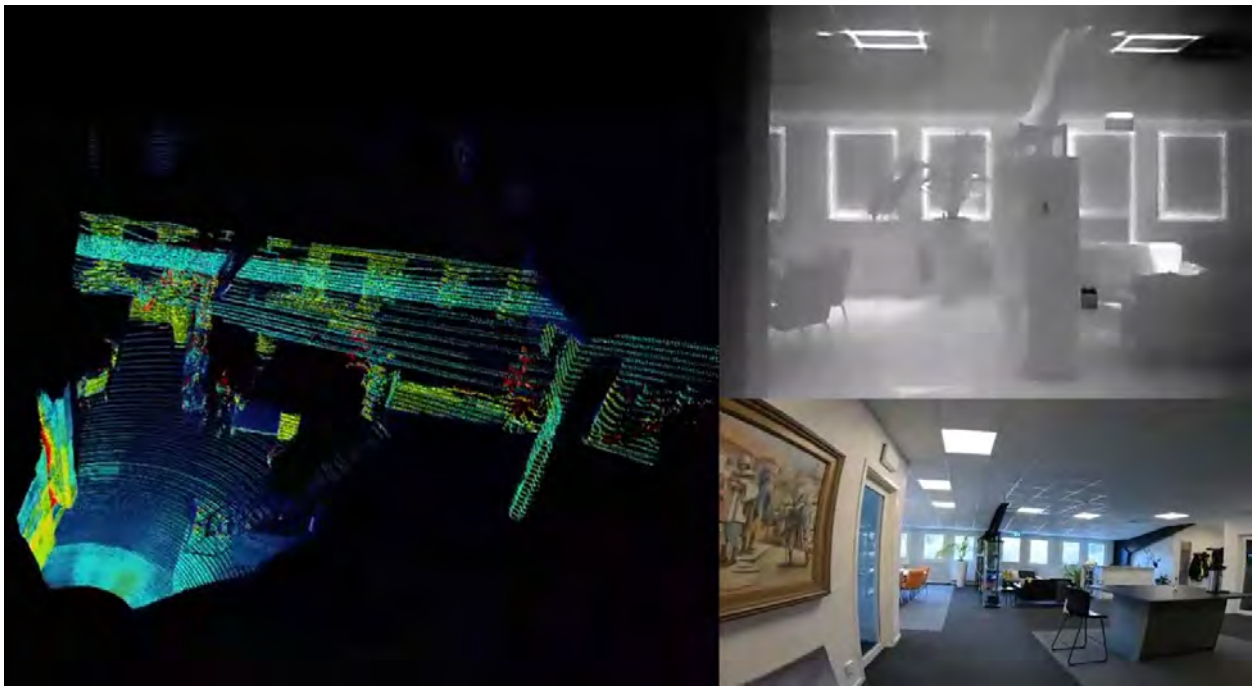


Enhancing Emergency Operations with Sensor Fusion and AI

Fusing LiDAR and Thermal Vision with Deep Learning for Scene Understanding in Smoke-Filled Environments

Master's Thesis in Complex Adaptive Systems and Systems, Control and Mechatronics

Joel Carlsson and Denis Grahovic

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Enhancing Emergency Operations with Sensor Fusion and AI

Fusing LiDAR and Thermal Vision with Deep Learning for Scene Understanding in Smoke-Filled Environments

Joel Carlsson
Denis Grahovic

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of physics
Division of material physics
Chalmers University of Technology
Gothenburg, Sweden 2025

Enhancing Emergency Operations with Sensor Fusion and AI
Fusing LiDAR and Thermal Vision with Deep Learning for Scene Understanding in Smoke-Filled Environments
Joel Carlsson
Denis Grahovic

Supervisor: Magnus Karlsteen, Department of physics
Examiner: Magnus Karlsteen, Department of physics

Industrial Supervisor: Pau Mallol, UA Vision Nordic AB
Industrial Supervisor: Andreas Eriksson, UA Vision Nordic AB

Master's Thesis 2025
Department of physics
Division of material physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An overview of the proposed system architecture, meant to be used in the final prototype

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Enhancing emergency operations with sensor fusion and AI
Fusing LiDAR and Thermal Vision with Deep Learning for Scene Understanding in Smoke-Filled Environments
Joel Carlsson
Denis Grahovic
Department of physics
Chalmers University of Technology

Abstract

In recent years, society has experienced rapid technological advancements across all fields, pushing the boundaries of what was once thought possible. Emergency services, such as fire departments, have largely been left behind. Operating in hazardous, smoke-filled environments places firefighters at extreme risk, where no compromises can be made on the tools and technologies that support them. This project addresses that gap by exploring the development of a Multi-Environment Camera (MEV-Cam) system designed to aid firefighters during rescue operations in visually impaired environments. The study focuses on the fusion of LiDAR and thermal imaging, investigating how the sensors can leverage their strengths and complement each others weaknesses to provide more reliable scene understanding.

To address these challenges, four tailored datasets were created to simulate the intended application scenario, each containing synchronized data from the sensors. Fusion based models were developed for enhanced 3D visualization. For scene understanding, deep learning models were investigated to both LiDAR and thermal data, using architectures based on YOLO and PointNet. Additionally, pose estimation was explored using monocular visual odometry. A major focus of the project was on enhancing the raw collected data to ensure the sensors could be effectively used in this system. In parallel, a hardware prototype was developed to enable efficient data collection in real world environments.

Results show that airborne particles significantly degrade LiDAR performance, while thermal sensors remain relatively unaffected. However, sensor fusion can compensate for these limitations. Deep learning models demonstrated the ability to accurately interpret scene structure under degraded conditions. After effective data enhancement, the MEV-Cam system showed improved performance across its various modules.

While these results highlight the promise of the technology, they also reveal current limitations and suggest several innovative directions for future work. This project marks the beginning of a new project to support life saving operations, by utilizing the latest sensor technology and AI.

Keywords: Sensor fusion, Emergency operation, Object detection, LiDAR sensor, Thermal sensor, Machine learning, Visual odometry, 3D segmentation

Acknowledgements

We would like to express our sincere gratitude to UA Vision Nordic AB for providing us with the opportunity to carry out this thesis project. A special thanks goes to our industrial supervisor, Pau Mallol, for his continuous support and valuable insights. Our thanks also go to Andreas Eriksson for his constant availability and willingness to help whenever needed. We are also deeply thankful to our academic supervisor and examiner, Magnus Karlsteen, for his academic guidance, constructive feedback, and commitment to our progress.

Lastly, we would like to extend our heartfelt thanks to Chalmers University of Technology and the friends we've met along our studies. Together, we have shared both challenges and triumphs, creating lifelong memories that we will always carry with us.

Contents

List of Figures

List of Tables

1

Introduction

In recent years, society has experienced rapid technological advancements across all fields, pushing the boundaries of what was once thought possible. One area that has seen significant progress and growing prioritization is LiDAR technology. Widely used in the automotive sector, LiDAR plays a crucial role in enabling self-driving vehicles by generating high-precision 3D visualizations of the surrounding environment. However, despite major breakthroughs, several critical challenges remain. One of the most pressing issues is LiDAR's reduced effectiveness in foggy and smoke-filled environments, which limits its usability in certain real-world scenarios.

LiDAR has become the go-to technology for high-accuracy 3D mapping and is applied in various industries, including archaeology, topography, and construction. However, one field where LiDAR remains underutilized is emergency response services. Firefighters, in particular, face extreme challenges due to limited visibility in smoke-filled environments, significantly slowing down rescue operations and putting lives at risk. Despite the urgent need for a visually aiding tool in such situations, the primary tool currently available is infrared (IR) cameras, also known as thermal cameras. While IR can detect heat signatures, it lacks the ability to generate a comprehensive 3D representation of the environment.

This is where a Multi Environment Camera (MEV-Cam) comes into play. The MEV-Cam leverages a LiDAR-IR fusion model to enhance LiDAR's 3D performance in smoke-filled environments. Additionally, by integrating AI-powered object detection into the MEV-Cam, the system can more effectively locate individuals in life-threatening situations, improving rescue efficiency. However, despite this advancement, challenges still remain. LiDAR struggles with light scattering in dense smoke or fog, leading to significant information loss. Meanwhile, thermal cameras face limitations in depth perception since they only produce 2D thermal images, and they also have difficulty distinguishing between objects with similar temperatures.

This project aims to address these limitations by improving the sensor fusion process within the MEV-Cam, enabling real-world deployment in rescue operations. By refining data integration techniques and enhancing object detection capabilities, the goal is to create a foundation for a system that overcomes the weaknesses of both LiDAR and IR technology. A successful implementation would revolutionize firefighting operations in the future by navigating hazardous environments with greater accuracy and saving more lives in emergency situations.

1.1 Background

Research has focused on improving visual perception in smoke-filled environments. Prior work has explored the use of LiDAR and thermal cameras, with a focus on sensor fusion techniques. Various multi sensor fusion algorithms have been investigated, including probabilistic methods, evidential fusion, neural networks, decision trees, and heuristic approaches [10], [11], [12]. Most of the relevant studies have used stereo thermal camera setups to enable depth estimation through triangulation [12], [13]. Other studies have explored monocular thermal cameras in combination with additional sensors such as IMUs or depth sensors to support depth estimation [14], [15]. These experiments demonstrate that missing data from LiDAR can effectively be complemented by combining 3D points derived from thermal sensors.

Despite these efforts, the fusion of LiDAR with only monocular thermal sensor remains a challenging task, particularly due to the difficulty of estimating depth from only thermal images which still is under active research. Monocular Depth Estimation (MDE) using thermal sensors inherits challenges of RGB based MDE, with the additional complexity of low contrast, high noise ratio, and a lack of texture and colour information [16]. With the emergence of new technologies, AI based approaches for estimating depth from monocular thermal images are being increasingly explored [17].

Another relevant and highly researched area is object detection. Using thermal images for classification is a well established technique, widely used in industries such as surveillance and industrial monitoring [18]. With the recent development in AI, new models with increased performance have reached remarkably accuracy [19, 20]. Similarly, object detection and segmentation using 3D point cloud data have become increasingly common, enabling both object-wise classification and point-wise segmentation [21], [22]. These techniques are often implemented in applications where awareness about the surroundings, such as robotics and autonomous navigation, are important.

1.2 Aim

The aim of this master's thesis is to investigate how LiDAR and thermal imaging sensors can be utilized to overcome the limitations of LiDAR performance in environments with airborne particles. How can a MEV-camera be used for navigation and visualization through smoke-filled structures during a fire rescue, where vision is impaired? More specifically, the goal is to investigate if the combination of a LiDAR together with a single thermal camera is enough to properly estimate the surroundings in smoke filled environments were humans eyesight is not enough and also if object detection can be integrated for even better aid for firefighters. The final aim is defined as follow:

Investigate how the sensors LiDAR and IR can be enhanced, and combined to better estimate the surroundings and identifying the environment in cases where humans vision is impaired.

1.3 Delimitations

In order to establish a clear and manageable project, the following delimitations have been defined. These boundaries were set to ensure that the project remains feasible within the given time frame and available resources, while still allowing for meaningful experimentation and analysis.

- Due to resources at hand, this project will not implement the hardware but instead focus will be on the software. The company associated with the project will in parallel provide the necessary hardware.
- Due to the time constraint, the project will not implement any real-time solutions. Additionally due to the time constraint, a fully connected system will not be developed and all modules will be developed independently.
- The project will use cold smoke (theater smoke) rather than real smoke environments during tests, due to safety considerations and practical constraints.
- The sensors used in the project will be one LiDAR, one Thermal camera and one RGB camera. The sensor fusion will only be made between the LiDAR and IR.

1.4 Scope and Research Objectives

LiDAR (Light Detection and Ranging) and IR (Infrared) sensors each have strengths and limitations in environmental perception. LiDAR excels at generating precise 3D maps, but its performance is affected by smoke, fog, and other visual obstructions. IR cameras, on the other hand, can detect heat signatures in complete darkness and penetrate smoke, yet they suffer from lower resolution and sensitivity to surface reflections, which can obscure details like edges and depth. The key challenges in this project involve enhancing LiDAR's resilience to distortions through data filtering and optimization, integrating the IR sensor to compensate for the LiDAR's limitations, and designing an effective information flow that determines how data from both sensors should be processed and merged for robust environmental estimation. Additionally, the importance of understanding and identifying the environment can not be understated. Thus, another challenge is to support the firefighters in rescue operations by highlighting the necessity of using AI classification.

By investigating and finding solutions for these challenges, this project can enhance the ability for MEV-cameras to operate in fog and smoke, making it a practical solutions for new applications. From the sensors' limitations, three objectives have been constructed and will be the focus areas for the master thesis.

- Investigate how the sensors (LiDAR and IR) can be enhanced in areas with poor visibility individually but most importantly combined to complement each others weaknesses. Is the information enough to properly estimate the surroundings in cases where humans vision is impaired?
- Using the sensor data, examine how the combination of these sensors can assist in the classification and identification of surrounding objects in areas with poor visibility, with the support of AI. Investigate if the information from the sensors is enough for the AI based decision making to sufficiently detect the surrounding objects.
- Investigate and lay the foundation for the system architecture. Identify the key modules needed for a functioning system and explore their ability to collaborate in solving the problem.

2

Theory

This section presents the theoretical background necessary to understand the project. It begins with an overview of general concepts and gradually delves into the specific algorithms utilized throughout the work. In several parts, methodological choices have been made based on the literature review presented.

2.1 Light Scattering

The propagation of light and laser beams through the atmosphere can be significantly affected by disturbances such as fog and smoke. These atmospheric aerosols cause scattering and absorption of light, which in turn degrades the performance of laser-based sensing systems [23],[24]. Aerosols such as smoke and fog consist of small particles in the air. When light or laser beams encounter these particles, scattering occurs. Two primary forms of scattering are relevant in such cases: Mie scattering and Rayleigh scattering [25]. A visualization of the different scattering types can be seen in Figure 2.1, and they apply as follows:

- **Mie scattering:** This occurs when the size of the particles is comparable to the wavelength of the light or laser beam (common in aerosols like smoke and fog). It scatters light in many directions, reducing the intensity of the beam along its original path, and can produce multiple false returns in laser-based sensors due to the broad angular scattering [25].
- **Rayleigh scattering:** This occurs when the particles are much smaller than the wavelength of the light or laser beam (common in clear air). It results in more uniform scattering, primarily in the forward and backward directions, and is generally less disruptive to optical sensing systems [25].

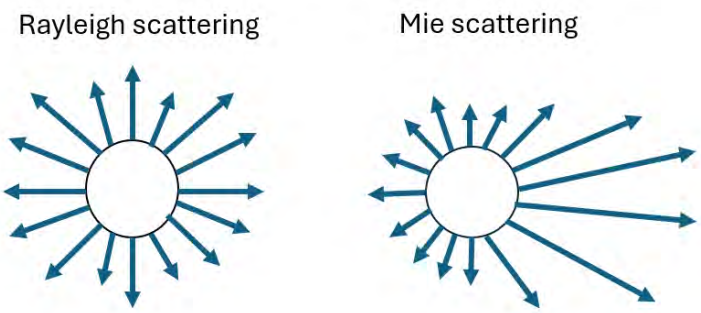


Figure 2.1: Rayleigh and Mie scattering

Absorption is another problem that affects the propagation of light in the presence of aerosols, such as fog and smoke. The particles within these aerosols can absorb specific wavelengths of light, depending on their chemical composition. This absorption results in a direct loss of energy from the light beam, which reduces its effective range and signal strength. For optical sensing systems, this leads to a loss of information and reduced performance [23], [24], [25].

2.2 Light Detection and Ranging (LiDAR)

LiDAR is a remote sensing technology that uses laser pulses (light waves) to measure distances and generate detailed 3D maps of the environment. LiDAR operates in shorter wavelength than RaDAR usually in the ultraviolet, visible and infrared region of the electromagnetic spectrum [26]. A LiDAR sensor emits rapid pulses of laser light (typically between the wavelength of 900-1500 nm). These pulses travel at the speed of light and spread in a predetermined pattern. When the laser pulse hits an object (e.g. ground, tree, building, or vehicle), it reflects back toward the LiDAR sensor. The strength of the reflected pulse depends on the object's surface properties (colour, texture, and reflectivity) [27].

The system determines the distance of the object by leveraging time-of-flight (ToF) methodology. It measures the time taken for each laser pulse to return to the sensor and calculates the distance [d] by.

$$d = \frac{ct}{2} \tag{2.1}$$

where c is the speed of light and t is the flight time of the laser pulse. Since the direction of each emitted pulse is precisely known, the object's position can be accurately identified [5], [28].

One of the weaknesses of a LiDAR system is their sensitivity to harsh environmental conditions such as fog or rain. When LiDAR pulses travels through these conditions,

it quickly weakens due to absorption and scattering caused by water droplets. This greatly reduces the system's ability to detect objects at long distances. Additionally, scattered particles create many false detections by reflecting light back to the sensor, making the system less reliable [29]. Similar results have been shown in smoke filled environments, where the LiDAR has shown to either scatter or absorb its emitted signal, leading to reduced detection accuracy and range [28], [30].

2.3 Infrared (IR)/ Thermal Camera

All objects emit thermal energy, and the amount emitted depends on their temperature. Infrared (IR) cameras utilize this property by using IR detectors to measure the infrared radiation emitted from an object. IR radiation, also known as infrared light, is a type of electromagnetic radiation similar to visible light but with longer wavelengths (3000nm - 15000nm). By detecting this radiation, IR cameras convert it into an electronic signal, which is then processed to create a visual representation of temperature differences, allowing the human eye to perceive heat variations [31].

The main strength of an IR camera is its ability to provide visibility in low-visibility environments. Since the image from an IR camera is based on infrared radiation, it is not affected by darkness. However, in smoky or foggy environments, the IR camera's performance can be influenced depending on the temperature and density of the smoke or fog. If the smoke or fog is cooler than the observed object, it will likely cause the camera to measure a lower temperature due to the absorption and scattering of infrared radiation. Similarly, if the smoke or fog is warmer, its particles will emit additional infrared radiation, leading to an overestimation of the object's temperature. The negative effect is greater with fog compared to smoke since the ambient temperature radiation causes temperature changes in the fog droplets, while smoke particles are less affected. Despite these effects, IR cameras remain effective in these environments, though their performance may be slightly reduced in terms of accuracy and detection range [32],[24].

2.4 Computer Vision

This section provides a overview of all computer vision-related theoretical concepts that have been applied throughout the project. It outlines the foundational principles, algorithms, and techniques which are used in the implementation, offering the necessary context for understanding.

2.4.1 IR Data Preprocessing

IR images are very susceptible to noise and therefore methods to enhance the images need to be applied. There are two very common methods which are used together to enhance the image contrast. These are: Gaussian filtering and Contrast Limited Adaptive Histogram Equalization (CLAHE) [33], [34].

The Gaussian filter is used to smooth the image by averaging the pixel values in a local neighbourhood and that way suppresses high-frequency noise. The filter operates using a convolutional kernel, the size can be adjusted according to the image dimensions and computational needs [35], [34]. The filtering process is performed by computing the gaussian kernel and is defined as follows:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

Where G is the value of the gaussian kernel, σ is the standard deviation controlling the spread (blur strength) and (x, y) are spatial coordinates relative to the center of the kernel.

CLAHE is used to enhance the contrast of an image in a localized and adaptive manner. The image is first divided into small, non-overlapping regions called *tiles*, typically of size 8×8 pixels. For each tile, a histogram of pixel intensities is computed to capture the local distribution of brightness values. To prevent noise from being overly amplified, a clip limit is applied to the histogram, any bin that exceeds this threshold is clipped, and the excess is redistributed uniformly across all bins [33].

