment is observed randomly because the number of obstacles affects how many points are generated in the point cloud, which affects the FPS. Thus, random observing seemed most fit. Earlier the real-time was defined as soft real-time, and the FPS would be at least 5. Currently, the system is not achieving the desired observation rate; thus, it does not fulfil this requirement. The limit comes mainly from the available computational capability of the Jetson Xavier NX. A lot of the calculation steps is run on the CPU, making the system slow on those steps. Also, the desired image AOI 1920 by 1080 pixels is quite large, and it creates a lot of calculations for the system. The system could improve performance by either optimizing the calculation or lowering the desired specifications. First, the optimization could be done either with the current hardware or with better hardware. The first option would require more work since the researchers must convert the possible CPU code to run on the GPU. The second one is easier since the whole software is run on the ROS environment, giving a possibility to change the machine that it is run. Although, the CUDA code must be alternated to match the new GPU. If one would continue using the Jetson Xavier NX, making a cluster of those system modules would also be beneficial. In those clusters, one module could run the disparity image calculation and the other the point cloud creation and so on. That would linearly increase the computational capability.

Due to the limitations present in many disparity image calculation algorithms, as well as the one used in the implemented 3D perception system, the horizontal FOV in the observable environment is slightly lower than the raw camera images have, and some blind areas exist. The observable area in the 3D perception system is illustrated in Figure 32.

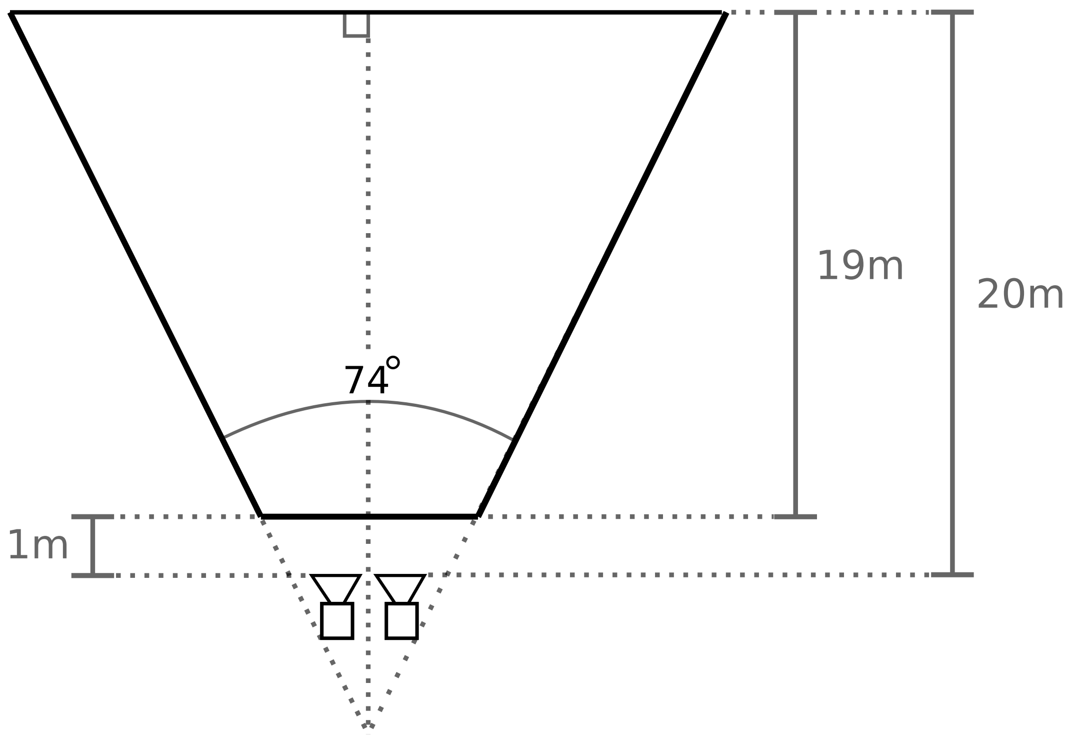


Figure 32. *The observable area of the implemented 3D perception system, measured from the filtered point cloud.*