Next, a cumulative distribution function (CDF) is computed from the clipped histogram and used to remap the pixel intensities. The new intensity $[I']$ value for each pixel is given by:

$$I' = \frac{\text{CDF}(I) - \text{CDF}_{\text{min}}}{(\text{CDF}_{\text{max}} - \text{CDF}_{\text{min}})} \cdot (L - 1) \tag{2.3}$$

where I is the original pixel intensity, $\text{CDF}(I)$ is the cumulative frequency at intensity I, CDF_{min} is the smallest non-zero value in the CDF, CDF_{max} is the largest value in the CDF, and L is the number of possible intensity levels (typically 256). This mapping ensures that the histogram is spread out more evenly, improving contrast within each tile.

After all tiles have been processed, bilinear interpolation is applied across tile boundaries. For each pixel at the edge between four adjacent tiles, its final value is computed as a weighted average of the values from those four tiles, based on the pixel's relative distance from each tile center. This ensures smooth transition between regions in the enhanced image [33].

2.4.2 3D-2D Projection

Projection of 3D points into a 2D frame is a standard technique used in many computer vision applications. By using the camera's intrinsic and extrinsic parameters, a set of 3D points in world coordinates can accurately be transformed into 2D image points. This can be achieved by using the following equation:

$$x = PX \tag{2.4}$$

Where X is the 3D points in homogenous coordinates $X = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^\top$ and P is the camera matrix for a specific camera and is on the form $P = K \times \begin{bmatrix} R & t \end{bmatrix}$. Here R and t are the rotation and translation of the camera with respect to the world coordinate frame. K is the camera intrinsic matrix describing the properties of the camera and is on the form [36]:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

where:

- f_x, f_y: focal lengths in pixels,
- c_x, c_y: principal point offsets,
- s: skew coefficient (often 0).

After x is obtained, it has to be turned into pixel coordinates. The form of the projected points x is $x = \begin{bmatrix} u & v & w \end{bmatrix}^\top$. This is transformed to pixel coordinates by setting the depth (z-value) to one so that all the points are on the same 2D frame. The final 2D projected points thus become [36]:

$$x = \begin{bmatrix} \frac{u}{w} \\ \frac{v}{w} \\ 1 \end{bmatrix} \tag{2.6}$$

2.4.3 2D to 3D Backprojection

The process of projecting 2D points back into 3D is the inverse of the 2D projection described above. The 3D points can be computed by using the pixel coordinates together with a known depth Z and the camera's intrinsics parameters. The points are obtained by using the following equations [36]:

$$\begin{aligned} X &= \frac{(u - c_x) \cdot Z}{f_x} \\ Y &= \frac{(v - c_y) \cdot Z}{f_y} \\ Z &= Z \end{aligned} \tag{2.7}$$

2.4.4 Feature Extraction

Detecting features is a key part of many image based algorithms. By identifying edges and corners in an image, it is possible to create complex image based systems, such as Visual Odometry, SLAM and AR/VR. The main methods used to detect features are ORB, SIFT and AKAZE [37], [38], [39]. In Table 2.1 the strengths and weaknesses of all three models are presented.

Table 2.1: Feature detection methods and its pros and cons.

Methods	***Pros***	***Cons***
ORB	*- Low computational cost.* *- Low memory usage* *- Suitable for real-time*	*- Sensitive to significant affine transformations.* *- Less robust to large scale changes.*
SIFT	*- Highly robust to scale, rotation, and noise.* *- Good performance in low-texture and low-light scenes.* *- Accurate and repeatable keypoints*	*- Computationally expensive.* *- larger memory usage* *- Unsuitable for real-time.*
AKAZE	*- Robust to noise and blur.* *- Good compromise between speed and performance* *- Better edge preservation via non-linear scale space*	*- Slower than ORB.* *- Less mature – fewer tuning resources/libraries* *- Can underperform in very high-frequency textures.*

In this project, ORB and SIFT will be used and not AKAZE. SIFT is used because of it's high accuracy to test the limits of the developed systems. ORB will be used to test performance for a computationally effective system and evaluate future real-time possibilities. AKAZE, which offers a balance between speed and robustness, does not align with the specific goals of either high accuracy or real-time efficiency and is therefore excluded [37], [38], [39].

Oriented FAST and Rotated BRIEF (ORB) is a computationally efficient and robust keypoint detection method [37]. It uses the FAST (Features from Accelerated Segment Test) algorithm to detect keypoints. This is done by evaluating the pixel intensity of a circle with 16 pixels around a candidate pixel and the pixels which are way darker or brighter than the candidate pixel are defined as corners. To enhance robustness, ORB also uses Harris corner response which computes the gradients of the 16 pixels and creates a second-moment matrix $[M]$:

$$M = \sum_{u,v \in \text{window}} \begin{bmatrix} I_x^2(u,v) & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y^2(u,v) \end{bmatrix} \tag{2.8}$$

Where I is the image intensity function. Using this matrix, the eigenvalues are calculated and the Harris response function R is used:

$$R = \det(M) - k \cdot (\text{trace}(M))^2 \tag{2.9}$$

where $\det(M) = \lambda_1 \lambda_2$ and $\text{trace}(M) = \lambda_1 + \lambda_2$.
The R is evaluated as follows:

- Large and positive R = corner
- Negative R = edge
- R ≈ 0 = flat region

After the keypoints are detected, an orientation is added to each keypoint to prevent possible issues due to rotation of the image. The rotation is added by using the intensity centroid method. This method computes the total intensity in a patch of pixels and the total intensity in x and y directions. It then computes the centroid $[C]$ as follows:

$$C_x = \frac{m_{10}}{m_{00}}, \; C_y = \frac{m_{01}}{m_{00}} \tag{2.10}$$

Where m is the moments used to describe the distribution of pixel intensities. The angle $[\theta]$ is then obtained by computing the following:

$$\theta = \arctan\left(\frac{C_y - y}{C_x - x}\right) \tag{2.11}$$

After the keypoints are obtained, BRIEF (Binary Robust Independent Elementary Features) is used to get the descriptors. It compares the intensity values of pre-selected pixel pairs within a local patch around the keypoint. For each pair, a binary value is set based on which pixel is brighter. This results in a compact binary string representing the local appearance [37].

Scale-Invariant Feature Transform (SIFT) is a highly robust method for keypoint detection and description [38]. It detects keypoints by implementing difference of gaussians (DoG). SIFT constructs a scale space by progressively blurring the image with gaussian filters of increasing standard deviation σ. Keypoints are detected by computing the DoG $[D]$ between adjacent blurred images in the scale space:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.12}$$

Where L is computed by using the following:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.13}$$

$G(x, y, \sigma)$ is a gaussian kernel at scale σ, $I(x, y)$ is the input image and $*$ signifies convolution. Each pixel is compared to it's 26 nearest neighbours and if the pixel is

a local extremum it is chosen as a keypoint candidate.

After the keypoints have been detected, they are further refined so that only stable keypoints remain. This is done by computing the hessian of (2.12) and if the eigenvalues of the hessian exceed a chosen threshold, they are removed. When only stable keypoints remain, they are given an orientation based on the gradient distribution. The gradient magnitude [m] and orientation [θ] is as follows:

$$m(x,y) = \sqrt{(L_x)^2 + (L_y)^2}, \quad \theta(x,y) = tan^{-1}(\frac{L_y}{L_x}) \tag{2.14}$$

Finally, the descriptors are obtained by analysing the gradient distribution in a region around each keypoint. The patch is rotated to align with the keypoints orientation and divided into 4x4 cells. For each cell, an 8-bin histogram of gradient directions is computed, resulting in a descriptor with 128 dimensions. This way the descriptors become robust to scale, rotations and variations in the image [38].

2.4.5 IR Edge Detection

Canny edge detection is a method used in thermal imaging to identify and extract meaningful edges whilst also minimizing noise. The model consists of the following five steps [40]:

First reduce the likelihood of detecting false edges caused by image noise, a noise reduction step is applied at the beginning of the process. the noise reduction step for this case is the data preprocessing described above.

The next step is to find the intensity gradient of the image. The gradient in the x and y directions is obtained by convolving the image with Sobel kernels, which approximates the first derivative of the intensity in each direction. This is then used to highlight the areas with the biggest changes in intensity, which in most cases are edges [41]. The horizontal and vertical gradients [G] are defined as:

$$G_x = \frac{\partial I}{\partial x}, \quad G_y = \frac{\partial I}{\partial y} \tag{2.15}$$

Where I is the image intensity function. Using (2.15), the direction and magnitude of the gradient can be computed as follows:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{2.16}$$

Where G is the intensity magnitude and θ is the direction of the gradient.

After this, non-maximum suppression needs to be applied. This method thins out wide lines in order to get precise edges in the gradient direction. For each pixel, the previously computed gradient direction is used to find the two nearest neighbours along that direction. If the current pixel is not the local maxima, it is set to zero. This ensures that the edges are one pixel wide [41].

Once all pixels have been processed, it needs to be decided if they are true edges or not. This is done by implementing double thresholding. Two thresholds are chosen: an upper threshold T_u and a lower threshold T_l. Each pixel is compared to these thresholds and they are classified as follows:

$$\begin{cases} \text{strong edge} & \text{if } x > T_u \\ \text{weak edge} & \text{if } T_u \geq x > T_l \\ \text{non-edge} & \text{if } x \leq T_l \end{cases}$$

A strong edge indicates that the pixel is very likely an edge whilst a weak edge is not as certain. The final step in Canny edge detection is hysteresis tracking. This method tracks the connections of the weak edges and if they are connected to a strong edge, they are classified as an edge. If this is not the case they are discarded [35], [41].

These five steps in the Canny algorithm are carefully designed to maximize edge detection performance while minimizing the influence of noise.

2.4.6 Depth Mapping and Sensor Fusion

Fusing 3D LiDAR points with 2D image data presents several challenges, primarily because 2D images do not contain depth information. As such, a depth map must be generated to assign approximate depth values to each pixel in the image based on corresponding 3D LiDAR data [36, 42].

To construct the depth map, LiDAR points are transformed into the camera coordinate system using the extrinsic calibration parameters. These points are projected onto the 2D image plane using the camera's intrinsic parameters as shown in (2.5). Any projected points that fall outside the image boundaries are discarded. The depth (z-coordinate) of the remaining points is mapped to the corresponding pixel locations, creating a sparse depth map.

To build a sparse depth map, the depth values are assigned to their corresponding valid LiDAR points (points within the image frame). If more than one LiDAR point is on the same pixel, then the closest LiDAR points is kept (smallest depth value). This results in an image where only a small amount of pixels have assigned depth values.

The sparse depth map needs to be densified in order to assign depth values to all pixels within the image. To densify this map, image inpainting techniques are used to estimate missing depth values. Two commonly used methods are [42], [43]:

- Telea inpainting, which obtains information using weighted averages of nearby pixels.

- Navier-Stokes inpainting, which uses the gradient of surrounding regions to compute values and tends to preserve edge structures more accurately.

Although the Navier-Stokes approach is generally slower than Telea's, it offers better accuracy in depth reconstruction and is therefore chosen for this project. The resulting dense depth map allows each image pixel to be associated with a depth value, enabling the projection of image features into 3D space [43].

2.5 Visual Odometry

This section presents the computer vision theory specifically relevant to visual odometry. It provides an explanation for mathematical models, and algorithms used to estimate camera motion from image sequences.

2.5.1 Feature Detection and Tracking

Detecting and tracking features is a key part of Visual Odometry (VO) and there are many different methods which can be used for this. The most widely used methods are descriptor based methods like ORB/SIFT/AKAZE and continuous methods like Kanade–Lucas–Tomasi (KLT) feature tracking [36].

For this project, KLT tracking was used in addition to SIFT and ORB, because of it's effectiveness and low computational cost. Compared to the descriptor based methods, KLT tracking doesn't need to recompute the features unless it reaches a threshold. This makes it very effective for straight paths and slow turns which this project mainly focuses on. The descriptor based methods need to compute features in each frame, making them more computationally demanding compared to the KLT tracking [36], [37], [38].

KLT feature tracking is a widely used algorithm for tracking feature points between consecutive frames in an image sequence. It assumes that an image undergoes constant motion and that the intensity of pixels remain constant. This way the following optical flow constraint can be determined [36]:

$$I_x \cdot v_x + I_y \cdot v_y + I_t = 0 \tag{2.17}$$

Where I_x, I_y are the spatial image gradients, I_t is the temporal intensity change and v_x, v_y are the pixel velocities in the x and y direction. Since there is only one equation and two unknowns, it is solved using the least squares algorithm in a local window around each pixel.

The KLT algorithm then tracks the point in the next frame by repeating the process and estimating how far each patch has moved. But not all pixels can be tracked reliably. To find good features to track, the KLT algorithm uses the second-moment matrix:

$$M = \sum_{u,v \in \text{window}} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{2.18}$$

By estimating the eigenvalues of this matrix, the features can be classified as follows [36]:

- The patch is a corner if both eigenvalues are large.
- The patch is an edge if one eigenvalue is small
- The patch is a flat surface if both eigenvalues are small.

This way the KLT algorithm finds and tracks good features efficiently.

2.5.2 Pose Estimation

In VO, the main goal is to estimate the pose (translation and rotation) of the camera in use. This is done in by first estimating the essential matrix $[E]$ for two consecutive frames [36], [44]:

$$x_2^\top E x_1 = 0 \tag{2.19}$$

Where x_1 and x_2 are the pixel coordinates for camera one and two respectively. The essential matrix contains the relation between two different cameras, and it is therefore possible to extract the change in pose between two frames from it. The essential matrix is obtained by using the Random Sample Consensus algorithm (RANSAC). The algorithm uses the following steps.

1. Randomly select a minimum number of points (five for the essential matrix) to estimate the model.
2. Fit the model by using these points.
3. Test all datapoints against the model, all points that fall below a set error threshold are considered inliers.
4. Count the number of inliers and save the model if the number of inliers is the highest so far.
5. Repeat for either a set number of iterations or until a set number of inliers are found.
6. Re-compute the model using all inliers after all iterations are finished.

This results in a high quality estimation of the essential matrix and the rotation and translation can be recovered from the following relation [36], [44]:

$$E = [t]_\times R \tag{2.20}$$

Where $[t]_\times$ is a skew-symmetric matrix. After the relative pose is obtained, it can be further enhanced using some form of true positional data. Since a single camera can't estimate any depth values, triangulation is applied. It uses the following relation:

$$x_1 = P_1 X,\ x_2 = P_2 X \tag{2.21}$$

Where x_1 and x_2 are the 2D coordinates, P_1 and P_2 are the camera matrices for frame 1 and 2 and X is the 3D homogeneous coordinates from camera 1 and 2. From this, a linear system can be set up:

$$AX = 0 \tag{2.22}$$

Where A is a 4×4 matrix constructed from x_1, x_2, P_1, P_2 respectively.
This system can be solved by using Singular Value Decomposition (SVD):

$$A = U\Sigma V^\top \tag{2.23}$$

Here, U is an orthogonal matrix, Σ is a diagonal matrix and $V^\top$ is a transpose of an orthogonal matrix. By using this, the 3D coordinates X can be obtained. By having some form of true data, for example a set recording height, this can be used to scale the data in order to get a more accurate pose estimation [36], [44].

2.6 Filter

This section outlines and explains the algorithms employed in the project for filtering LiDAR data. It provides an overview of the techniques used to remove noise, outliers, and irrelevant points.

2.6.1 Distance Based Filter

Distance based filtering, also known as filtering on Euclidean distance, is a technique that evaluates the distance between data points and a set reference point by calculating the straight-line distance between them in a 3 dimensional space. By establishing a threshold value, the filter can effectively distinguish between relevant data and outliers, ensuring that only points within a certain Euclidean distance are considered for further analysis [45].

Let $p, q \in \mathbb{R}^3$ represent 2 different points in the 3 dimensional data, then the Euclidean distance is given by:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \tag{2.24}$$

2.6.2 Field based filter

Field based filtering refers to the process of selecting or discarding data points based on the values of specific attributes, or fields, collected by a sensor [46]. Each sensor typically gathers various types of information, and these different data types are stored in their respective fields. For instance, a LiDAR sensor, captures several key pieces of information, including the reflection magnitude of a point, the intensity of the return signal, as well as the spatial coordinates in the form of x, y, and z positions [5].

This method of filtering is particularly valuable in applications where it is necessary to reduce the amount of irrelevant or noisy data, concentrate on objects of interest, or enhance the efficiency of subsequent data processing stages [47]. Field-based filtering achieves this by applying conditions such as thresholds, range checks, or sorting criteria to the various fields collected by the sensor.

To formalize this, each data point collected by the sensor can be represented as:

$$P_i = \{x_i, y_i, z_i, f_i^1, f_i^2, \ldots, f_i^m\}$$

were f_i^j represents the value of the j-th field for point i, and m is the total number of fields (attributes),

Field-based filtering involves defining a filtering function $F(P_i)$ that evaluates whether a point should be kept or discarded based on conditions applied to these fields. Mathematically, this can be expressed as:

$$F(P_i) = \begin{cases} 1, & \text{if the conditions on the fields are satisfied} \\ 0, & \text{otherwise} \end{cases} \tag{2.25}$$

One common filtering criterion might involve applying a threshold. For example, to retain only points with an specific value greater than a minimum threshold $f_{\min}$, the filtering condition would be:

$$F(P_i) = \begin{cases} 1, & f_i^j > f_{\min} \\ 0, & f_i^j \leq f_{\min} \end{cases} \tag{2.26}$$

2.6.3 Statistical Outlier Filter

To remove statistical outliers, a statistical analysis on the neighbourhood of each point is performed. This includes looking at each point's neighbourhood and computing the mean distance between the point and its neighbours. By excluding outlier points whose mean distance deviates from the global mean by more than a specified number of standard deviations [48].

Let P be the set of m points

$$P = \{p_1, p_2, \ldots, p_m\}, \quad \text{where } p_i = \{x_i, y_i, z_i\}.$$

and N_i be the set of k-nearest neighbours of point i where N_i^j represent the j-th neighbour of point i. For each point P, calculate the average distance $[d_i]$ from P to its k-nearest neighbours.

$$d_i = \frac{1}{k} \sum_{j=1}^{k} ||P_i - N_i^j||, \quad \text{where } ||\cdot|| = \text{Euclidean distance.} \tag{2.27}$$

From this, calculations for obtaining the global mean $[\mu]$ and standard deviation $[\sigma]$ of the distance of neighbourhoods are performed.

$$\mu_d = \frac{1}{m} \sum_{i=1}^{m} d_i \tag{2.28}$$

$$\sigma_d = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(d_i - \mu_d)^2} \tag{2.29}$$

where m is the number points. Assume the distribution of d_i values is approximately Gaussian $\mathcal{N}(\mu_d, \sigma_d^2)$. Set an interval with a user tuned parameter α (std multiplier) to define the constrains.

$$\text{Accepted range: } [\mu_d - \alpha\sigma_d, \mu_d + \alpha\sigma_d]$$

Points not fulfilling the constraints are considers outliers and removed from the dataset i.e.,

$$d_i < \mu_d - \alpha\sigma_d \text{ or } d_i > \mu_d + \alpha\sigma_d.$$

2.6.4 Plane Fitting

Since the primary use case of the application is expected to be mostly indoor environments, where structural elements such as walls, floors, and ceilings serve as key features for understanding and estimating the surroundings. Detecting planar surfaces within the 3D point cloud becomes crucial for reliable mapping and localization. This can also serve a function in filtering, providing geometric constraints corresponding to the physical layout [49], [50], [51].

To detect plane surfaces in a 3D point cloud, the Random Sample Consensus (RANSAC) algorithm is often employed due to its robustness to noise and outliers. In plane detection, three points are randomly sampled, since a plane in 3D space can uniquely be defined by three points. Once a plane is defined by RANSAC, each points perpendicular distance to the plane is calculated and classified as an inlier or outlier depending on a threshold (how well the point fits the plane). This process is repeated for a specific number of iterations and the plane with the most inliers is selected as the best fitting plane [52, 53]. The method for obtaining a 3D plane, is as follows.

First sample three random points out of the set of points P

$$p_1 = (x_1, y_1, z_1) \quad p_2 = (x_2, y_2, z_2) \quad p_3 = (x_3, y_3, z_3)$$

Estimate the plane equation, see Appendix A.1:

$$ax + by + cz + d = 0$$

For every point $p_i = (x_i, y_i, z_i)$ in the point cloud, compute the perpendicular distance $dist_i$ from the point i to the plane.

$$dist_i = \frac{|ax_i + by_i + cz_i + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{2.30}$$

The point is considered an inlier if the following condition is satisfied

$$dist_i \leq \beta \tag{2.31}$$

where β is a user tuned parameter. Repeat the process and select the plane with the most inliers.

2.6.5 Downsampling of 3D Point Clouds

Point clouds are collections of data points often consisting of millions or even billions of points. While this level of detail is valuable, it can make processing and computations ineffective. To address this, point cloud data is typically downsampled, reducing the number of points while retaining the overall structure. There are many methods for performing downsampling on a 3D point cloud each with different pros and cons, see Figure 2.2.

Table 2.2: Down sample methods for point clouds and its pros and cons [3], [4].

Methods	***Pros***	***Cons***
Random downsampling	*- Fast and simple.* *- No extra computation required*	*- Removes points randomly, losing structure.* *- May miss crucial features.*
Voxel downsampling	*- Uniform distribution of points.* *- Adjustable density control.* *- Easy to implement*	*- Can blur fine details.* *- May lose sharp edges if voxels are too large.*
Uniform Grid downsampling	*- Preserves spatial consistency.* *- Works well for evenly distributed points*	*- May oversmooth details.* *- More computationally expensive.*
Curvature-Preserving downsampling	*- Retains edges and high-detail regions.* *- Ideal for fine-detail tasks.*	*- Computationally expensive.* *- Requires curvature estimation.*
Cluster downsampling	*Reduces noise while keeping local details.* *- Works well on non-uniform distributions.*	*Poor for high-density point clouds.* *- May create voids.*

After a theoretical analysis of the sampling methods, voxel downsampling was selected for implementation due to its balance between efficiency, structural preservation, and computational feasibility. Voxel downsampling allows for adjustable density control, making it adaptable to different processing needs. Considering the project's time constraints, its simplicity and ease of implementation further reinforced its suitability over more computationally intensive alternatives. Additionally, the important feature of upsampling is possible.

Voxel downsampling partitions 3D space into a grid of fixed-size cubes, called voxels, and retains a single representative point per voxel. By reducing the total number of points while preserving the overall structure, it efficiently simplifies the data. The representative point is typically computed as the mean position or the voxel centroid (if it contains points) [4].

Let P be the set of N points

$$P = \{p_1, p_2, \ldots, p_N\}, \quad \text{where } p_i = \{x_i, y_i, z_i\}. \tag{2.32}$$

The voxel Indices are obtain by

$$v_i = ([\frac{x_i - x_{min}}{s}], [\frac{y_i - y_{min}}{s}], [\frac{z_i - z_{min}}{s}]) \tag{2.33}$$

where s is the voxelsize (boxsize). Let the aggregated points S_v be the set of point that falls inside of the voxel v_i.

$$S_v = \{p_i \in P | v_i = v\} \tag{2.34}$$

The Voxel Point, Each set of S_v will be represented as their centroid $\hat{p}_v$.

$$\hat{p}_v = \frac{1}{|S_v|} \sum_{p \in S_v} p \tag{2.35}$$

The down sampled point set $\hat{P}$ is represented as

$$\hat{P} = \{\hat{p}_v | v\} \tag{2.36}$$

where v is unique voxel in P [3]

2.6.6 Upsampling of 3D Point Clouds

To recover a denser point cloud from the voxel-downsampled set $\hat{P}$, the original point cloud set P is filtered and points outside a voxel are removed. This method restores all original points that belong to voxel regions retained after the downsampling step, preserving more geometric detail while staying consistent with the simplified voxel structure.

Let the original point be

$$P = \{p_1, p_2, \ldots, p_N\} \subset \mathbb{R}^3$$

and the the downsampled point cloud, obtained using a voxel size $s \in \mathbb{R}^+$ be

$$\hat{P} = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_M\} \subset \mathbb{R}^3$$

Each point $p_i = (x_i, y_i, z_i) \in P$ is assigned a voxel index as follows:

$$v(p_i) = \left(\left\lfloor \frac{x_i}{s} \right\rfloor, \left\lfloor \frac{y_i}{s} \right\rfloor, \left\lfloor \frac{z_i}{s} \right\rfloor \right) \tag{2.37}$$

Let V_{valid} be the set of voxel indices occupied by the downsampled point cloud:

$$V_{\text{valid}} = \left\{ v(\hat{p}_j) \mid \hat{p}_j \in \hat{P} \right\} \tag{2.38}$$

The upsampled point cloud $\tilde{P}$ is then defined as the subset of original points that fall within the valid voxel regions:

$$\tilde{P} = \{p_i \in P \mid v(p_i) \in V_{\text{valid}}\} \tag{2.39}$$

2.6.7 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a unsupervised machine learning model, allowing for classification of unknown number of cluster in the data. In simple terms, DBSCAN works by defining rules for how one point can be considered reachable from another and its a fundamental algorithm for density-based clustering. It effectively identifies clusters of varying shapes and sizes within large datasets, even in the presence of noise and outliers [54].

Let $X \in \mathbb{R}^d$ be the d dimensional dataset and $X = \{p_1, p_2, \cdots, p_n\}$ where $p_i = \{x_i, y_i, z_i\}$. Then the ϵ-Neighbourhood is defined as:

$$N_\epsilon(p) = \{q \in X : d(p, q) \leq \epsilon\} \tag{2.40}$$

where p is a specific point (center of neighbourhood), q represent any other point in X and let $d(p, q)$ be the distance function between p and q (Euclidian distance). A point is called corepoint if:

$$|N_\epsilon(p)| \geq minPts \tag{2.41}$$

where $minPts$ is a user defined parameter deciding "dense" regions in the dataset. A point q is consider a border point if:

$$1. \quad |N_\epsilon(q)| \leq minPts \quad \text{(not a corepoint)} \tag{2.42}$$

$$2. \quad q \in N_\epsilon(p) \quad \exists \quad p \quad \text{(lies within a neighbourhood)} \tag{2.43}$$

otherwise the point q is considered as noise. How to form cluster reachability between points, needs to be defined. A point q is direct density reachable if

$$q \in N_\epsilon(p) \quad \text{and } p \text{ is a core point} \tag{2.44}$$

A point q is density reachable if there exist a sequence (chain) of points $\hat{P}$.

$$\hat{P} = \{p_0, p_1, p_2, \cdots, p_{n-1}, p_n = q\} \tag{2.45}$$

such that p_{i+1} is direct density reachable from p_i. This lets you walk from p to q step by step, moving from one dense neighbourhood to the next.

To form clusters, density reachability strings together direct density-reachability, which means that the point must be in a core point's ϵ-neighbourhood to group points into larger clusters. If you can form such a path of core points from p to q, then q belongs to the same dense region (cluster) as p [55].

2.7 SLAM

The Simultaneous Localization And Mapping (SLAM) algorithm is a method used to, as described in the name, create a 3D mapping of an environment whilst simultaneously estimating the position from the sensor. This is done by observing each individual sensor frame, resulting in a final 3D point cloud and trajectory, holding information from all timesteps.

The main goal of SLAM is to enable an agent (such as robots, vehicles or humans) to navigate and explore a unknown environment, create a map of that environment, and at the same time determine the current position within that map. This is done in real-time, without any prior knowledge of the environment.

The problem can be formulated as:

1. Estimate the agents state $[x_t]$ over time (e.g. position and orientation)
2. Build a map $[m_t]$ of the environment simultaneously.

given:

- Controls $[u_t]$ that describes the agents movement.
- Observations $[o_t]$ from sensors about the environment.

Mathematically it can be described as the joint probability of the state and map:

$$P(m_{t+1}, x_{t+1} | o_{1:t+1}, u_{1:t}) \tag{2.46}$$

The probability above represents the probability of m_{t+1} and x_{t+1} being the correct map and state respectively, given all past sensor observations $o_{1:t-1}$ (image data, LiDAR data, etc) and movement data $u_{1:t}$ (IMU or odometry). For further explanation see Appendix A.2.

2.8 AI (Scene Understanding)

To achieve effective scene understanding, a common approach is to develop a model that predicts and interprets the environment based on 3D point clouds or 2D images. There are several methods for extracting scene information from a point clouds and images, with the two most common being object detection and classification,where the goal is to identify and categorize distinct objects, and semantic segmentation, where each point in the cloud/image is assigned a label corresponding to its class.

2.8.1 3D Segmentation (PointNet)

Semantic segmentation of 3D point clouds can be broadly categorized into two main approaches based on how they operate on the data, indirect and direct segmentation. Indirect methods first convert the point cloud into intermediate representations, such as voxel grids, multi-view images, or meshes, before applying segmentation techniques. In contrast, direct methods operate on the raw, unstructured point cloud data without intermediate transformations [56].

A wide range of methods have been proposed for 3D point cloud segmentation, each with distinct advantages and limitations. These approaches differ in how they capture spatial relationships, handle varying point densities, and balance computational efficiency and scalability [57], [58]. Table 2.3 presents the pros and cons of some of the most widely used deep learning-based semantic segmentation methods for 3D point clouds.

Table 2.3: Semantic segmentation methods for 3D point clouds.

Methods	***Pros***	***Cons***
VoxNet [59]	*Simple voxel-based approach, easy to implement; works with standard 3D CNNs.*	*Loses fine detail due to voxelization; high memory usage for high resolution.*
PointNet [1]	*Processes raw point clouds directly; efficient and treats the input as a set, not a sequence.*	*Ignores local structure; struggles with complex scenes.*
PointNet++ [60]	*Captures local context using hierarchical feature learning; better generalization.*	*Increased complexity and computational cost; sensitive to point density variation.*
RandLA-Net [61]	*Lightweight and scalable to large point clouds; preserves edge features.*	*May underperform in capturing global context due to randomized sampling.*
DGCNN [62] *(Dynamic Graph CNN)*	*Dynamically constructs local graphs to capture edge features; strong segmentation accuracy.*	*Graph construction is computationally expensive; slower on large-scale data.*

After a theoretical analysis of various semantic segmentation methods, the original PointNet model was selected. The primary reason for this choice is its good performance on the S3DIS (Stanford 3D Indoor Scene Dataset) [63], which closely resembles the use case, apart from the presence of smoke in the environment. PointNet offers a balanced trade-off between model complexity and performance, making it suitable under the project time constraints.

Its relatively low complexity allows for deeper understanding and easier customization, which is crucial for adapting the model to our specific needs. Moreover, PointNet is well documented, open source, and has been widely adopted in the research community. It has also served as the foundation for many of the more recent, advanced point cloud models.

PoinNet was one of the first deep learning methods for handling directly 3D point cloud and receives unordered point sets as input, meaning the order of the points does not matter. PointNet has two major parts, Classification Network and Segmentation Network, allowing the model to both perform object classification and semantic segmentation if needed. See Figure 2.2 for an illustration of the architecture.

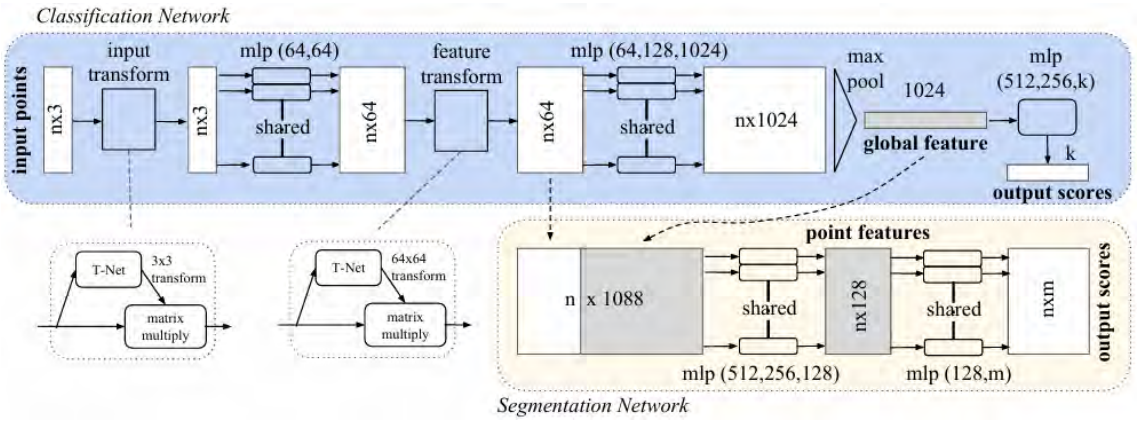
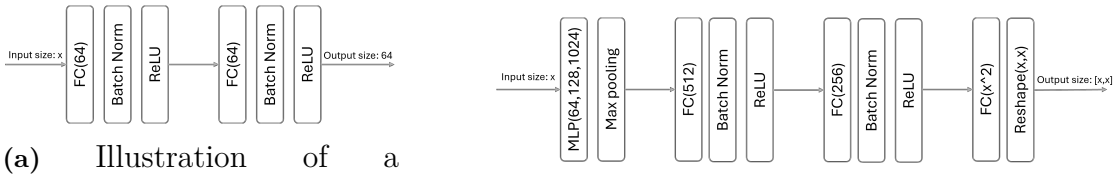


Figure 2.2: PointNet Architecture (from original paper [1]). The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net [1].

The model takes as input n points in 3D space, where each point is represented by its coordinates (x, y, z). For segmentation tasks, the model outputs an $n \times m$ matrix, where each row is a one-hot encoded vector corresponding to the predicted class label of each point. The classification network contain transformation networks (T-Nets), multi layer perceptrons (MLP), and a maxpooling layer (maxpool).

The MLPs are small neural networks composed of fully connected layers with shared weights, implemented as 1D convolutional layers with a kernel size of one, followed by a batch normalization and ReLU activation function. For example, the first MLP(64,64) indicates that each point with a input of size three (corresponding to the (x,y,z) coordinates) is passed through a hidden layer with 64 units, followed by an output layer with 64 units [1], see Figure 2.3a.



(a) Illustration of a mlp(64,64)

(b) Illustration over the T-Net

Figure 2.3: An illustration of parts in the classification network of PointNet.

The T-Nets are smaller networks with the goal of predicting an affine transformation matrix and applying it to the input. The intuition behind transforming the input data is as follows, what if the original coordinate system is not the most suitable for representing the data? To address this, PointNet introduces the T-Net, a small neural network that learns a transformation matrix to map the input points into a more optimal coordinate space [64]. The T-Net architecture consists of an initial shared MLP with layer sizes (64,128,1024), applied independently to each input point. This is followed by a max-pooling layer to aggregate a global feature vector. The global feature is then passed through two fully connected layers of sizes 512 and 256, each followed by batch normalization and a ReLU activation function. Finally, a fully connected layer outputs either 9 or 4096 values, depending on the input feature dimension,which are reshaped into a transformation matrix of size 3×3 or 64×64 [1], see Figure 2.3b.

The segmentation network takes both local and global features extracted from the classification network, meaning the input is $n \times 1088$, see Figure 2.2, and processes them using a shared MLP with layer sizes (512,256,128). This is followed by another shared MLP with layers (128,m), where m is the number of segmentation classes. The output is a per-point classification score across the m classes [1].

2.8.2 YOLO (Object Detection)

Real-time object detection is not a new thing nor is it a big challenge to implement. The most well-known and commonly used industrial application is the YOLO series. There exist many version of the YOLO model, The first one entered the world in 2016 and has continued to be enhanced, the latest researched version is currently YOLO version 9 authorized by the original research group [65]. The structure of a YOLO model can be divided into 3 part, see Figure 2.4, each fulfilling a separate purpose, and can be implemented in different ways.

- Backbone: Responsible for extracting meaningful features from the input image. It transforms raw pixel data into abstract, informative feature representations important for detecting objects. The backbone generates feature maps at multiple resolutions, capturing both local details (such as edges, textures, and small objects) and global context (larger shapes, object arrangements, and overall image semantics). This component is typically implemented using a Convolutional Neural Network (CNN), such as Darknet-53, CSPDarknet, MobileNet, EfficientNet, or ResNet.

- Neck: Responsible for aggregating and refining feature maps produced by the backbone. Its primary goal is to effectively combine low-resolution (global) features with high-resolution (local) features, enabling accurate detection of objects at multiple scales. Commonly implemented as a Feature Pyramid Network (FPN) and/or Path Aggregation Network (PANet).
- Detection Head: Responsible for predicting bounding boxes, object confidence scores (probability of object presence), and class probabilities for detected objects. Two primary types of detection heads exist, Anchor-based: Predicts bounding boxes by adjusting pre-defined anchor boxes through offsets (relative adjustments). Anchor free: Predicts bounding boxes directly without relying on anchor boxes. Both anchor based and anchor free detection heads are generally implemented using convolutional layers.

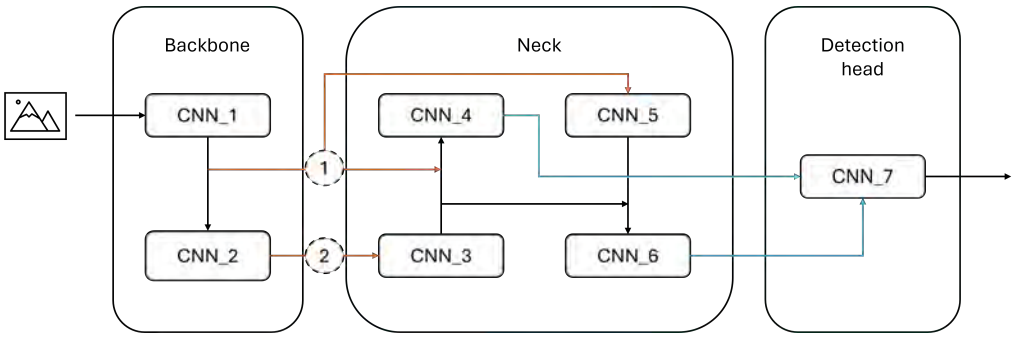


Figure 2.4: Simple concept illustration of the YOLO architecture, 1. global feature map, 2. local featuremap.

The selected object detection model is YOLO version 8 (YOLOv8), a stable and well-documented architecture officially recognized by the original YOLO authors. It is widely adopted in both academic and industrial contexts due to its balance of performance, modularity, and ease of deployment. YOLOv8 is available in five variants, each offering a trade-off between model complexity and accuracy. These versions range from 3.2 million to 68.2 million parameters, providing flexibility depending on computational constraints [66].

The choice of model version depends on the balance between computational efficiency and detection accuracy. For this project, real-time performance is important, along with maintaining a high level of accuracy. Therefore, the YOLOv8n (nano) variant has been selected, which is the lightweight version. It requires only 8.7 billion Floating Point Operations (FLOPs) per inference, making it well-suited for deployment in real-time systems [66]. The architecture of YOLOv8 is illustrated in Figure 2.5.