The system has a horizontal 74° FOV in the filtered point cloud, which guarantees good visibility over the surrounding environment in front of the system, even at close distances. The loss in the horizontal FOV is around 20° when compared to the original horizontal FOV that a raw camera image has, captured by one Basler camera. Most of the loss in the horizontal FOV comes from the significant baseline; in this case, the distance between cameras is 37 cm. The loss can be explained as both cameras must see to the same location to extract 3D points from it using epipolar geometry, and if both cameras can not see to that location, it becomes a blind spot. There is always a blind area on both sides of the observable area. There also is a small blind area at a close distance between the cameras. Some loss is also from the disparity image calculation, which leaves the left side of the disparity image with zero disparity values for the pixels. The area does not differ depending on the environment, and thus, it is segmented out from the point cloud to increase the quality of the cloud. However, this segmentation had only a minor impact on the horizontal FOV.

The largest depth that is observable with the system is 20 meters to the front of the setup. This is due to the limitation in the source code to segment out the points behind this depth to achieve better performance and filter some of the outliers. The limitation removes the points at unreachable and infinity depths. Although, the used SGM algorithm itself cannot calculate correct disparities for longer distances than 20 meters with this stereo camera setup. The 3D perception system uses the maximum disparity value available in Hernandez-Juarez's SGM algorithm (Hernandez-Juarez et al. 2016), meaning that the maximum depth is limited on the SGM source code also. In addition, one of the common disadvantages of these disparity calculation algorithms can be seen in Figure 32. The observable area does not start right from the cameras; instead, there exists a blind area, which is estimated to be around 1 meter starting from the camera lenses to the front of the system, but no exact measurement has been made.

The observation area could be increased by achieving better calibration and solving the problem related to Hernandez-Juarez's SGM algorithm noise problem in the left side of the disparity image. The maximum depth could also be increased by using a larger maximum disparity value, but that would also decrease the blind areas of the stereo camera.

The two crucial requirements were discussed earlier, the observation rate and area requirements. Now the implemented 3D perception system is reviewed against the rest of the requirements in Table 4. Firstly, there was not set any particular manipulation scope for the 3D perception system, but the ideal scope for it is in a low-speed autonomous driving or manipulator movement. The observation rate of the system is too low for high-speed driving and movements. The observation area is big enough, for example, excavator arm manipulating. The vibration effect on the system has not been evaluated since the final placement of the system has not been decided. This implementation focused more on the hardware and available ROS node possibilities for 3D perception systems. The system is designed from scratch to be operated in a marker-less environment, and thus, it fulfils the requirement regarding marker-less environments. Only a stereo camera was used as a perception sensor. Both of the cameras were placed inside protective casings, which protects them from all weather conditions, except for dirt in the lenses of the casings. Although, the system has not been tested on adverse conditions for faulty observations. Therefore, adverse conditions requirements can not be said to be fully full-filled. The system was implemented for research usages, and thus, the safety standards have not been full-filled, and that would be one of the improvement categories for the system.

The algorithm for the 3D perception system was mainly chosen based on the speed of the algorithm and as well as the accuracy. It had shown promising results on the KITTI dataset

(Hernandez-Juarez et al. 2016), and it was implemented as CUDA code. The point cloud filtering algorithms were chosen on the same criteria. Thus, the algorithms were chosen based on the optimization criteria, but they have not been reviewed against non-ideal data features. As can be seen from Figure 23, some of the demanding calculation tasks have been implemented to be run on the GPU, but there still is some calculation steps that could be done with the GPU, for example, point cloud creation.