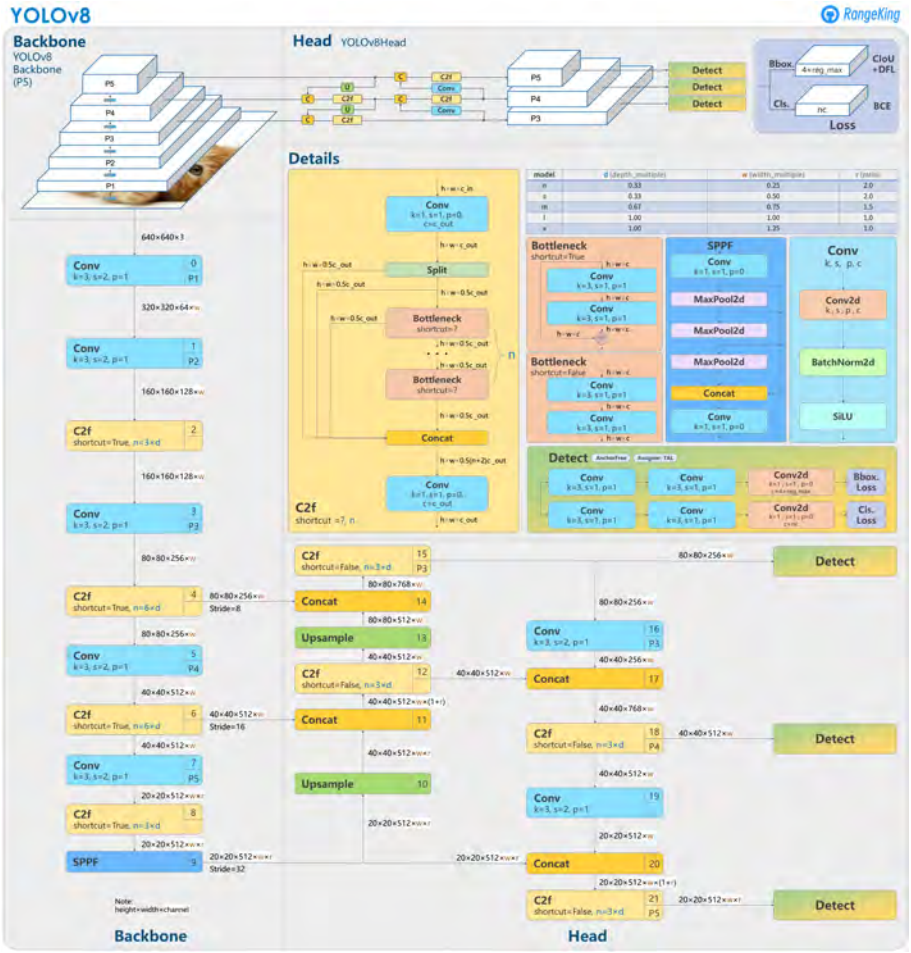


Figure 2.5: Architecture of YOLO version 8 [2], visualisation made by GitHub user RangeKing.

The architecture employs EfficientNet-B4 as its backbone, consisting of 28 layers, which are responsible for extracting features from the input image. From this backbone, three distinct feature maps at different resolutions are produced. These feature maps are then passed forwarded to a combined Neck and Detection Head, implemented using a Neural Architecture Search–based Feature Pyramid Network (NAS-FPN). The NAS-FPN module have 36 layers [67].

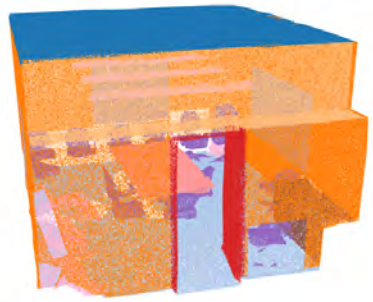
It is important to note that object detection happens at multiple spatial resolutions, enabling the model to capture both local details (e.g., small objects) and global context (e.g., larger structures). After the initial predictions are generated, a confidence threshold is applied to discard detections with low confidence scores. Non-Maximum Suppression (NMS) is performed to eliminate bounding boxes that significantly overlap the same object. This processing is conducted across the different levels of detection and the remaining high-confidence, non-overlapping predictions are then aggregated into a single unified output [67].

2.9 Dataset (Open Source)

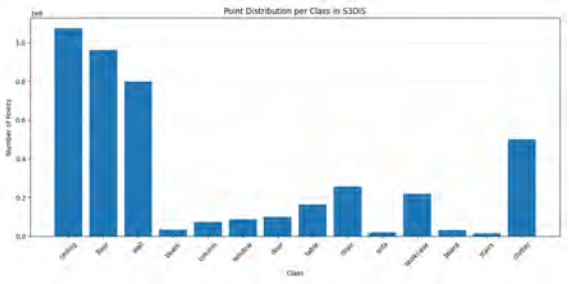
Based on the projects characteristics and delimitation, no existing datasets are available that directly matched the specific use case. Therefore data collection was necessary to support the development of the prototype and due to time constraints and the demanding requirements of developing AI models, it is important to identify and utilize open-source datasets that can simulate or closely resemble the use-case scenarios. This will allow for training AI models on larger dataset and later customize the models to the specific use case, with either logic or fine tuning.

2.9.1 S3DIS (Stanford 3D Indoor Scene Dataset)

The Stanford Large-Scale 3D Indoor Scene (S3DIS) dataset was released back in 2016 and is a well know collection designed to enhance research in 3D indoor scene understanding. It contains six large indoor areas, holding 272 rooms, recorded from three different buildings, see Figure 2.6a. Each point within these 3D point clouds is represented by both the coordinate (x, y, z), their corresponding rgb colour (r, b, g) and holds an annotation from one of the 14 semantic classes [63], see Figure 2.6b. The S3DIS dataset has become a benchmark in the field of 3D computer vision and is widely used for training and testing models aimed at indoor related applications.



(a) Visualization over one room in the dataset. (Coloured class-wise)



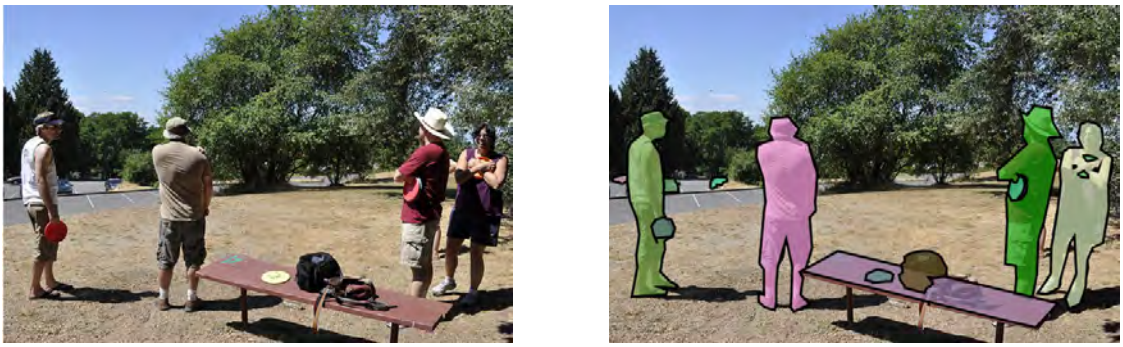
(b) Bar plot over the semantic classes.

Figure 2.6: Illustration of the S3DIS dataset.

2.9.2 COCO (Common Objects in Context)

The Common Objects in Context (COCO) dataset was introduced in 2014 and has become one of the most widely used benchmarks for object detection, segmentation, and image captioning in the field of computer vision. It contains over 200,000 labelled images, featuring more than 1.5 million object instances across 80 semantic categories, with annotations including bounding boxes, object class labels, instance segmentation masks, and keypoints for human pose estimation, see Figure 2.7a. The dataset captures objects in complex and cluttered real-world environments,

providing diverse and challenging scenarios [68]. COCO has played a crucial role in advancing research in object-level scene understanding and continues to be a standard dataset for training and evaluating modern computer vision models [69].



(a) Original image **(b)** Segmented image

Figure 2.7: Illustration of an image in COCO dataset

2.9.3 Flir Dataset (Teledyne FLIR Thermal Dataset)

The Flir dataset is owned and distributed by the Teledyne FLIR company. It contains correlated RBG images and Thermal images, and was introduce with the goal to encourage research on sensor fusion algorithms. The dataset includes 26,442 fully annotated frames with 520,000 bounding box annotations holding 15 different object categories. The frames are recorded in traffic scenarios. In the Figure 2.8, an illustration of an image from the dataset is illustrated.

(a) Original image **(b)** Annotated image

Figure 2.8: Illustration of an image in the Flir dataset.

2.10 Hardware (HW)

This section presents the hardware components used in the project, detailing their functionalities and key specifications. The hardware is categorized into different modules that contribute to the overall performance and objectives of the project, each with a specific function.

2.10.1 LiDAR (Seyond Robin W1G LiDAR)

The Seyond Robin W1G LiDAR is a semi-solid-state LiDAR which reaches twice the range of similar LiDARs on the market whilst still managing to detect points within a 0.1m distance. It utilizes ToF methodology to calculate distances and is made to be used for a variety of sectors, such as automotive, robotics and much more [5]. Table 2.4 shows a list of the LiDARs key specifications.

Table 2.4: Specification of SEYOND [5].

LiDAR specifications	
Laser wavelength	*905nm*
FOV	120° × 70° *(width x height)*
Maximum detection range	*150m*
Minimum detection range	*0,1m*
Frame rate	*10-20 FPS*

2.10.2 Thermal Camera 1 (TELEDYNE FLIR T540)

The FLIR T540 is a thermal imaging camera designed for industrial and electrical applications. It offers precise temperature measurements up to 1500°C with a thermal sensitivity below 40mK. The device supports radiometric video and automatic focus for efficient diagnostics [6]. Table 2.5 shows a list of the camera's key specifications.

Table 2.5: Key feature of TELEDYNE FLIR T540 [6].

Key features	
Framerate	*30Hz*
Sensitivity (NEdT)	*<40mK*
Focal length	*17mm*
Field of View (FoV)	24°/14°

2.10.3 Thermal Camera 2 (TELEDYNE FLIR Tau 2)

The FLIR Tau 2 was selected as the final thermal sensor for its compact design and because it was already the available thermal camera in the project. This long-wave infrared (LWIR) camera is suited for embedded systems and field deployments, used for example in drones [7]. Key specifications of the FLIR Tau 2 are summarized in Table 2.6.

Table 2.6: Key feature of TELEDYNE FLIR Tau 2 [7].

Key features	
Framerate	*30Hz*
Bpectral band	*7,5-13,5 μm*
Sensitivity (NEdT)	*<40mK*
Focal length	*13mm*

2.10.4 RGB Camera (GoPro)

The GoPro HERO10 Black was selected as the RGB sensor due to its high image quality, portability, and it already being available in the project. It has been used in research and industrial applications, including autonomous systems and environmental data collection [8]. Key specifications of the GoPro HERO10 Black are summarized in Table 2.7.

Table 2.7: Key feature of GoPro [8].

Key features	
Model	*HERO 10 Black*
Video resolution	*4K 120*
Photos resolution	*23 MP*

2.10.5 Mini Computer (LattePanda)

The LattePanda Alpha 864s was selected as the hardware for the mini-computer. This high-performance single-board computer is designed for low power consumption while being capable of running a full Windows or Linux operating system [9]. Key features of the LattePanda Alpha 864s are summarized in Table 2.8. For more detailed specifications, refer to the official Product Specification.

Table 2.8: Key feature of LattePanda Alpha 864s [9].

Key features
CPU: Intel® Core™ i5 8210Y / 8200Y
Core: 1.6-3.6GHz / 1.3-3.9GHz Dual-Core, Four-Thread
RAM: 8GB
Storage: 64GB
Co-processor: Arduino Leonardo
*Dimension: 115mm * 78mm * 14 mm*

3

Methods

The aim of the project is to investigate how the sensors LiDAR and thermal camera can be enhanced and combined to better estimate the surroundings and identifying the environment in cases where humans vision is impaired. The methodology for achieving and investigating this aspect involved initially designing a prototype system that integrates all necessary components to improve the estimation and identification of the surroundings. Due to obstacles and limitation encountered throughout the project, the methodology shifted to investigation of the individual parts in the system with no connection made between each other. However, the mindset of the system based approach has been considered in the development to allow an easy and simple implementation of the modules in the system.

To understand the reasoning behind the investigated components and the choices made, it's important to get an overview of the system and the methodology. The system will hold two sensors, collecting 3D point clouds from the LiDAR and 2D images from the thermal camera, the output will be a visualization of the surroundings.

The 3D data is used throughout the system as the main estimation data, but is noisy and contains false positive points. To enable its use for improving performance in other parts, it needs to be as accurate as possible and properly filtered. Since the LiDAR point cloud degrades in smoke filled environment, the 2D data from the thermal sensor are fused and mapped to 3D, resulting in a, hopefully, better 3D point cloud. This point cloud is meant to be used in the SLAM algorithm, creating a combined 3D scene over the collectable data throughout the timesteps. The SLAM was initially thought to be implemented and a licence was given, however the licence only lasted for four weeks at the beginning in the project. Therefore development has been made to implement the SLAM but wasn't used or isn't presented, only discussed. This resulted in exploration of the possibilities of estimating the odometry based on the sensors (a part in SLAM). An important part is also to investigate how a system can identify and estimate its understanding of the surroundings. This has been explored through methods like object detection and segmentation.

In addition to the development of the system, data collection has been necessary to enable development tailored to the specific scenario and to allow for verification. This resulted in a prototype capable of recording data from the final sensors and ready to incorporate the implementations of the final system. Analysis of the recorded data, methods, results, and parameters has been made to ensure good performance of the modules and later in the final system.

3.1 Data Collection/Gathering

This section outlines the methodology used for data collection throughout the project. Data was collected continuously throughout the project, since the only available sensors at the project start were a LiDAR sensor, Section 2.10.1, and RGB camera, Section 2.10.4. A total of four data collection sessions were conducted, each with a specific focus and involving an additional new sensor. The test scenarios were designed to simulate real scenarios relevant to the intended application. To simulate reduced visibility conditions caused by real fire smoke, a theatre smoke machine was used. This type of smoke is used in rescue training by fire departments, since it creates realistic low visibility environments. Various smoke levels were generated and recorded, allowing thorough analysis of the modules and systems performance, while identifying under which circumstances the modules and overall system may fail. An overview of the data collection sessions is provided in Table 3.1.

Table 3.1: Summaries of the tests performed for data collection.

Scenario	***Nr smoke lvls***	***Durtion***	***Sensors***
Test 1 (2025-02-01) (Section 3.1.1)			
Static	8 smoke levels	*1 sec (10 frames)*	*LiDAR, rgb*
Dynamic	8 smoke levels	*10 sec (100 frames)*	*LiDAR, rgb*
Test 2 (2025-03-13) (Section 3.1.2)			
House	3 smoke levels	*50 sec (500 frames)*	*LiDAR, IR, rgb*
Kitchen	3 smoke levels	*30 sec (300 frames)*	*LiDAR, IR, rgb*
Operation	2 smoke levels	*70 sec (700 frames)*	*LiDAR, IR, rgb*
Test 3 (2025-04-29) (Section 3.1.3)			
Inside	no smoke	*152 images*	*IR*
Inside	no smoke	*videos*	*IR*
Test 4 (2025-05-07) (Section 3.1.4)			
Dynamic inside	4 smoke levels	*55 sec (550 frames)*	*LiDAR, IR, rgb*

3.1.1 First Test (2025-02-01)

The first test was conducted specifically to collect LiDAR data for early-stage development. The primary objective was to record data that could be used to develop different filtering and data enhancement methods, but also to allow for early analysis. The setup utilized for the test included the Seyend LiDAR, see Section 2.10.1, in conjunction with an RGB camera (GoPro), see Section 2.10.4, to enable visual comparison, Figure 3.1a. This combination allowed for a more comprehensive analysis by correlating LiDAR data with visual comparison.

(a) Image over the sensors for first test

(b) Illustration over the scene for first test

Figure 3.1: Illustration of the setup for first test.

The environment for which the tests were preformed was a canteen with a lot of different geometries placed on and around the tables, e.g human like object, plants, etc, to test the LiDAR's ability to identify objects of varying sizes, Figure 3.1b. This was done to enable more in depth analysis of how the LiDAR performance decreases with impaired vision caused by smoke. 8 smoke levels were created and recorded, ranging from no smoke (reference point) to a fully smoke filled room (density 7), Figure 3.2. Two distinct scenarios were recorded to capture data under varying smoke conditions. In the static scenario, the sensors were mounted on a fixed table and recorded for 1 second at a frame rate of 10 Hz, resulting in a total of 10 frames. In contrast, the dynamic scenario involved moving the sensors along a predefined path to collect data suitable for the development of temporal filters and SLAM. For the dynamic case, recordings were conducted over 10 seconds at the same frame rate of 10 Hz, resulting in 100 frames in total.

(a) Smoke density 1

(b) Smoke density 4

(c) Smoke density 7

Figure 3.2: Visualization of the different smoke levels in test 1.

3.1.2 Second Test (2025-03-13)

The second test was conducted in collaboration with the emergency services during one of their field exercises simulating a real rescue operation. The test took place inside a house structure, Figure 3.3b, which was filled with cold smoke (theater smoke) to simulate reduced visibility conditions. The aim was to record three distinct smoke levels, ranging from no smoke (reference point) to a fully smoke-filled environment (density level two). This would allow for obtaining realistic and matching data sets for all sensors, and be used in the developing process. Due to limitation encountered under the session, the smoke levels recorded was:

- No smoke (density level 0)
- Varied smoke level (density level 1): the smoke level varied through the different rooms. Simulating a real scenario where the smoke level dynamically increases as the recording progresses deeper into the house, and with one single smoke source.
- Density level 2: Medium/light homogenous smoke level across all the rooms.

Multiple scenarios were recorded, both to simulate operational conditions and to provide diverse datasets for development purposes. In this test, the first version of the prototype ("lollipop") was utilized, holding the LiDAR, Section 2.10.1, and RGB camera, Section 2.10.4, along with an external IR camera, Section 2.10.2. In Figure 3.3a, the sensor used together with the scene where the test was performed is illustrated.

(a) Image of the sensors for second test

(b) Illustration of the scen for second test

Figure 3.3: Illustration of the setup for first test.

The recorded scenarios include an inside walkthrough, a combined outside-to-inside transition, and a kitchen environment with interior features. Each scenario is designed to serve a specific purpose during the development phase, providing diverse data conditions to support system testing, analysis, and optimization.

- "**Inside**": This scenario involves following a predefined path inside the house on the ground floor. In this scenario, all three smoke levels were recorded.

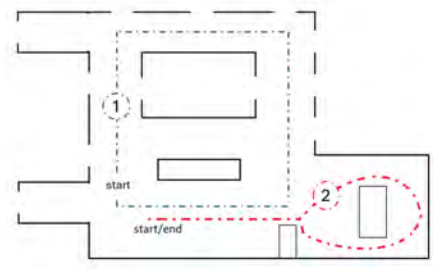
This setup is designed to support the development and evaluation of the entire system, including the SLAM capabilities.