One of the requirements was that the 3D perception system should be dependable, meaning that the system does not fail too often and, in case of failure, does not cause harm. This requirement has not been reviewed, but from the system testing, it can be said that if the system starts successfully, then it should not crash very often. The property of not causing any harm is more related to the control system that uses the observations of the environment. It can be said that the system produces a lot of false positives in some scenarios and more in cases of vibration. Therefore, the control system using the data should have safety mechanisms for these scenarios. The system energy consumption of the system has not been analyzed but based on the component choices, Basler cameras and Jetson Xavier NX; the energy consumption should be pretty low, meaning that the energy efficiency requirement is full-filled.

5.2 3D perception system design principles

The second research objective is to form a 3D perception system design principles for heavy-duty field robots. In total, there were ten requirements from five different categories identified for the 3D perception system in heavy-duty field robots in Table 4. Now using these requirements, the design principles are formed.

In total, two heavy-duty field requirements were identified in chapter 3.1. The first requirement is that the 3D perception system should be designed to fit the correct manipulation scope, and the second is that the system should be robust to vibration. The critical aspect of the first requirement is sensor selection. Each sensor has very different properties, advantages and disadvantages, and choosing the correct type of sensor for the application is essential. From the sensor properties, the two important ones are the data type, what the sensor outputs and the maximum depth of the sensor. The data type defines what kind of tasks the system can perform. For example, object recognition on camera images versus point clouds is different. The maximum depth also defines what kind of tasks the system can do. Long-range LiDARs are sufficient for autonomous highway driving, but safety laser scanners are not. The second requirement has a significant effect on either the mechanical durability of the sensor or on the observation of the sensor. The vibration can be damped either mechanically, through different mountings and rubber paddings, or on the software side by doing virtual stabilization for cameras. As mentioned earlier, the movement towards electric engines might eliminate the vibration problem since electric engines cause a lot less vibration, or not at all, versus diesel engines. The first design principle for the 3D perception system in heavy-duty field robots is that perception sensors with their placement are in an essential role during the design.

The requirements regarding the surrounding environment were concluded to two requirements: the 3D perception system should be suitable to marker-less environments and be resistant to adverse conditions. The first requirement is related to the heavy-duty field robots working in unstructured environments, where inserting markers to objects and features are tedious. Using algorithms designed for marker-less environments is not trouble-free. The two problems are that the accuracy of SLAM algorithms is lower than the marker-based SLAM algorithms (Feng et al. 2015) and the marker-less environments need more computation power. Then again, with the increasing GPU calculation power with the future being in an unstructured environment, the requirement is in an important role. The adverse conditions cause problems in the durability of the sensors and as well as in the observations. One potential solution to false positives caused by faulty observations of the surrounding environments is to implement good safety functions to the robot that respond to these failures. Also, the sensor count can be increased, and overlapping between sensors could be introduced. The problems that LiDARs face because of fog, rain, and snowstorms are being under significant interest and solutions to the problems are searched. Some solutions have been presented that modify the wavelength of the laser as well as the power of the laser beam (Kutila et al. 2018; Michaud et al. 2015). The environmental requirements can be concluded to a second design principle: the target environment should be kept as one of the main design criteria during the development.

The third main category for the requirements is the safety requirements. In this category, two requirements were identified: the 3D perception system should obey the obligatory safety standards and follow the good practices from related standards, and the 3D perception system should be placed to get the best possible observation area and the least amount of blind spots. It was concluded that there are many different standards applicable to heavy-duty field robots, and the standards are still quite lacking in some industries. Therefore, the designer should look broadly at safety standards from different industries. The second requirement is quite similar to earlier was concluded about the importance of the perception sensors and their placement in the system. The sensor placement is capsulized in the first design principle. Therefore, the safety requirements can be concluded to the third design principle: that the relevant safety standards should be used as a guide during the development.

One of the main features of a 3D perception system is the observation rate, and therefore, the real-time requirements were defined. The requirements are mainly related to the software side of the 3D perception systems, and they are: the algorithms in 3D perception system should be chosen on the system optimization criteria and be critically reviewed against non-ideal data features, and the calculation steps in the 3D perception system should fit the correct calculation architecture. The first requirement serves as guidance on how the algorithms in the 3D perception systems can be compared and how differences between them can be distinguished. As seen, the algorithm's choosement is important, and tailoring it to specific hardware takes time. Therefore, the designer should be entirely sure about the algorithm that has been chosen. In chapter 3.2, two good listings were presented that give good guidance on choosing the algorithm. The second requirement targets that the developer has been thinking carefully about what parts of the system can and is reasonable to run on special hardware, such as GPUs, and what on the CPU. The difference can be significant to both directions in the overall performance if some parts of the system are not developed on the correct calculation architecture. It is worth trying to implement every part of the 3D perception system in GPU if the calculation part has either data or step independence because the current GPUs are so fast. Given these points, the real-time requirements can be formed as a fourth design principle, that is that the algorithm choosing and tailoring should be done carefully and with time.

The last main category for the requirements is the embedded requirements. These are more related to the embedded systems and edge calculation but also are related to the 3D perception systems. The embedded requirements were concluded that the 3D perception system should be dependable and energy-efficient. Dependability was defined as the system must have a low probability of failing, and in case of failure, there should be no harm caused. The dependability requirement is related to the safety standards requirement, where the performance levels were discussed. Making 3D perception systems dependable is not straightforward, and it needs cooperation from both perception sensor manufacturers as well as from the software developers. The main problems in this requirement are mostly related to the software side, having crashes and bugs, but also some perception sensors cannot be dependable in every scenario like it was seen with LiDARs. The energy efficiency requirement is important, but it does not need itself its own design principle, and it can be easily be included in the first and second already mentioned design principles. Therefore, the embedded requirements can be concluded as the fifth design principle: one of the main focus areas of a 3D perception system development is to have a dependable system.

As has been noted, the requirements give good guidance on how to implement a 3D perception system in heavy-duty field robots, and they are quite specific. The design principles were concluded from these ten requirements from five different categories into five design principles that should help and guide the developers on designing a 3D perception system for heavy-duty field robots. As a summary, the design principles are concluded in the following listing:

1. Perception sensors with their placement are in an essential role during the designing.

2. The target environment should be kept as one of the main design criteria during the development.
3. The relevant safety standards should be used as a guide during the development
4. The algorithm choosing and tailoring should be done carefully and with time.
5. One of the main focus areas of a 3D perception system development is to have a dependable system.

6. CONCLUSIONS

The first research objective of this thesis was to implement a 3D perception system that can be integrated as a part of a heavy-duty field robot allowing autonomous functions, using a stereo camera, Jetson embedded PC, and ROS. The whole 3D perception system was successfully implemented, and the filtered point clouds were obtained as the result of the system. The system was designed using the given components and software environments, and it made use of several open-source already available libraries. Related to the first research objective, the first research question was to determine whether the implemented system can fulfil the heavy-duty field requirements. The implemented system did not achieve defined real-time, be dependable, or meet the safety standards. However, the other requirements were met or could not be defined because they are more related to the end application.

The second research objective of this thesis was to define a set of design principles for a 3D perception system in heavy-duty field robots. The first research question full-filled the first research objective, and the two other research questions were related to the second research objective. The second research question targeted to find out what must be considered when designing a 3D perception system for a heavy-duty field robot and the last research question was that what are the design principles of the system. First, the requirements for a 3D perception system were formed, and in total, ten requirements were defined, which are in Table 4. Then using these requirements, the design principles were formed, focusing on one main category at a time, resulting in a set of 5 design principles found at the end of chapter 5.2.