- "**Outside + Inside**": This scenario follows the same predefined path as the "Inside" scenario but begins outside the building, capturing the transition from an outdoor, smoke-free environment to a fully smoke-filled interior. This scenario was recorded for the smoke levels: no smoke and density 1. The purpose is to record the gradual change in conditions, providing data that simulates a firefighter's point of view as they move from clear visibility into dense smoke.
- "**Kitchen**": This scenario consists of a walkthrough of the kitchen area, including interior features such as windows, furniture, and a kitchen island. All smoke levels were recorded for this scenario. It is primarily intended for the development and analysis of system performance in a more realistic indoor environment. Additionally, this scenario serves as a verification case to evaluate how well key features can be detected and captured under varying smoke densities.

3.1.3 Third Test (2025-04-29)

The third test was performed with the aim of creating a dataset for training object detection and collection of development and test data for IR odometry. The sensor used was the Thermal camera 2, Section 2.10.3, and the test was performed with no smoke, and in a office environment with a lot of geometrical obstacles. The aim was to create data simulating real scenarios with minimal noise. For the dataset collection, up to three people were included in the images with different poses, standing, crouching, etc. For the collection of data to the odometry a closed path around in the office was recorded. In Figure 3.5, illustration of the sensor used and path taken for odometry estimation is illustrated.

(a) Image over the IR sensor for the third test



(b) Path for the odmetry in the office environment (2), path for fourth test (1)

Figure 3.4: Illustration of the setup for third test.

3.1.4 Fourth Test (2025-05-07)

The fourth test was performed to generate a final, high quality dataset containing synchronized data from the final sensors. The aim was to capture realistic and reliable sensor data under controlled and representative conditions, which is going to be used in the evaluation part of the project. This test was carried out in the same indoor office environment as test three, Section 3.1.3, due to easy access and availability. To simulate the usecase of emergency scenarios, cold smoke (theater smoke) was used to create varying levels of smoke, mimicking the conditions in a fire rescue operations.

Four distinct smoke levels were introduced: starting from clear visibility (no smoke) to heavy smoke, with two intermediate density levels in between. This allows for good systematic evaluation of both the sensors performance and for the developed softwares (models, algorithms) in increasing smoke levels. The difference between no smoke and heavy smoke is illustrated in Figure 3.5b. The recordings followed the predefined path illustrated with number 1 in Figure 3.5, with aligned start and end positions. The final sensor configuration used during the test is seen in Figure 3.5a. Where the LiDAR, Section 2.10.1, Thermal Camera 1, Section 2.10.3, and the RGB camera , Section 2.10.4, was used.

(a) Image over the sensors for the fourth test

(b) The heavy smokelevel (on top) and the reference, no smoke (at the bottom).

Figure 3.5: Illustration of the setup for third test.

3.2 Data Enhancement

This section provides an explanation of the methodology used in the design of the filtering technique. It also presents an analysis of the approach, including the reasoning behind the design choices.

3.2.1 Analysis of LiDAR Sensor

Since the LiDAR sensor captures a range of measurements, including intensity, reflectivity, distance to the observed object, and the angle of the emitted signal, analysing these signals provides valuable insight into how environmental factors, such as smoke, affect sensor performance. This analysis is particularly useful for identifying which signal properties can be leveraged for filtering. Among these measurements, intensity is the most critical to examine. Intuitively, smoke particles are expected to have different reflectivity characteristics compared to solid objects. Reflectivity is approximated using the raw intensity values, which typically range from 0 to 255.

To assess how varying levels of smoke affect LiDAR intensity measurements, it is sufficient to analyse the distribution of intensity values under different smoke conditions. The analysis is visualized using bar plots, where intensity values are grouped into bins of width 10 (e.g., 0–10, 10–20, ..., 240–250). In addition to the visual distribution, key statistical properties such as the mean, median, and standard deviation will be computed for each smoke level to quantify the impact on signal strength.

3.2.2 Voxel Downsampling Analysis

To efficiently downsample the point cloud from high to lower resolution, voxelization is applied. This is done to reduces the computational load, needed for real-time, but also to lower the bias introduced by the LiDAR sensor. Since LiDAR emits light waves in uniform angular intervals, objects that are closer to the sensor are captured with higher point density, while those farther away appear with lower resolution. This phenomenon can be observed in Figure 4.2a in Section 4.1.1. To leverage the trade off between computational efficiency and data accuracy/resolution in our use case, a voxel size analysis is conducted to identify a suitable balance. For indoor environments, recommended voxel sizes typically range between 0.01 m (1 cm) and 0.05 m (5 cm) [4]. Therefore voxel sizes close to these values are investigated and Table 3.2 presents the exact sizes explored.

The analysis is carried out under conditions without smoke to isolate geometric objects and its characteristics change of lower resolution, allowing for clearer evaluation. Visual inspections of objects are used to assess the impact of different voxel sizes and finally select a suitable voxelsize.

Table 3.2: Experimental parameters for analysis of voxelsize.

Tested Voxelsizes				
Voxel size	*0.1 (10cm)*	*0.05 (5cm)*	*0.03 (3cm)*	*0.01 (1cm)*

3.2.3 LiDAR Filtering

To be able to use the LiDARs 3D point cloud in the system, removal of false positives (noise and other false points) is crucial to enhance the system performances. To enable an effective filtering method for the LiDAR generated point cloud, an analysis of the collected data was performed. This analysis identified a set of requirements that the filtering method must meet to ensure good performance of the system.

The filtering method is required to operate only on the positional data, i.e. the coordinates (x,y,z) captured by the LiDAR sensor. All other sensor measurements, such as reflection intensity, are excluded due to their unreliability in the presence of smoke. As shown from the result of the analysis of the reflectivity, Section 4.2.1. Reflection measurements by the sensor become significantly disrupted in smoke filled environments, thus making them unsuitable in this usecase.

From the collected dataset, it was observed that LiDAR behaviour varies when encountering windows. The sensor can either produce false readings due to reflections, see Figure 4.2a, or pass through the glass and capture measurements from objects on the opposite side, see Figure C.2 in Appendix C. Both cases result in undesired data, either false positives or measurements that negatively impact the system. Therefore, the filtering method must be designed to isolate only the relevant indoor environment surrounding the sensor,e.g. the interior of the room.

Smoke introduces two major challenges for the LiDAR measurements. First, smoke particles are detected as dense clusters of points that appear in front of the sensor, as illustrated in Figure 4.5c. Second, as smoke density increases, the number of false positive points beyond the smoke cluster decreases. This underscores the need for a filtering method that is adaptive to varying smoke levels. The filter must detect and remove smoke clusters to preserve meaningful structural information from the environment.

The requirement list for the LiDAR filtering can be shown down below:

- Only use coordinated measurements from the LiDAR sensor (x, y, z).
- Capture only the indoor environment, e.g the room.
- Removing False Positives, e.g. smoke clusters, reflections, etc.

The overall filtering method can be shown in Figure 3.6, which shows all the different parts and the information flow. After the main filtering functionality, an upsampling option is available, allowing the resolution to be set independently from the voxel downsampling.

Figure 3.6: Flowchart over LiDAR point cloud filtering.

The filtering method is designed to process each individual frame of the incoming 3D point cloud stream. This algorithm, see Algorithm 1, dynamically adapts its filtering strategy based on the density of points in each frame, allowing it to handle varying levels of visibility caused by smoke or other obstructions. It incorporates both geometric reasoning and statistical noise removal to retain meaningful structures while discarding false positive data. A summary of all parameters used throughout the filtering method is provided in Table 3.3.

The distance-based removal is performed as the very first step, even before voxel downsampling. This is because the cluster directly in front of the sensor is considered to be false positives caused by smoke particles. These points can negatively affect the voxel downsampling process, which uses the centroid of each voxel to represent the point cloud. The threshold for this filtering was chosen based on both the LiDAR's minimum effective range of 0.1 meters, Section 2.10.1, and visual inspection of multiple frames. The threshold was gradually increased until the entire false-positive cluster was consistently removed across all frames. The final threshold $[\tau_d]$ was set to 0.5 meters, ensuring reliable removal while details under one meters in front of the sensors remains.

Voxel downsampling is applied to improve computational efficiency, with the goal of enabling real-time processing. In addition to reducing the number of points, this step helps normalize the point density, especially in regions close to the LiDAR where the density is naturally higher. This is important for later parts in the filtering steps that rely on local distribution patterns and neighbourhood based operations, as uneven point densities can otherwise lead to biased or inconsistent results. After, the analysis was conducted to evaluate the impact of voxel downsampling on both the resolution of the point cloud and the computational performance, Section 3.2.2. The trade-off between preserving the resolution for details and achieving a manageable computational load was considered and a voxel size $[\nu]$ of 0.05 meters (5 cm) was selected. It provided a good compromise between maintaining details and improving processing speed, see Section 4.2.2.

Considering the effect of smoke, there is an increase in false positives detected in front of the sensor, while the remaining portion of the returning signal has a higher probability of being true positives. According to the findings when analysing the frames from the different test, most of the existing points are more likely to be true positives, Figure 4.6. The smoke level is estimated by analysing the number of points in the point cloud. Several factors must be considered when selecting an appropriate threshold for determining the smoke level:

- Smaller rooms naturally contain fewer points in the point cloud after voxel downsampling, due to denser voxel boxes.
- The chosen voxel size directly influences the density and total number of points in the downsampled point cloud.
- Higher smoke levels lead to the removal of a larger cluster of points during distance removal, which further reduces the number of remaining points.

Based on the voxel downsampling analysis, a typical room without smoke contains approximately 14,000 points after downsampling, see Table 4.1. The threshold $[\tau_s]$ for classifying smoke level is set at 9,000 points. This corresponds to a loss of over 35% of the point cloud due to smoke interference. If the number of remaining points fall below this threshold, the smoke level is classified as high, otherwise it is considered low. The reason for classifying the smoke level is due to the resulting discontinuities in the point cloud. High levels of smoke create isolated clusters of true positives, Figures 4.14b and C.2b, which complicates the task for the DBSCAN algorithm. These fragmented clusters make it more difficult for DBSCAN to accurately determine which clusters represent meaningful structures and should be kept.

For high smoke levels, the only necessary step is to remove statistical outliers, Section 2.6.3. which makes the data more clean of noise based on the point clouds statistical distribution. After exploration the selected parameter k (nearest neighbour to look at) was set to 20 and the α (standard deviation multiplier) affecting the threshold, was set to 2.5.

For low smoke levels, the algorithm adopts a more sophisticated approach to eliminate false positives such as reflections, smoke-induced clusters, and other noise. It begins by identifying the six largest planes ($p = 6$) in the frame. These planes are detected by selecting points that lie within a distance of 0.8$*$voxel size from each plane. The logic behind extracting the six dominant planes is to use them for cluster removal after DBSCAN clustering. DBSCAN is applied with the following parameters: Epsilon $[\epsilon]$ of 0.5 and minimum number of 20 points $[n_{points}]$ required to form a cluster. After identifying x clusters, the largest cluster is kept along with any clusters that lie on at least three of the 6 largest planes, with orthogonal condition. This approach allows for saving multiple clusters under the assumption that elements like floors or ceilings, typically shared across different parts of a room, should lie on the same geometric planes. Clusters aligned with these dominant planes together with orthogonal planes are likely to represent true positive segments of the room structure. Lastly, statistical outliers are removed with the same parameters as for the

high smoke level case.

The upsampling option enables the ability to keep a high resolution final 3D point cloud while maintaining computational efficiency during processing. Making this feature optional allows the system to remain adaptable for potential real-time applications in the future.

Algorithm 1 Filtering Method

```
1: [redacted]
2: [redacted]
3: [redacted]
4: [redacted]
5: if [redacted] then
6:     [redacted]
7:     [redacted]
8:     [redacted]
9:     [redacted]
10:    [redacted]
11:    for [redacted] do
12:        if [redacted] then
13:            [redacted]
14:            if [redacted] then
15:                [redacted]
16:            end if
17:        end if
18:    end for
19:    [redacted]
20: else
21:    [redacted]
22: end if
23: return [redacted]
```

Table 3.3: Summary over the parameters selected for LiDAR filtering.

Parameter	***Value***
Distance threshold $[\tau_d]$	
Voxel size $[\nu]$	
Smoke lvl threshold $[\tau_s]$	
Num planes $[n_{planes}]$	
Dist to plane $[d_p]$	
Epsilon $[\epsilon]$	
Min points $[n_{points}]$	
Std multiplier $[\alpha]$	
Num nearest neighbour $[k]$	

3.3 Sensor Fusion

The sensor fusion model integrates data from a LiDAR sensor and a single infrared (IR) camera. The purpose of this model is to enhance the visualization of key objects in 3D space by incorporating edges detected by the thermal sensor. The pipeline proceeds as follows: the IR data is first preprocessed, followed by edge detection to extract relevant features. Then, the LiDAR point cloud is projected onto the 2D image plane, and depth values are assigned to the IR edge points via depth mapping using the corresponding LiDAR projections. Finally, these 2D IR points are backprojected into 3D space, resulting in a improved 3D point cloud. In Table B.2 in Appendix B.2, the model specific parameters and the relative sensor positions are illustrated.

- The first stage of the process involves preprocessing the IR data, as outlined in Section 2.4.1. After preprocessing, the Canny edge detection algorithm is applied to the enhanced thermal image to identify meaningful edges. The IR camera used has relatively low resolution compared to conventional RGB cameras. Moreover, the data was collected in degraded environments, which further reduced image quality. As a result, conservative edge detection parameters were chosen to prioritize noise robustness, ensuring the reliability of detected features.

- In the second stage, LiDAR points are projected onto the 2D IR image plane using standard projection equations as described in Section 2.4.2. The camera intrinsic matrix used for this step was:

$$K_{IR} = \begin{bmatrix} 764.7 & 0 & 320 \\ 0 & 764.7 & 256 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Due to lack of the necessary materials, no calibration of the IR camera was performed. It was instead obtained from the cameras specifications which can differ slightly from the true camera intrinsic matrix.

After the points have been projected, they are used to generate a sparse depth map. The depth mapping process, described in Section 2.4.6, is essential for this model, particularly because the setup includes only a single IR camera with no helping sensors, thus being unable to single handedly estimate depth. Therefore, the IR data must be geometrically aligned with the LiDAR data. This setup demonstrates the capabilities of the IR camera within a sensor fusion model, though it does introduce a dependency on the quality of the LiDAR data. Sparse or noisy LiDAR data can lead to inaccuracies in depth assignment.

- The final stage is to backproject the 2D IR edge points, which now have assigned depth values, into 3D space following the approach described in Section 2.4.3.

For clarity in visualization, LiDAR points and IR-derived points are visualized with separate colors. This makes it easier to differentiate between the two data sources and enables users to quickly identify structural features such as doors, windows, and walls.

Since the IR camera captures data at 30 frames per second (fps) and the LiDAR operates at 10 fps, the two sensors were synchronized by selecting every third IR frame. The difference between three consecutive frames was minimal, thus this was considered acceptable. However, following this method removes the possibility of using any temporal logic between these frames.

Finally, the model was designed with computational efficiency in mind. All processing steps were optimized to balance performance and speed. Two versions of the model were implemented: one optimized for real-time performance with slightly reduced accuracy, and another that prioritizes accuracy at the cost of increased processing time. The main difference between these two versions is that the densification of the depth map was downscaled by a factor of 0.5 to reduce computational time for one of the models. Since the hardware uses a LiDAR sensor operating at 10 Hz, the processing time for each frame must be below:

$$t = \frac{1}{f} = \frac{1}{10} = 0.1\text{s} \tag{3.2}$$

3.4 Monocular Visual Odometry

The monocular visual odometry (VO) model has been developed to evaluate the capabilities of a single-camera system, with the goal of giving accurate positional data in a real-time system. While the current implementation is not real-time based, the model has been designed with computational efficiency in mind. It is also generic, supporting both infrared (IR) and RGB camera inputs to enable performance comparisons under different sensing conditions. The methodology employed in the model is structured as follows: If IR data is received, it is first preprocessed. Feature detection and tracking are performed across consecutive frames. Triangulation is used to estimate height information, which is employed to scale the pose to a realistic metric. The final pose is visualized from a bird's-eye perspective, allowing the trajectory to be viewed across the full path. Table B.3 in Appendix B.3 shows the selected parameters used in the model.

The model is designed with a dual focus: first, to evaluate the VO system with detectors for the most accurate possible results; and second, to test its performance for minimal computational load to explore the potential for real-time compatibility. Three versions of the model were implemented. A SIFT-based version was created to prioritize accuracy, with less concern for computational time. ORB and KLT-based versions were developed to maximize computational efficiency and real-time potential. The methodology used in the models is presented below:

- The data preprocessing follows the same logic as for the sensor fusion case in Section 3.3

- Then SIFT, ORB and KLT tracking are used to detect and track features across consecutive frames. Because the IR camera has a lower resolution and higher noise levels compared to the RGB camera, the input parameters for the KLT algorithm must be adapted accordingly. For IR data, more conservative parameter settings are used, such as a lower minimum feature distance and smaller tracking windows to ensure robustness in low-texture regions. In contrast, the RGB camera allows for more relaxed parameter selection due to its higher resolution and better visual information. The specific parameter configurations used for each camera type are listed in Appendix B.3.

- Once features are detected, the essential matrix is computed to estimate the relative pose between frames. This pose is refined through triangulation of matching feature points between the current and previous frames. The known camera height during data collection is used for scaling the estimation into realistic matrices by utilizing the median height of the triangulated 3D points. The camera matrices used for the IR and RGB cameras are the following:

$$K_{RGB} = \begin{bmatrix} 1485.52 & 0 & 924.39 \\ 0 & 1487.23 & 802.35 \\ 0 & 0 & 1 \end{bmatrix}, \quad K_{IR} = \begin{bmatrix} 764.7 & 0 & 320 \\ 0 & 764.7 & 256 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

This VO model uses a simplified structure to highlight trends and evaluate common patterns observed in standard monocular VO systems. As mentioned previously, the primary goal is to test the VO pipeline under varying conditions and feature detection methods. Therefore, no constraints were used on rotation or translation during evaluation, ensuring that both strengths and limitations of the system could be observed clearly.

To accurately evaluate the strengths and weaknesses of each feature detection model, in addition to estimating a full closed trajectory, three distinct test scenarios were implemented: straight-line walking, sideways walking, and in-place rotation. These tests were designed to isolate and examine how each model handles different types of motion, specifically translational and rotational movements. By analysing the results of these scenarios, insights can be gained into the performance and limitations of each model.

3.5 PointNet

The PointNet model has been designed as the originals authors model, see Section 2.8.1, with no modifications and has been implemented using pytorch. The input size, n, has been set to 4096 with consideration to the trade-off between accuracy and computational efficiency. The output size, m has been set to 14, resembling the different classes from the dataset S3DIS, see Section 2.9.1.

The training algorithm used is Adaptive Moment Estimation (Adam optimizer), it utilise Stochastic gradient decent (SGD) together with momentum and RMSProp. Learning rate decay was used during training, with a multiplication factor [γ] of 0.5 was used and applied every 20 epochs, hence the learning rate gets halved every 20 epochs. During the training a batchsize [B] of 16 was used, regulated after the device which the training was performed on. The initialized learning rate [α] was set to 0.005 and was trained for 150 epochs. A more detailed summary over the parameter used can be seen in Table 3.4. The final model was decided by taking the model during the training which had the best validation accuracy to avoid overfitting.

To evaluate how well the model performs on the specific use case and collected dataset, interference was performed by selecting frames of interest across varying levels of difficulty. This will help identify under which circumstances the model fails. Each frame will undergo the same data augmentation as the training data with the exception of adding noise and random rotation. This will ensure consistency during evaluation.

Table 3.4: Summary over parameters use in training for PointNet.

Parameter	***Value***
Input size $[n]$	[redacted]
Output size $[m]$	[redacted]
Epochs	[redacted]
learning rate $[\alpha]$	[redacted]
Batch size $[B]$	[redacted]
Training algorithm (Adam optimizer)	
Decay rate (1:st momentum) $[\beta_1]$	[redacted]
Decay rate (2:nd momentum) $[\beta_2]$	[redacted]
weight decay $[\lambda]$	[redacted]
Stabilization factor $[\epsilon]$	[redacted]
Learning rate decay (Step decay)	
gamma $[\gamma]$	[redacted]
step size	[redacted]

3.5.1 Loss Function

It's important to choose a relevant loss function for the training to efficiently work. It tells the model how good its predictions are and changes its weights accordingly. The loss function used for training the PointNet is shown in equation (3.4).

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cls}} + \frac{\alpha}{B} \sum_{i=1}^{B} \left(\left\| \mathbf{I}_3 - \mathbf{T}_{3\times3}^{(i)} \cdot \left(\mathbf{T}_{3\times3}^{(i)} \right)^\top \right\|_F + \left\| \mathbf{I}_{64} - \mathbf{T}_{64\times64}^{(i)} \cdot \left(\mathbf{T}_{64\times64}^{(i)} \right)^\top \right\|_F \right) \tag{3.4}$$

where $\mathcal{L}_{\text{cls}}$ is the negative log-likelihood classification loss, $\mathbf{T}_{3\times3}^{(i)}$ and $\mathbf{T}_{64\times64}^{(i)}$ are the transformation matrices predicted for the i-th sample in the batch. $\mathbf{I}_3$ and $\mathbf{I}_{64}$ are the identity matrices of size 3×3 and 64×64, respectively. $\| \cdot \|_F$ denotes the Frobenius norm. B is the batch size, and α is a regularization weight.

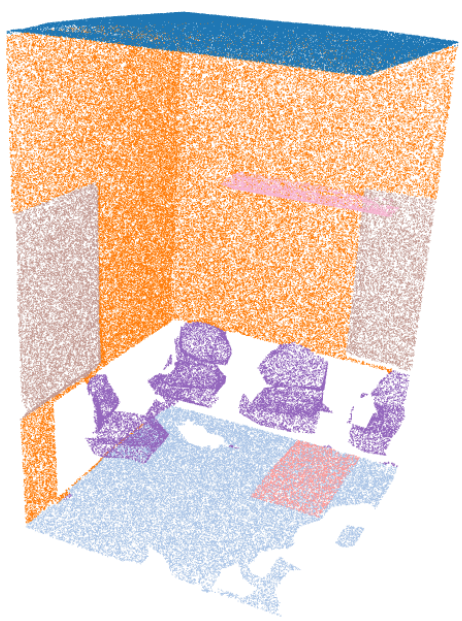
In simple terms $\mathcal{L}_{\text{cls}}$ encourages correct class predictions, while the term $\|\mathbf{I} - \mathbf{T}\mathbf{T}^\top\|_F$ penalizes transformation matrices predicted by the T-Nets that deviate from orthogonality.

3.5.2 S3DIS (Dataset)

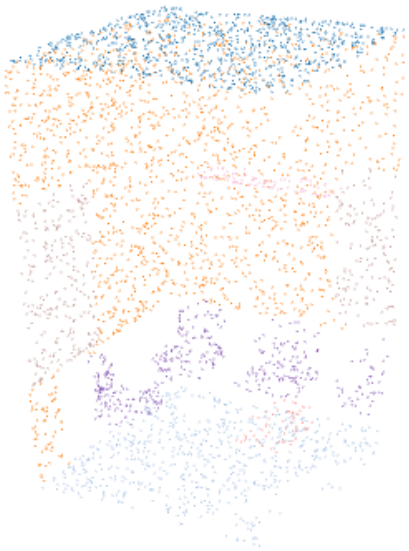
The dataset used to train the model has been preprocessed into smaller segments, which serve as input samples. Each room in the dataset, Figure 2.6, has been divided into 2×2 meter blocks along the x- and y-axes, extending fully along the z-axis, see Figure 3.9. Since the model expects an input size of 4096 points, each training sample consist of 4096 randomly drawn points form a block. To improve generalization and robustness, data augmentation is applied to all training samples. This includes normalization, a random rotation around the z-axis, and the addition of Gaussian noise, equation (3.5). For testing the model only normalization is performed. These augmentation steps help the model become more robust to variations in orientation, scale, and sensor noise. The division between the training, validation, and test sets follows the structure of the dataset. Areas 1–4 are used for training, area 6 is used for validation, and area 5 is reserved for testing. The noise added is presented below.

$$\eta_i \sim \text{clip}\left(\mathcal{N}(0, \sigma^2),\ -c,\ c\right) \tag{3.5}$$

where i is the ith point in the training sample, $\mathcal{N}(0, \sigma^2)$ is a Gaussian distribution with mean 0 and variance σ^2, σ is set to 0.01, and $\text{clip}(x, -c, c)$ limits the value of x to the range $[-c, c]$, c is set to 0.05.



(a) Visualization over one block of the customized S3DIS dataset.



(b) Visualization over one training sample (4096 points from the training sample)

Figure 3.7: Illustration of the customized S3DIS dataset.

3.6 YOLO

For the object detection task, the YOLOv8n model provided by the Ultralytics Python library is utilized [70]. To reduce development time and leverage existing knowledge, the adoption of a transfer learning approach by starting from a pre trained model trained on the COCO dataset, Section 2.9.2, was used. This enables to fine tune the model on the specific usecase, allowing for development of a more robust and efficient model tailed to the specific usecase, even with a relatively small dataset.

In total, three fine tuned models were created. The first two models were trained on open source datasets customized to better fit the project, Sections 3.6.2 and 3.6.3, each with a distinct purpose: to evaluate how well the model performs without using any data specific to the application and usecase. The third model, in contrast, was fine tuned on the self created dataset collected from test three, Section 3.1.3, which contains only data relevant to our use case. This comparison allows us to assess the benefits of using usecase specific data versus generic open source datasets.

The training process is conducted using the Ultralytics training framework, which provides a good integrated pipeline for fine-tuning YOLO models. This allows for better training and reduces the implementation effort. The loss function used during training is a common loss function for object detection and can be seen in equation (3.6). To mitigate the risk of overfitting, early stopping is implemented. A patience-based approach is used, if the validation accuracy does not improve for 15 consecutive epochs, the training process is stopped. The model with the highest validation accuracy observed during the training is then selected as the final model. In addition, Ultralytics performs a range of data augmentation and preprocessing techniques during training to improve performance. These augmentations is presented below [71].

- Mosaic augmentation: Combines 4 training images into one.
- HSV (Hue, Saturation, Value) augmentation: Randomly adjusts hue, saturation, and brightness of the image.
- Scaling: Random resizing of images while keeping the aspect ratio.
- Translation: Random shifts along x and y axes.
- Flipping: Horizontal flipping the image.
- Shear and rotation: Minor geometric distortions to simulate real-world variations.

The architecture of the YOLO model follows the exact same design as illustrated in Section 2.8.2 with the exception that the input image size has been modified to be 416 instead of 640, resulting in a input size $[n]$ of 173 056. The output size $[m]$ has been set to the specific dataset used to fine-tune the model, 3 classes for the Coco dataset, 16 for the Flir dataset and one for the self created. The parameters used in the model and its training phase are illustrated in Table 3.5.

Table 3.5: Summary over parameters use in training for YOLO.

Parameter	***Value***
Input size $[n]$	
Output size $[m]$	
Epochs	
Image size	
Patience (early stopping)	
Learning rate $[\alpha]$	
Learning rate fraction (used for decay) $[\alpha_{frac}]$	
Batch size $[B]$	
Training algorithm (Adam optimizer)	
Decay rate (1:st momentum) $[\beta_1]$	
Decay rate (2:nd momentum) $[\beta_2]$	
Weight decay $[\lambda]$	
Stabilization factor $[\epsilon]$	

The evaluation of the models has been performed by looking at the accuracy measurements mAP50 and mAP50-95 on the self created dataset from test three, Section 3.1.3, together with the visual evaluation of several interesting images. The following sections present the ultralytics loss function together with the dataset used for training the models.

3.6.1 Ultralytics Loss Function

The total loss $\mathcal{L}_{\text{total}}$ used for training in object detection at ultralytics consists of three main components: box (localization) loss, objectness loss, and classification loss. It is defined as:

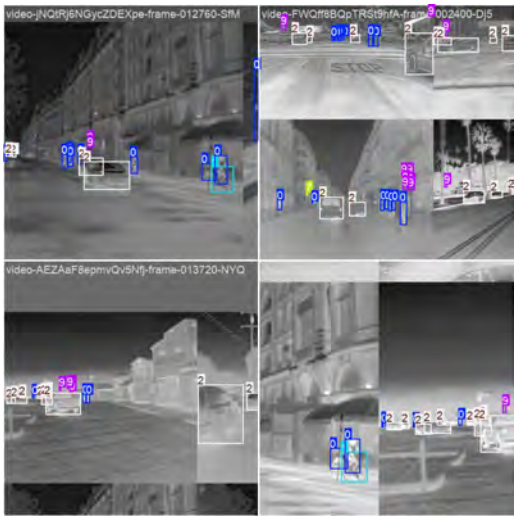
$$\mathcal{L}_{\text{total}} = \lambda_{\text{box}} \cdot \mathcal{L}_{\text{box}} + \lambda_{\text{obj}} \cdot \mathcal{L}_{\text{obj}} + \lambda_{\text{cls}} \cdot \mathcal{L}_{\text{cls}} \tag{3.6}$$

where $\mathcal{L}_{\text{box}}$ is the box regression loss, calculated using the complete IoU (CIoU) loss. $\mathcal{L}_{\text{obj}}$ is the objectness loss, computed using binary cross-entropy. $\mathcal{L}_{\text{cls}}$ is the classification loss, also computed using binary cross-entropy. $\lambda_{\text{box}}, \lambda_{\text{obj}}, \lambda_{\text{cls}}$ are weighting factors for each loss component.

3.6.2 Flir Dataset

The FLIR dataset has not undergone any preprocessing prior to the training augmentation. It was selected due to its relevance to the application, as it consists of pre-labelled thermal images that closely resemble the type of data used in the project. Since the data is already labelled and clean, evaluating how the model performs after being exposed to thermal characteristics is essential to the usecase. In Figure 3.8, an image of the final trainingset together with a training example is illustrated.

(a) An image from the flir dataset.



(b) A training sample of the flir dataset after data annotation and augmentation

Figure 3.8: Illustration of the FLIR dataset.

3.6.3 COCO Dataset

The dataset used for fine-tuning the model has been modified to better suit the specific usecase. Since the COCO dataset, see Section 2.9.2, has 80 semantic classes and most of them aren't relevant for the usecase, the dataset has been customized to only include images with people, dogs and cats, resulting in three classes. The images have also been transformed to mimic thermal images, transforming the RGB image to gray scale using the formula below [72]:

$$\text{Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{3.7}$$

where R, G, and B represent the red, green, and blue channel intensities in the original RGB image. The final step in mimicking the thermal image is to normalize the brightness and increasing the contrast. This is performed by using histogram equalization, see Section 2.4.1. In Figure 3.9, an image of the final trainingset together with a training example is illustrated.

(a) An image from the customized COCO dataset.

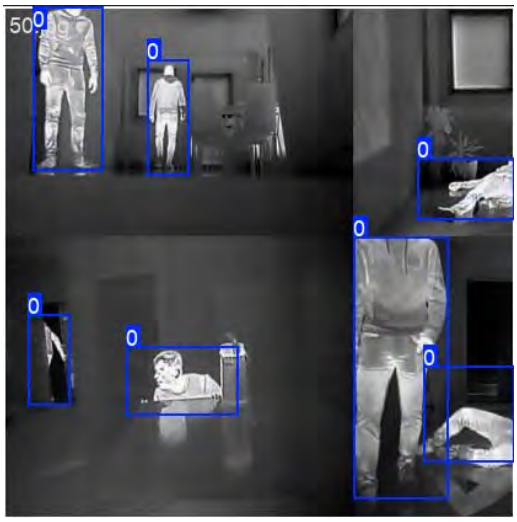
(b) A training sample of the COCO dataset after data annotation and augmentation

Figure 3.9: Illustration of the customized COCO dataset.

3.6.4 Self Created Dataset

The resulted images from the third data collection, Section 3.1.3, was manually annotated with squares holding only the label "person" with an open source software [73]. The collected data set only includes people, which is the most important object to be able to detect in the usecase. Since the implementation of the model has been through Ultralytics and the training as well, the data augmentation that was used can be seen in Section 3.6. An image from the dataset together with a training example after the augmentation is shown in Figure 3.10.

(a) An image from the collectable dataset.



(b) A training sample of the self created dataset after data annotation and augmentation

Figure 3.10: Illustration of the self created dataset.

3.7 Hardware

To enable comprehensive data collection throughout the project, new sensors were continuously integrated to improve the quality of the test data. In parallel with software development, a prototype was developed by an external part to provide smooth data recording. This prototype incorporates the final sensor setup including LiDAR, Thermal, and RGB sensors, all connected to a minicomputer along with supporting components, see Section 2.10. The complete prototype is shown in Figure 3.5a. Additionally, a system architecture was implemented on the prototype to host the final proposed system.

3.8 Software

The projects aim is to investigate how the different sensors can be combined to complement each other. Therefore, the design of the overall system is developed, to be efficient and allow information from different sensors to be used together. In Figure 3.11 the overall system flow chart can been seen. The solid boxes illustrates the investigated modules in the project while the dotted boxes illustrates modules in the system that haven't been investigated enough, and won't be part of the result but are important for the overall system. The reason why SLAM wasn't investigated further, was because of licensing issues. The given licence only allowed usage up to 4 weeks at the start of the project. The initial plan was to utilize a SLAM algorithm provided by a company called ExWayZ [74] and has been shortly investigated based on its suitability and relevance to the problem at hand.

Figure 3.11: Flow chart over the initially proposed system.

The development have been divided into two separate code bases, one utilizing Linux as a operating system meant to be deployed on the prototype and one using windows allowing for easy and efficient development. The Linux system utilizes the software library called Robot Operating System (ROS), which is a well know and a common open source library for developing robot applications. The ROS library supports development in both python and C++, which makes it suitable for the application [75]. It is also compatible with both the LiDAR, Section 2.10.1, and the initially proposed SLAM (exwayz). This code base holds the setup for ROS together with the implementation of the SLAM. The main development has been made in the windows system code base, since the main focus has been to investigate and analyse data, algorithms and compatibilities, which doesn't require any Linux integration. All the code in this code base is developed using python for easy and efficient development, allowing usage of open sourced libraries such as Pytorch and OpenCV [76, 77].

In this project, the version control system GitHub was used [78], enabling seamless collaboration and providing integrated support for documentation and code management.

4
Results

This section provides an overview of all the relevant results obtained from the methods outlined in the previous section. Highlights key trends, patterns, and anomalies.

4.1 Data Collection

As outlined in Section 3.1, four tests were conducted over the course of the project. This section provides a detailed overview of the data collected from each test, highlighting their key characteristics and distinguishing features.

4.1.1 First Test

This first test was performed with eight different smoke levels, and two different scenarios were recorded: static and dynamic. The collected data from smoke levels 0, 4 and 7 for the static case are presented in Figure 4.1. The figures show that the quantity and quality of LiDAR data are directly affected by the density of the smoke. As smoke density increases, light scattering becomes more common, leading to a reduction in valid LiDAR points. At lower smoke levels, false positive points are more common. Often due to reflective surfaces such as windows where points appear behind the actual structure. Additionally, in all smoke conditions, a dense cloud of points appears in front of the LiDAR, likely caused by backscattering from particles in the smoke.

(a) No smoke (Density level 0)

(b) Medium smoke (Density level 4))

(c) Heavy smoke (Density level 7)

Figure 4.1: Static recording from test one, showing different smoke densities.

The dynamic recordings used the same initial viewpoint as the static case, Figure 4.1, and involved rotation toward the most dense region of smoke in the room. The results for smoke levels 0, 4, and 7 are presented in Figure 4.2. As expected, when the LiDAR is oriented directly toward the densest part of the smoke, very few points are returned. At smoke level 4, mainly close points are detected; at level 7, almost no points are returned from within the smoke filled region. These results clearly illustrate the limitations of LiDAR performance in varying levels of smoke.

(a) No smoke (Denisty level 0)

(b) Medium smoke (Density level 4)

(c) Heavy smoke (Density level 7)

Figure 4.2: Dynamic recording from test one, showing different smoke densities.

4.1.2 Second Test

The second test aimed to collect more case specific data, thus performed in a smoke filled house. IR data was also collected for use in developing the sensor fusion model. An illustration of the house layout and the path taken during the recording is shown in Figure 4.3, with numbered locations representing the three key scenes analysed across different smoke levels.

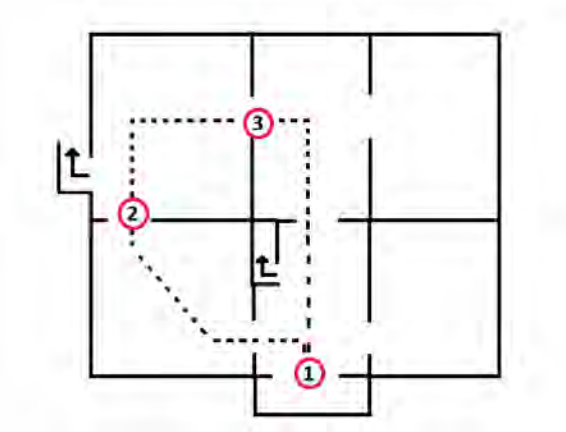
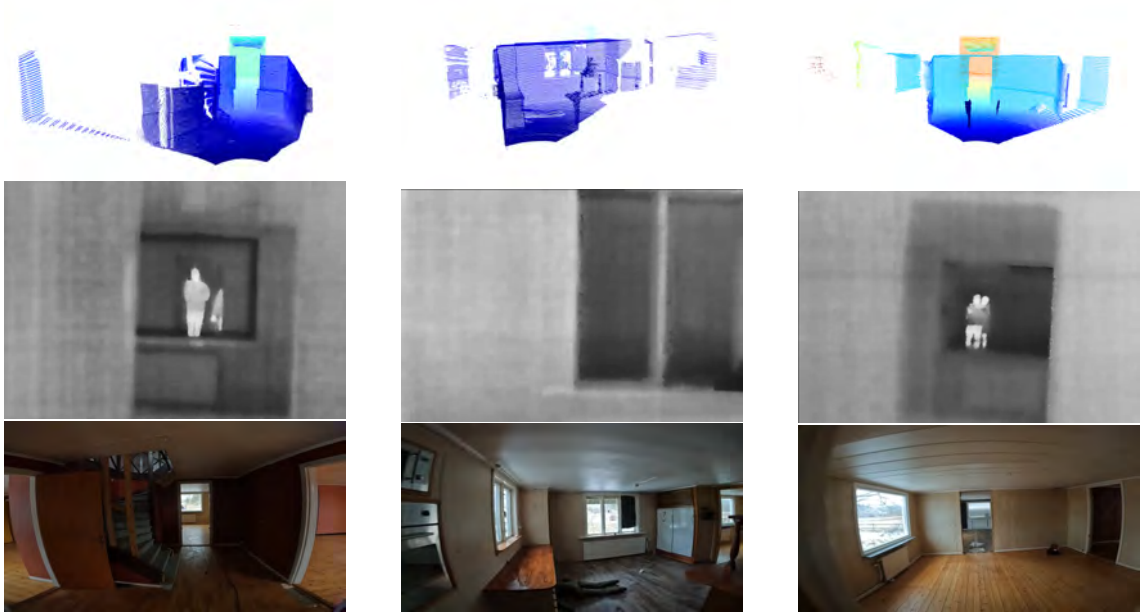


Figure 4.3: A map of the path taken during test 2 with the locations of the three scenes presented in this section.