The implementation was focused more on the possibilities that the current open-source libraries, ROS, and Jetson Xavier NX embedded PC can offer at the moment than meeting the requirements that were defined for the heavy-duty field robots. The current implementation was limited because of the limited computational capability. The system-on-chip and system-on-module products are advancing enormously at the moment, meaning that the current hiccups in the perception calculation are easily tackled in the future with more computation power. Therefore, the implementation is not a final product and should not be used in a robot as it is, and the development of the system should be continued. The design principles are currently working, but it is recommended to refine further through experience and testing.

REFERENCES

Aamodt, S. (2014). *Safety Laser Scanners vs. Safety Mats - Which One Do I choose?* URL: https://cdn.sick.com/media/docs/5/15/415/whitepaper_safety_laser_scanners_vs._safety_mats_en_im0058415.pdf.

Başarslan, O. and Yaldiz, E. (Apr. 2017). Implementation of FMCW radar for training applications. *2017 4th International Conference on Electrical and Electronic Engineering (ICEEE)*. Journal Abbreviation: 2017 4th International Conference on Electrical and Electronic Engineering (ICEEE), pp. 304–308. DOI: 10.1109/ICEEE2.2017.7935839.

Basler (2016). *Basler ace User's Manual for GigE Cameras Document Number: AW000893*.

– (June 2020). *Interfacing Basler Cameras with ROS Document Number: AW001491*.

– (2021). *Pylon ROS driver*. URL: https://github.com/basler/pylon-ros-camera (visited on 12/18/2021).

Bhatt, N. (2020). *Hardware Trigger Support, open pull request to Pylon ROS driver GitHub repository*. URL: https://github.com/basler/pylon-ros-camera/pull/46 (visited on 12/18/2021).

Carter, J., Schmid, K., Waters, K., Betzhold, L., Hadley, B., Mataoskky, R. and Halleran, J. (2012). *Lidar 101: An Introduction to Lidar Technology, Data, and Applications*. URL: https://coast.noaa.gov/digitalcoast/training/lidar-101.html (visited on 02/11/2021).

Cook, S. (2012). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Saint Louis: Elsevier Science & Technology. ISBN: 978-0-12-415933-4.

Davies, E. R. (2012). *Computer and Machine Vision: Theory, Algorithms, Practicalities*. 4th ed. Waltham, Mass: Elsevier.

Devianart (2021). *Robotic Arm Industrial Vector by superawesomevectors on DeviantArt*. en. URL: https://www.deviantart.com/superawesomevectors/art/Robotic-Arm-Industrial-Vector-643404944 (visited on 12/01/2021).

Druml, N., Maksymova, I., Thurner, T., Lierop, D. van, Hennecke, M. and Foroutan, A. (Sept. 2018). *1D MEMS Micro-Scanning LiDAR*.

Feng, C., Xiao, Y., Willette, A., McGee, W. and Kamat, V. R. (Nov. 2015). Vision guided autonomous robotic assembly and as-built scanning on unstructured construction sites. en. *Automation in Construction* 59, pp. 128–138. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2015.06.002. URL: https://www.sciencedirect.com/science/article/pii/S092658051500120X (visited on 10/15/2021).

GamerNexus (2021). *Specs Dictionary - CUDA Cores | GamersNexus - Gaming PC Builds & Hardware Benchmarks*. URL: https://www.gamersnexus.net/dictionary/2-cuda-cores (visited on 12/12/2021).

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. and Marín-Jiménez, M. (June 2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* 47, pp. 2280–2292. DOI: 10.1016/j.patcog.2014.01.005.

Heck, M. (Jan. 2016). Highly integrated optical phased arrays: Photonic integrated circuits for optical beam shaping and beam steering. *Nanophotonics* 6. DOI: 10.1515/nanoph-2015-0152.

Heikkila, J. and Silven, O. (1997). A four-step camera calibration procedure with implicit image correction. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1106–1112. DOI: 10.1109/CVPR.1997.609468.