The smoke-free condition is presented in Figure 4.4 for the three scenes, shown in Figure 4.3. While the LiDAR data includes some false positives due to reflections, mainly near windows, it generally captures the room structure well. However, the IR camera used in this test has significantly lower resolution than the RGB and LiDAR sensors, which limits its contribution to the 3D reconstruction process. Because both sensors were handheld during recording, some variability in their relative positions due to natural hand movement is present. This affects sensor alignment and contributes to data inconsistencies.



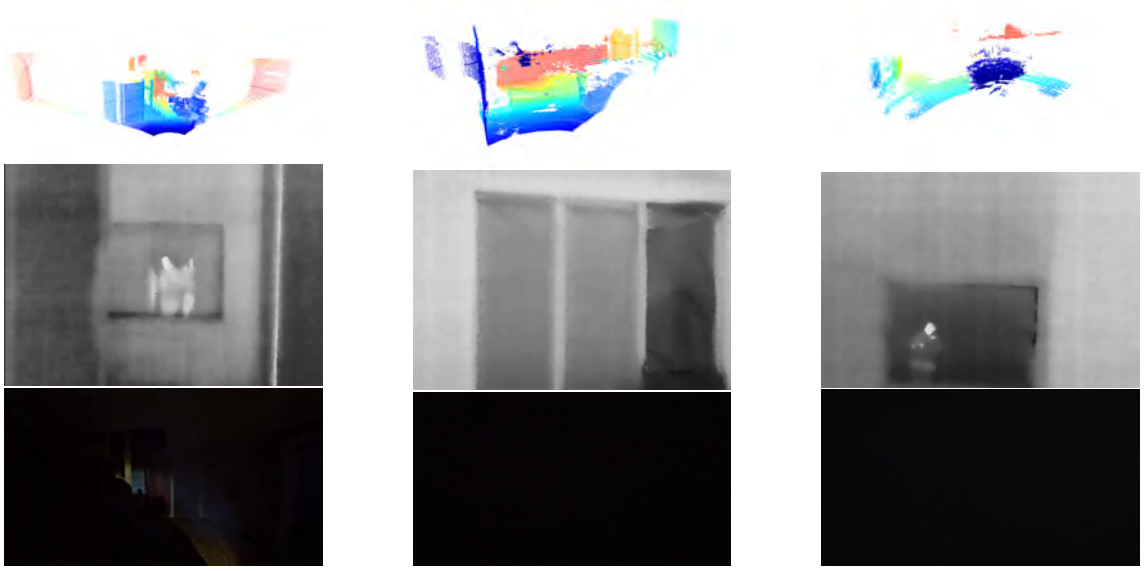
(a) Scene 1 **(b)** Scene 2 **(c)** Scene 3

Figure 4.4: Collected data from LiDAR, IR and RGB from test 2 without smoke (Density level 0).

The second test is with smoke density 1 is shown in Figure 4.5. It can be observed that the RGB camera barely picks up any information due to low light levels inside the house. However, this also shows that the LiDAR and IR sensors are fully operational even without much light. It can be observed from the IR data that it is minimally affected by the smoke and shows similar results as the smoke free test. It can also be seen from the LiDAR that in cases where the sensor is angled towards the main smoke filled area (the room of scene 3), loses a large amount of points. From Subfigure 4.5a it can be observed that as well as losing data from the room with a high smoke level, it also captures a large smoke cloud which greatly disrupts performance. For frame 2, it still manages to get a decent visualisation of the parts of the room which are closest to the LiDAR.

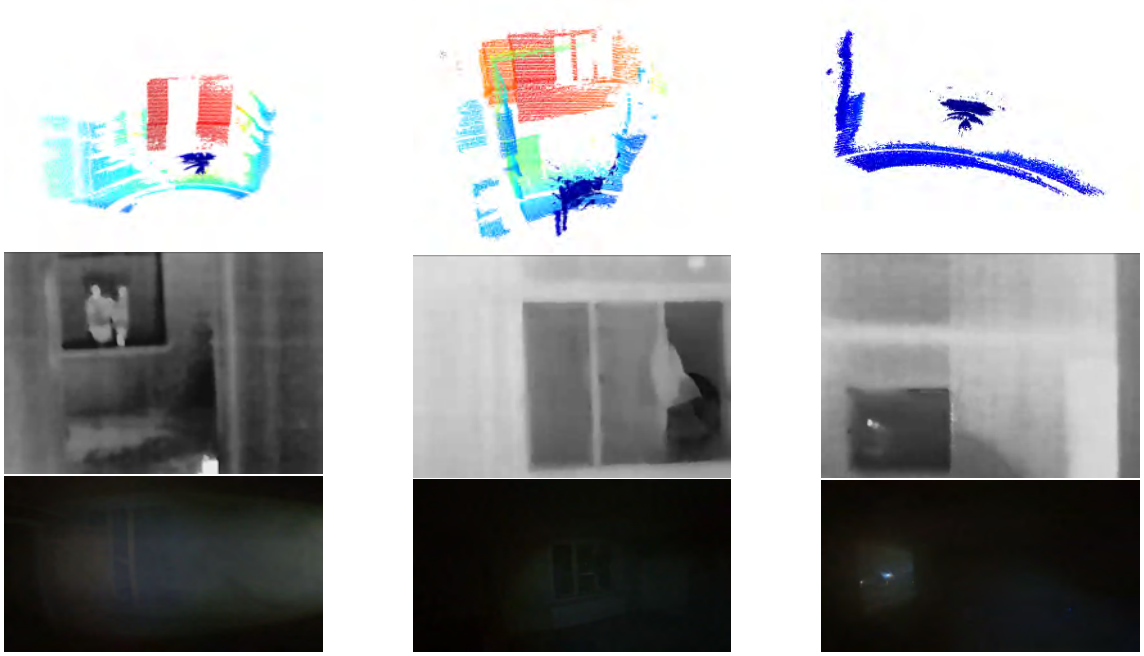
Figure 4.6 presents the results for uniform smoke (Density level 2). Here, the smoke is more evenly distributed throughout the house. The LiDAR detects only close objects across all three frames, indicating moderate data loss. While scenes 1 and 2 still give rough visualizations of the room layout, scene 3 shows a near complete

failure of point return. These results suggest that the LiDAR tolerates smoke levels to a certain extent and begins to fail at denser smoke levels.



(a) Scene 1 **(b)** Scene 2 **(c)** Scene 3

Figure 4.5: Collected data from LiDAR, IR and RGB from test 2 with varied smoke (Density level 1).



(a) Scene 1 **(b)** Scene 2 **(c)** Scene 3

Figure 4.6: Collected data from LiDAR, IR and RGB from test 2 with uniform smoke (Density level 2).

4.1.3 Third Test

Samples from the data collected during the third test are shown in Figure 4.7. A total of 152 images were obtained from this test to create a training dataset for the object detection model. The thermal images clearly distinguishes human bodies from their surroundings, due to the strong thermal contrast caused by body heat. The IR camera also performs well at varying distances, with minimal changes in detection quality. This is illustrated by the clarity of both the people in Figure 4.7a and the person near the door in Figure 4.7b. It is important to note that this particular test was conducted in a smoke free environment and smoke affected data is not part of the dataset.

(a) Image 1 **(b)** Image 2 **(c)** Image 3

Figure 4.7: Collected thermal images from test three.

4.1.4 Fourth Test

The results from the fourth and final test are shown in Figures 4.8 and 4.9, which display data collected at four different smoke density levels across two different scenes. As observed in previous tests, LiDAR data loss increases with smoke density. In this test, high quality RGB data was also collected, allowing for a easier visual comparison of how smoke impacts LiDAR performance. In particular, Figure 4.8d shows that the LiDAR struggles to find any points at smoke densities where the room becomes barely visible to the human eye.

Figure 4.9 further confirms that even under the highest smoke levels, the LiDAR can still register nearby objects, such as walls, when positioned very close. However, for the rest of the room, it consistently fails to return meaningful data. This confirms the conclusion that heavy smoke presents a repeatable challenge for LiDAR based sensing.

(a) No smoke (Density level 0)

(b) Light smoke (Density level 1)

(c) Medium smoke (Density level 2)

(d) Heavy smoke (Density level 3)

Figure 4.8: Collected data from LiDAR, IR and RGB from test 4 with multiple smoke levels for scene 1.

Looking at the sensor data from the IR camera, its performance can be seen to decrease slightly while the smoke level increases. In Figure 4.9a, the contrast of the window are clearly distinguished and vanish in higher smoke levels, see Figure 4.9.

(a) No smoke (Density level 0)

(b) Light smoke (Density level 1)

(c) Medium smoke (Density level 2)

(d) Heavy smoke (Density level 3)

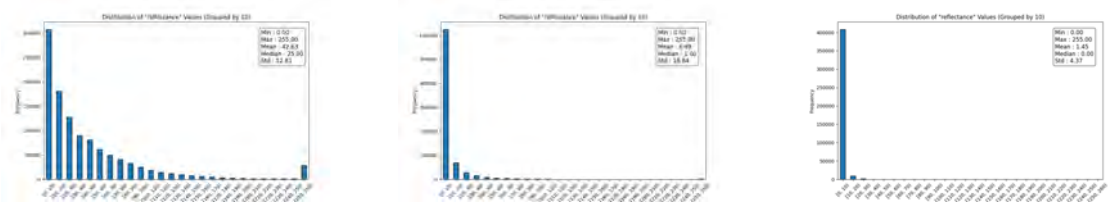
Figure 4.9: Collected data from LiDAR, IR and RGB from test 4 with multiple smoke levels for scene 2.

4.2 Data Enhancement/Filtering

This section presents an overview of the results obtained from the data enhancement process. It includes a detailed analysis of the enhanced data, as well as an evaluation of the filtering method applied.

4.2.1 Analysis of LiDAR Sensor

The results from the reflectivity analysis are used in the filtering process and help to identify and distinguish the necessary requirements. From Figure 4.10, it can be shown that the reflection attribute based on the intensity from the LiDAR sensor, see Section 2.10.1, is changing with the smoke level. With no smoke, Figure 4.10a, the distribution of the sensor data shows characteristics of a decreasing exponential distribution with a couple of outliers at interval 250-260 (255 maximum value of the reflection). The mean is 42.63, resulting in a exponential density function with $\lambda = 0.023$. The reflectivity decreases drastically when smoke is introduced, even for lower smoke densities (density 1), the distribution collapses sharply towards zero and the values of the reflections reduces such that the mean becomes 6.98, see Figure 4.10b. The values get even closer to zero as the smoke levels increase and the mean at density 7 takes the value of 1.45, see Figure 4.10c.

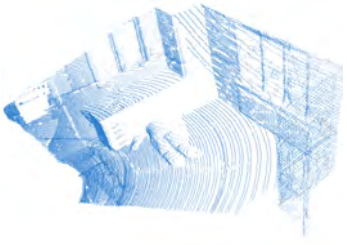


(a) No smoke **(b)** Smoke density 1 **(c)** Smoke density 7

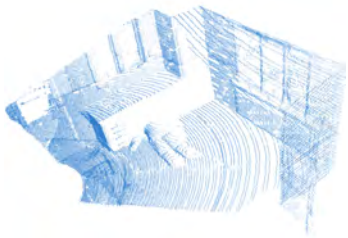
Figure 4.10: Bar plots of the reflection attribute for the LiDAR from Test 1.

4.2.2 Voxel Downsampling Analysis

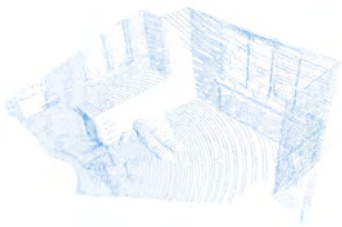
To determine an appropriate voxel size for downsampling, the results are analysed and presented in Figure 4.11. It can be observed that the geometric shapes of the original point cloud are well-preserved at smaller voxel sizes ranging from 0.01 to 0.05, Figures 4.11b - 4.11c. The structure of the scene remains intact and key architectural elements are visible. When the voxel size increases to 0.1 (10 cm), as shown in Figure 4.11e, the resolution is significantly reduced and the scene becomes sparse and most geometric details are lost, resulting in the overall structure no longer being visible.



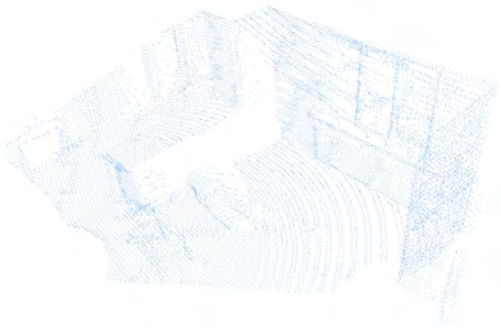
(a) Original point cloud



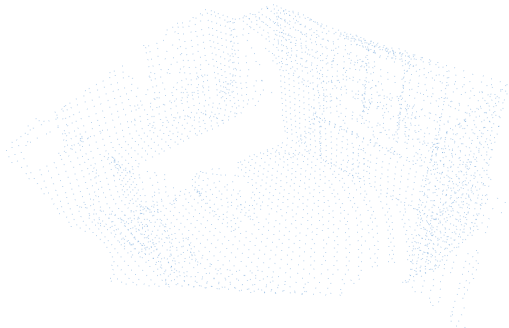
(b) Voxel size:0.01 (1cm)



(c) Voxel size:0.03 (3cm)



(d) Voxel size:0.05 (5cm)



(e) Voxel size:0.1 (10cm)

Figure 4.11: Voxel downsampling effect on LiDAR point cloud resolution at various voxel sizes.

To assess the trade-off between computational efficiency and resolution, Table 4.1 presents the number of points in the downsampled 3D point clouds. Since the computational cost of processing 3D point cloud data is mainly determined by the number of points, reducing the point cloud size is crucial for performance optimization. Even with a small voxel size of 0.01 (1 cm), the number of points is decreased by more than 20%, while preserving nearly all geometric shapes of the scene. As the voxel size increases, the size of the downsampled 3D point cloud gets reduced. At a voxel size of 0.03 (3 cm), the point cloud size is reduced by over 70%, and at 0.05 (5 cm), the reduction exceeds 87%. It is important to note that this analysis was performed on a standard-sized indoor room, typical of a residential home. In the project usecase even bigger rooms, such as warehouses, can be encountered.
These results demonstrate that voxel downsampling can significantly reduce data size without immediately compromising the overall scene geometry and resolution.

Table 4.1: Number of points in the point cloud after voxel downsampling.

Voxel size	***num points***	***fraction of original***
original	*97124*	*1*
0.01 (1cm)	*76937*	*0,7921*
0.03 (3cm)	*28444*	*0,2928*
0.05 (5cm)	*13499*	*0,1389*
0.1 (10cm)	*4193*	*0,0431*

The most important aspect to consider when choosing a voxel size its the trade-off between computational speed and preserving the geometric information. The result shows that voxel sizes between $0.01 - 0.05$ gives a good trade-off. Voxel sizes outside theses ranges, either result in to much loss of important features or not allowing the algorithm to be performed in real-time.

4.2.3 Filtering of LiDAR Data / 3D Cloud

This section presents the results of the LiDAR data filtering algorithm applied to scenes with different levels of smoke. The filter aims to remove noisy or unreliable points from the 3D point cloud, which are often introduced by smoke interference. In the visualizations, blue points represent the kept 3D point cloud (after filtering) while red points denote the removed points. The frames shown were selected to highlight challenging frames in which filtering plays an important role for enhancing the usability of the LiDAR point cloud.

(a) Frame from the forth data collection

(b) Frame from the second data collection

Figure 4.12: Visualization of the LiDAR filtering in no smoke.

Figure 4.12 shows the filtering results for scenes with no smoke. It can be observed that the LiDAR frames have false positives points, which are created by windows (reflections, goes through windows). These are removed by the filtering algorithm and can be shown as red points. This demonstrates the robustness of the algorithm even in scenes without smoke. It can also distinguish isolated clusters that do not belong to the main scene geometry. In these frames the removed points typically form smaller independent clusters which allow the algorithm to remove them.

(a) Frame from the fourth data collection

(b) Frame from the second data collection

Figure 4.13: Visualization of the LiDAR filtering in medium smoke.

Figure 4.13 illustrates the filtering applied to a scene with medium smoke. It show reduction in point cloud resolution, mainly due to scattering effects and the appearance of clusters in front of the sensor caused by the smoke. In Figure 4.13a the

algorithm mistakenly removes a cluster of true positives points, the red cluster to the right. The reason is that the cluster is separated from the main cluster and isn't a part of three of the six dominant planes, detected in the scene. This illustrates a limitation in the algorithm where it struggles to distinguish smaller separated clusters that are indeed valid parts of the scene. This can be compared with Figure 4.14a, where it shows a similar separated cluster of true positive points. The key difference is the classification of the smoke level. This frame is classified as lower smoke level, leading the algorithm to apply a less aggressive filtering strategy. This illustrate the importance of carefully selecting the smoke level threshold $[\tau_s]$ in the algorithm. The algorithm proves robust in the removal of the false positive clusters in front of the sensor, caused by the smoke, which can be seen in both Figures 4.13 and 4.14.

(a) Scene 1 from the second data collection

(b) Scene 2 from the second data collection

Figure 4.14: Visualization of the LiDAR filtering in heavy smoke.

The overall result of the filtering algorithm, shows that it successfully removes a significant amount of noise and false positive points from the original point cloud. Its adaptability across both smoke-free and smoke-filled environments enables reliable frame-wise filtering. This robustness makes it a valuable preprocessing step for tasks that rely on clean and consistent 3D data from the LiDAR. However, the algorithm has limitations that needs to be addressed. The parameter choices do matter and need to be carefully tuned. For more visualizations from the filtering algorithm, see Appendix C.1.

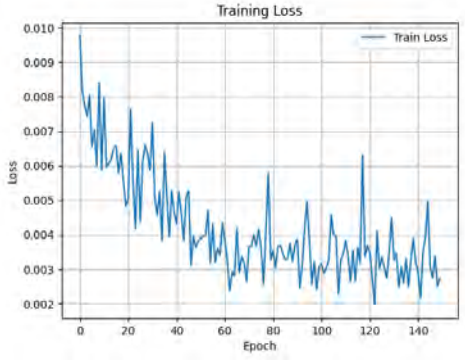
4.3 AI (Scene Understanding)

This section presents the results from the investigation into AI-based methods for environmental awareness, focusing on classification and segmentation techniques.

4.3.1 PointNet

The training of the PointNet model is illustrated in Figure 4.15. The plot of the training and validation accuracy, Figure 4.15a, indicates some degree of overfitting since the training accuracy continues to improve while the validation accuracy flattens out. However, since the final model selected is the one that achieved the best accuracy on the validation dataset, the risk of overfitting in the final model is minimized. The training loss, Figure 4.15b, further confirms convergence of the model.