Hernandez-Juarez, D., Chacón, A., Espinosa, A., Vázquez, D., Moure, J. and López, A. (Jan. 2016). Embedded Real-time Stereo Estimation via Semi-global Matching on the GPU. *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA* 80, pp. 143–153. ISSN: 1877-0509. DOI: 10.1016/j.procs.2016.05.305. URL: https://www.sciencedirect.com/science/article/pii/S1877050916306561.

Hirschmuller, H. (2008). Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE transactions on pattern analysis and machine intelligence* 30.2. Place: LOS ALAMITOS Publisher: IEEE, pp. 328–341. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1166.

Horaud, R., Hansard, M., Evangelidis, G. and Ménier, C. (2016). An overview of depth cameras and range scanners based on time-of-flight technologies. *Machine vision and applications* 27.7. Place: Berlin/Heidelberg Publisher: Springer Berlin Heidelberg, pp. 1005–1020. ISSN: 0932-8092. DOI: 10.1007/s00138-016-0784-4.

IEC (2009). *IEC 62267:2009 Railway applications - Automated urban guided transport (AUGT) - Safety requirements*.

IEC (2010). *IEC 61508:2010 Functional safety of electrical/electronic/programable electronic safety-related systems*.

– (2016). *IEC 60204 - Safety of machinery - Electrical equipment of machines*.

– (2019). *IEC TS 62998-1:2019 Safety of machinery - Safety-related sensors used for the protection of persons*.

– (2021). *IEC 62061:2021 Safety of machinery - Functional safety of safety-related control systems*.

Insider, L. (2021). *Adverse Condition Definition*. en. URL: `https://www.lawinsider.com/dictionary/adverse-condition` (visited on 12/02/2021).

ISO (2009). *ISO 10975:2009 Tractors and machinery for agriculture - Auto-guidance systems for operator-controlled tractors and self-propelled machines - Safety requirements*.

– (2014). *ISO 13482:2014 Robots and robotic devices - Safety requirements for personal care robots*.

– (2015). *ISO 13849:2015 Safety of machinery - Safety-related parts of control systems*.

– (2016). *ISO 18646:2016 Robotics - Performance criteria and related test methods for service robots*.

– (2017). *ISO 16001:2017 Earth-moving machinery - Object detection systems and visibility aids - Performance requirements and tests*.

– (2018a). *ISO 19014-1:2018 Eeart-moving machinery - Functional safety - Part 1: Methodology to determine safety-related parts of the control system and performance requirements*.

– (2018b). *ISO 25119:2018 (parts 1-4) Tractors and machinery for agriculture - Safety-related parts of control systems*.

– (2018c). *ISO 26262:2018 Road vehicles - Functional safety*.

– (2018d). *ISO18497:2018 Agricultural machinery and tractors - Safety of highly automated agricultural machines - Principles for design*.

– (2019a). *ISO 17757:2019 Earh-moving machinery and mining - Autonomous and semi-autonomous machine system safety*.

– (2019b). *ISO/PAS 21448:2019 Road vehicles - Safety of the intented functionality*.

– (2020). *ISO 3691-4:2020 Industrial trucks - Safety requirements and verification - Part 4: Driverless industrial trucks and their systems*.

JCB (2021). *JS120 Tracked Excavator Specification*. URL: https://manualzz.com/doc/27140640/js120-tracked-excavator---specification.

Kaehler, A. and Bradski, G. (2017). *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. eng. Publication Title: Learning OpenCV 3. Sebastopol: O'Reilly Media, Incorporated. ISBN: 1-4919-3799-8.

Keyence (2021). *Safety Laser Scanners | KEYENCE America*. URL: https://www.keyence.com/products/safety/laser-scanner (visited on 11/24/2021).

Kutila, M., Pyykönen, P., Holzhüter, H., Colomb, M. and Duthon, P. (2018). Automotive LiDAR performance verification in fog and rain. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1695–1701. DOI: 10.1109/ITSC.2018.8569624.

Lee, T. B. (Jan. 2019). *How 10 leading companies are trying to make powerful, low-cost lidar*. en-us. URL: https://arstechnica.com/cars/2019/02/the-ars-technica-guide-to-the-lidar-industry/ (visited on 10/26/2021).

Li, Y. and Ibanez-Guzman, J. (July 2020). Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems. *IEEE Signal Processing Magazine* 37.4, pp. 50–61. ISSN: 1558-0792. DOI: 10.1109/MSP.2020.2973615.

Liang, C.-J., Lundeen, K. M., McGee, W., Menassa, C. C., Lee, S. and Kamat, V. R. (Aug. 2019). A vision-based marker-less pose estimation system for articulated construction robots. en. *Automation in Construction* 104, pp. 80–94. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2019.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0926580518311579 (visited on 10/12/2021).

Luminar (2021). *Luminar (Nasdaq: LAZR) | Products*. en-US. URL: https://www.luminartech.com/products/ (visited on 11/30/2021).

Marwedel, P. (2021). *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Cham: Springer International Publishing AG. ISBN: 3-030-60909-X.

MathWorks (2021). *Using the Stereo Camera Calibrator App*. URL: https://se.mathworks.com/help/vision/ug/using-the-stereo-camera-calibrator-app.html (visited on 12/20/2021).

Michaud, S., Lalonde, J.-F. and Giguère, P. (2015). Towards Characterizing the Behavior of LiDARs in Snowy Conditions. *IEEE/RSJ IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles*.

Morelo, D. (2017). *GPU Programming Introduction*. en-US. URL: https://linuxhint.com/gpu-programming/ (visited on 12/12/2021).

Mujtaba, H. (Aug. 2021). *NVIDIA & Intel GPU Market Share Increased While AMD Declined In Q2 2021, GPU Shipments Hit 37% Growth Year-Over-Year*. en-US. URL: https://wccftech.com/nvidia-intel-gpu-market-share-increased-while-amd-declined-in-q2-2021/ (visited on 12/12/2021).

Nature (Oct. 1943). RADIO DETECTION AND RANGING. *Nature* 152.3857, pp. 391–392. ISSN: 1476-4687. DOI: 10.1038/152391b0. URL: https://doi.org/10.1038/152391b0.

Nvidia (2021). *Jetson Xavier NX Series*. en-us. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/ (visited on 12/21/2021).

O' Riordan, A., Newe, T., Toal, D. and Dooly, G. (Dec. 2018). *Stereo Vision Sensing: Review of existing systems*. DOI: 10.1109/ICSensT.2018.8603605.

Ouster (2021). *Ouster OS1-64 LiDAR Sensor | Level Five Supplies*. en-US. URL: https://levelfivesupplies.com/product/ouster-os1-64-lidar-sensor/ (visited on 11/23/2021).

Pegula, S. M. (2013). *An analysis of fatal occupational injuries at road construction sites, 2003–2010*. en. URL: https://www.bls.gov/opub/mlr/2013/article/an-analysis-of-fatal-occupational-injuries-at-road-construction-sites-2003-2010.htm (visited on 12/02/2021).

Pixabay (2021a). *Bulldozer Claw Machine*. URL: https://pixabay.com/vectors/bulldozer-claw-machine-robot-37379/?download (visited on 12/01/2021).

– (2021b). *Excavator side illustration*. en. URL: https://pixabay.com/vectors/excavator-heavy-machinery-truck-4562894/ (visited on 12/10/2021).

Rajput, T. (Nov. 2021). *Computer Vision System Market Size Expected to Reach USD 25.69 Billion at CAGR of 10.8%, By 2028*. en-US. URL: https://www.einnews.com/pr_news/556170731/computer-vision-system-market-size-expected-to-reach-usd-25-69-billion-at-cagr-of-10-8-by-2028 (visited on 11/12/2021).

Ravi, R. (Sept. 2020). *Nvidia CUDA Cores Explained: How are they different?* en-US. Section: Microarchitectures & Semiconductors. URL: https://www.techcenturion.com/nvidia-cuda-cores/ (visited on 12/13/2021).

Robotics, B. (2021). *Built Robotics transforms heavy equipment into autonomous robots for...* en-US. URL: https://www.builtrobotics.com/ (visited on 11/30/2021).

Rusu, R. B., Marton, Z. C., Blodow, N., Dolha, M. and Beetz, M. (Nov. 2008). Towards 3D Point cloud based object maps for household environments. *Semantic Knowledge in Robotics* 56.11, pp. 927–941. ISSN: 0921-8890. DOI: 10.1016/j.robot.2008.08.005. URL: https://www.sciencedirect.com/science/article/pii/S0921889008001140.

Sabet, K. (2016). *RF Tutorial Lesson 17: Simulating a Frequency-Modulated Continuous-Wave (FMCW) Radar System - Emagtech Wiki.* URL: http://www.emagtech.com/wiki/index.php/RF_Tutorial_Lesson_17:_Simulating_a_Frequency-Modulated_Continuous-Wave_(FMCW)_Radar_System (visited on 11/09/2021).

Sandvik (2021). *AutoMine® for Trucks*. en. URL: https://www.rocktechnology.sandvik/en/campaigns/automine-for-trucks/ (visited on 10/26/2021).

Shirokuma (2021). *Cupoch GitHub library page*. URL: https://githu.com/neka-nat/cupoch (visited on 12/17/2021).

SICK (Dec. 2021). *outdoorScan3 Safety laser scanner*. URL: https://cdn.sick.com/media/docs/3/63/763/product_information_outdoorscan3_safety_laser_scanner_en_im0082763.pdf.

Stachniss, C. (2009). *Robotic Mapping and Exploration*. 1st ed. 2009. Springer Tracts in Advanced Robotics, 55. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 3-642-01097-0. DOI: 10.1007/978-3-642-01097-2.

Team, T. (July 2021). *What's Happening With Velodyne Lidar Stock?* en. Section: Markets. URL: https://www.forbes.com/sites/greatspeculations/2021/10/07/whats-happening-with-velodyne-lidar-stock/ (visited on 11/23/2021).

Techpowerup (2021). *NVIDIA Jetson Xavier NX GPU Specs*. en. URL: https://www.techpowerup.com/gpu-specs/jetson-xavier-nx-gpu.c3642 (visited on 12/13/2021).

Tiusanen, R., Malm, T. and Ronkainen, A. (2020). An overview of current safety requirements for autonomous machines – review of standards. *Open Engineering* 10.1, pp. 665–673. DOI: 10.1515/eng-2020-0074. URL: https://doi.org/10.1515/eng-2020-0074.

Volvo (May 2021). *Electric Construction Equipment vs Diesel Performance | Volvo CE*. en-US. URL: https://volvoceblog.com/electric-construction-equipment-vs-diesel-performance-comparisons/ (visited on 11/30/2021).

Vosselman, G. and Maas, H.-G. (2010). *Airborne and Terrestrial Laser Scanning*. English. Dunbeath: Whittles Publishing. ISBN: 978-1-904445-87-6. (Visited on 11/03/2021).

Wang, W., Chi, H., Zhao, S. and Du, Z. (July 2018). A control method for hydraulic manipulators in automatic emulsion filling. en. *Automation in Construction* 91, pp. 92–99. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2018.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0926580517302789 (visited on 10/14/2021).

Yoo, H. W., Druml, N., Brunner, D., Schwarzl, C., Thurner, T., Hennecke, M. and Schitter, G. (2018). MEMS-based lidar for autonomous driving. *Elektrotechnik und Informationstechnik* 135.6. Place: Vienna Publisher: Springer Vienna, pp. 408–415. ISSN: 0932-383X. DOI: 10.1007/s00502-018-0635-2.

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11, pp. 1330–1334. DOI: 10.1109/34.888718.