(a) Plot over the training and validation accuracy



(b) Plot over the training loss

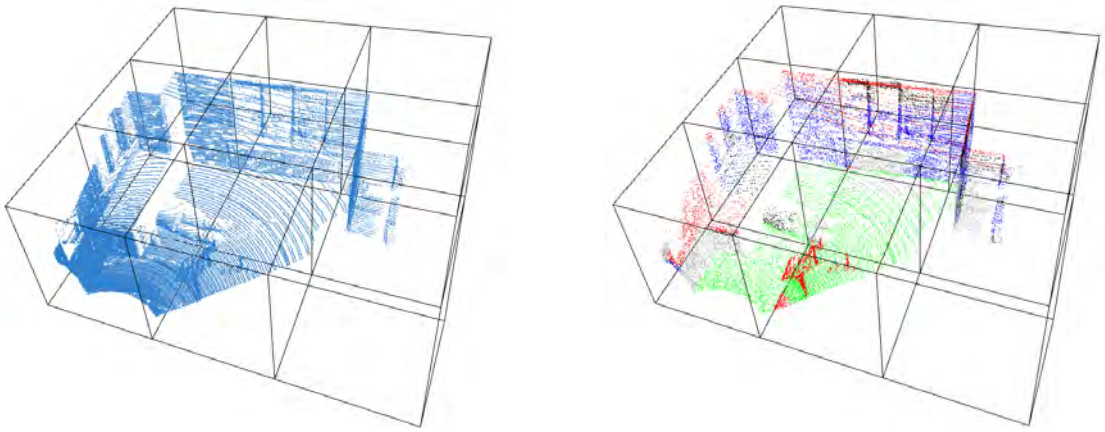
Figure 4.15: Illustration of the training phase.

The final PointNet model reached a validation accuracy at 73.83% on the S3DIS dataset, which highly demonstrates reasonable performance at first glance. After a closer look at the accuracy across the 14 semantic classes, It varies a lot, ranging from 0% to 99%, see Appendix B.1. This highlights a key limitation of the model's generalization to predict all class types. The PointNet model showed a good capability to predict classes representing larger structures in a room, such as ceilings, floors and walls, see Table 4.2. These results suggest that the model is effective at identifying large and distinct planar surfaces. This can be explained by the characteristics of the training dataset, which has similar structures (such as walls, ceilings and floors), see Figure 3.9.

Table 4.2: Class prediction for the top three class accuracies for PointNet on S3DIS dataset.

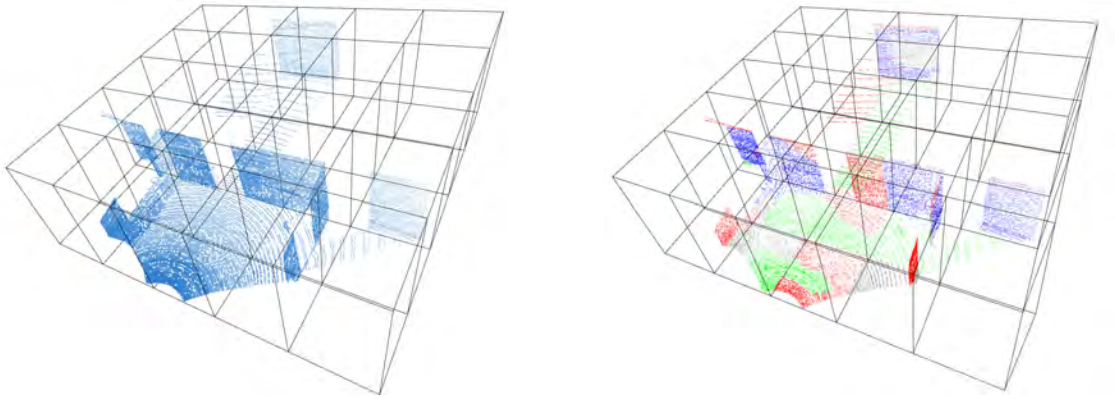
Class	***Accuracy***	***precision***	***recall***	***f1-score***
Ceiling	0.96	*0.95*	*0.96*	*0.95*
Floor	0.99	*0.94*	*0.99*	*0.97*
Wall	0.75	*0.61*	*0.75*	*0.67*

To allow comparison and usability for the model in the usecase, a visual evaluation on three different filtered point cloud frames has been made. In the figures, only classes mimicking large structures, such as ceiling (red), floor (green) and walls (blue), are predicted and the rest of the classes are merged (gray). This is a direct result from the varying class prediction on the S3DIS dataset. In Figure 4.16 a typical room from the second data collection illustrates that the model can reasonably predict the floor, as well as regions where the classification block contains both floor and ceiling (seen in the back of the room). The model performs even better in smaller spaces, shown in Figure 4.18, which presents a sample from the fourth data collection. In this sample most of the predicted blocks holds the typical structure of a room, having ceilings, floors, and walls, contributing to the model's strong performance. In contrast, for larger open areas where the prediction blocks lack a room structure, the model struggles with predicting the overall 3D point cloud, demonstrated in Figure 4.17.



(a) Original point cloud. **(b)** Prediction of the point cloud.

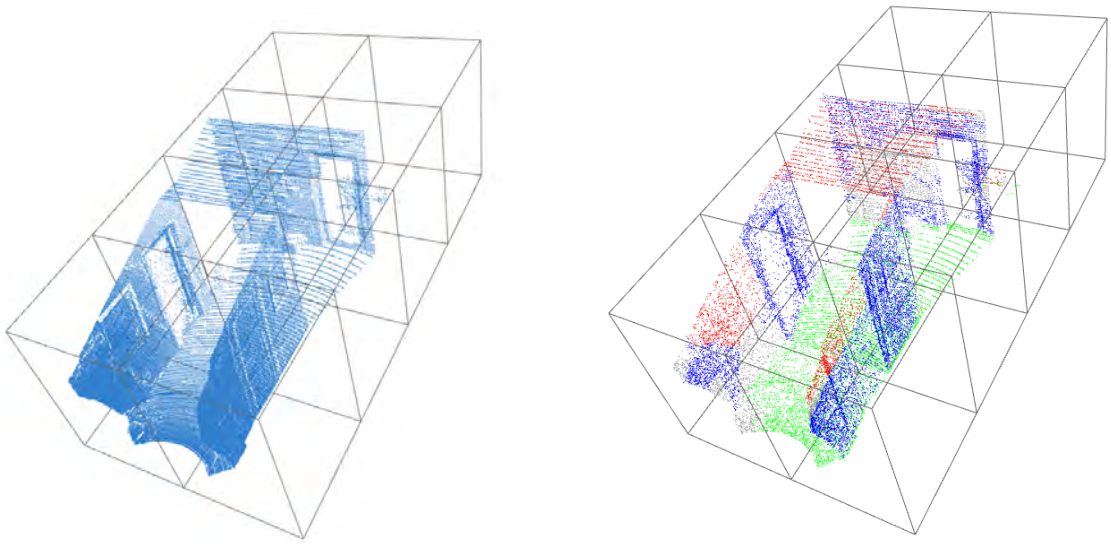
Figure 4.16: Visualization and prediction by the PointNet model on a normal sized room from the second data collection.



(a) Original point cloud. **(b)** Prediction of the point cloud.

Figure 4.17: Visualization and prediction by the PointNet model on a larger sized room/area from the second data collection.

These results indicate that it is possible to use the model for classifying the 3D point clouds, which is useful in both filtering and object detection/classification. The model performs well under the assumption that the predicted block in the 3D point cloud holds structural features from a room (ceiling, floor, wall). These often appear in smaller areas captured by the LiDAR. In the opposite, when the area becomes larger and the prediction blocks lose the structural information, the model fails to predict the overall 3D point cloud, see Appendix C.2 . The findings suggest that the model isn't reliable enough to be incorporated into the final system, but shows potential.



(a) Original point cloud. **(b)** Prediction of the point cloud.

Figure 4.18: Visualization and prediction by the PointNet model on a smaller sized area (hallway) from the fourth data collection.

4.3.2 Object Detection (YOLO)

In this section the result from the YOLO models will be presented. Each model converged and had consistent decrease in the loss matrices for both its training and validation dataset, indicating a stable training process, without any significant overfitting. The evaluation will be performed on the testset of the self created dataset, Section 3.6.4. This means that no model has seen the images before. The six images shown in the figures below have been selected, since they illustrate different scenarios with varying difficulty. The evaluation metrics can been seen in Table 4.3, where it's observed that the YOLO model trained on the flir dataset, Section 3.6.2, performed the worst. An important factor that needs to be addressed when comparing the result is that the flir dataset had the most amount of classes, 15 classes, compared to the custom Coco data set and self created dataset, which had 3 and 1 classes respectively. This makes it harder for the model to make accurate predictions for the usecase.

Table 4.3: Accuracy for the YOLO models validated on the test set for the self created dataset.

Trained on	***mAP50***	***mAP50-95***
Self created dataset	*0.968*	*0.561*
Custom Coco dataset	*0.733*	*0.366*
Flir dataset	*0.323*	*0.106*

The model trained on the self created dataset achieved a mAP50 score of 0.968, indicating that it classifies objects with very high accuracy. In contrast the mAP50-95 score of 0.561, reveals that the accuracy of the predicted bounding boxes is not optimal. This behaviour is illustrated in Figure 4.19 and more specifically, in the Subfigures 4.19c-4.19f, where multiple bounding boxes are being predicted. Although these boxes are not always tightly aligned with the ground truth, the correct class is still predicted. The model performs particularly well when the entire person is seen in the image and struggles slightly when only partial body parts are shown, but still succeeds in classifying the object correctly.

(a) Scene 1 (b) Scene 2 (c) Scene 3

(d) Scene 4 (e) Scene 5 (f) Scene 6

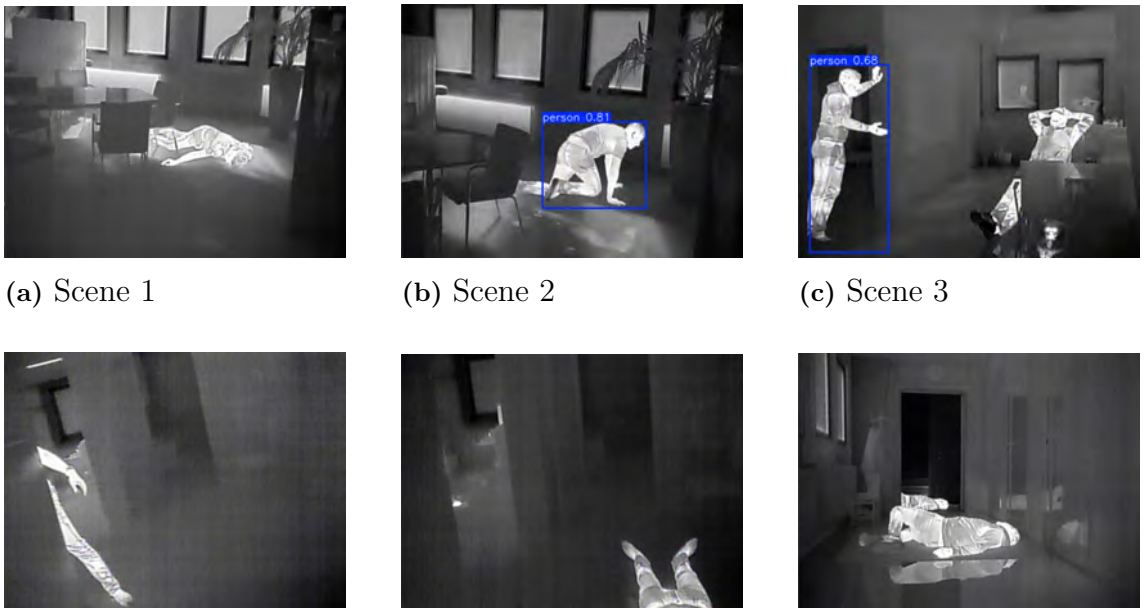
Figure 4.19: Object prediction with the YOLO model trained on the self created dataset.

The model trained on the flir dataset is illustrated in Figure 4.20. It can been observed that it classifies the objects incorrectly (motor, car) while also performing poorly in the bounding boxes for the correctly classified objects. This is also reflected in the models mAP50 and mAP50-90 score, Table 4.3. In contrast to the model trained on the customized Coco dataset isn't predicting wrong classes, but is unable to classify the object at all, see Figure 4.21.

(a) Scene 1 (b) Scene 2 (c) Scene 3

(d) Scene 4 (e) Scene 5 (f) Scene 6

Figure 4.20: Object prediction with the YOLO model trained on the Flir dataset.



(a) Scene 1 **(b)** Scene 2 **(c)** Scene 3

(d) Scene 4 **(e)** Scene 5 **(f)** Scene 6

Figure 4.21: Object prediction with the YOLO model trained on the custom COCO dataset.

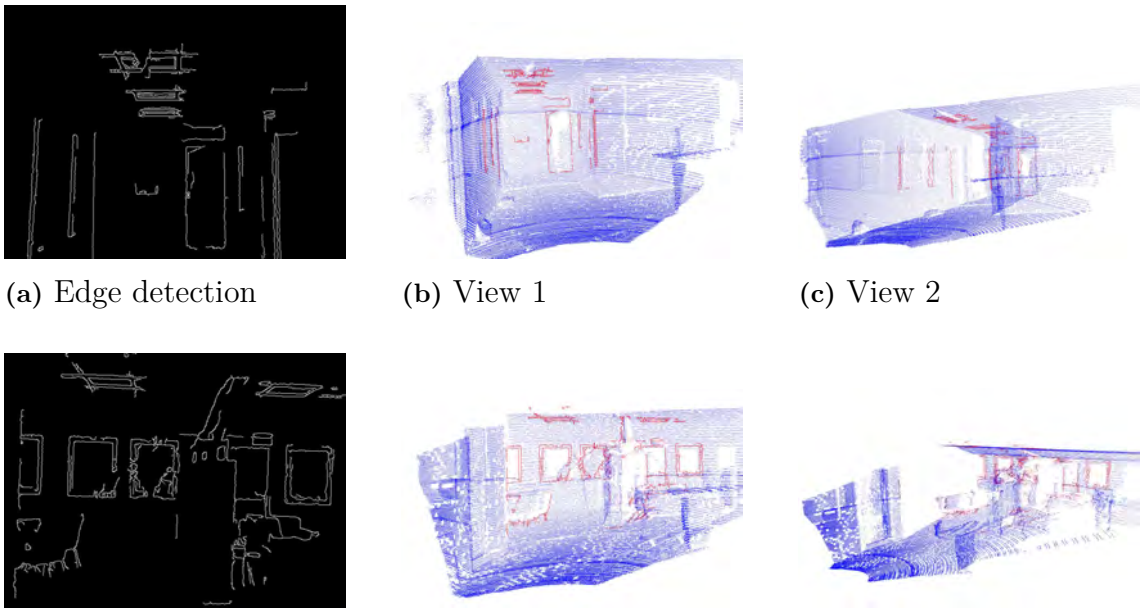
These results highlight the effectiveness of the dataset specific to the usecase and demonstrates that a focused dataset with fewer, represented classes can lead to much better results. Since it in the usecase is critical to minimize false negative prediction (missing a person), it illustrates the importance of robust and application specific datasets for object detection.

4.4 Sensor Fusion

This section presents results from the IR-LiDAR fusion model. Both 3D visualization and runtime performance is evaluated and presented. The observed cases were the smoke free scenario and the medium smoke density scenario from test 4. The highest smoke density was not used because the LiDAR failed to obtain points as visualized in Section 4.1.4

4.4.1 3D Visualization

In Figure 4.22, the results from the smoke-free IR–LiDAR fusion model are presented. In the figures, the red points represent the points obtained from the IR sensor and the blue points represents the filtered LiDAR data. It can be observed that most of the IR edge-detected points are correctly mapped to their corresponding 3D LiDAR points. Objects that are typically difficult to detect in the LiDAR point cloud, such as flat objects lying on flat surfaces, become clearly distinguishable in the fused model. For example, the flat lamps on the ceiling, which are barely visible in the raw point cloud, are clearly visualized after fusion with IR data. Slight misalignments can also be observed, caused by minor calibration errors in the thermal camera, see Section 3.3. However, these offsets are small and do not significantly affect the overall structure.



(a) Edge detection **(b)** View 1 **(c)** View 2

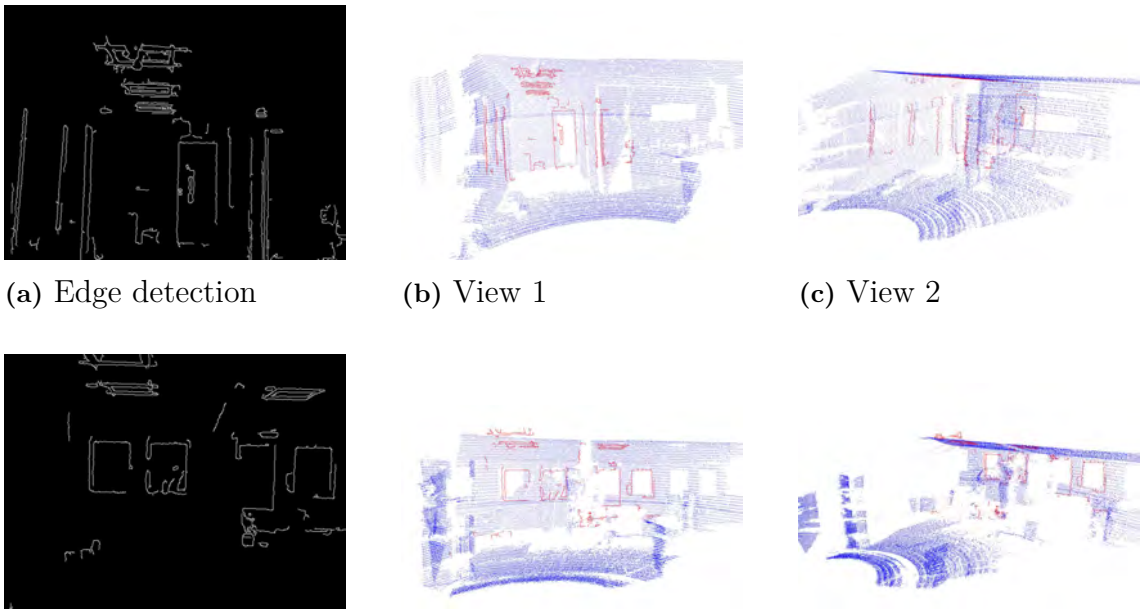
(d) Edge detection **(e)** View 1 **(f)** View 2

Figure 4.22: LiDAR-IR fusion without smoke (Density level 0) on two different scenes (a-c, d-f).

The fusion result under medium-level smoke conditions is shown in Figure 4.23. Most IR points still map accurately to the correct objects, although some points fall between multiple surfaces due to gaps in the LiDAR data caused by smoke. Despite

this, the overall system performance remains stable, indicating that medium smoke levels do not significantly degrade the fusion model. The model only begins to fail under heavy smoke conditions, where the LiDAR becomes unable to generate reliable point data, leaving the IR data with no corresponding depth references.

Additionally, in Figures 4.23e and 4.23f, some IR edge points appear to be positioned above the LiDAR point cloud, which in this case is a lamp. This is expected, as the IR camera is physically mounted above the LiDAR sensor, resulting in a wider vertical FoV. As a result, objects that are not visible to the LiDAR, can still be detected by the IR camera. This results in vertical points above the LiDAR point cloud, as these IR features have no corresponding depth values to map to within the LiDAR data.



(a) Edge detection **(b)** View 1 **(c)** View 2

(d) Edge detection **(e)** View 1 **(f)** View 2

Figure 4.23: LiDAR-IR fusion with medium smoke (Density level 2) on two different scenes (a-c, d-f).

The results also indicate that smoke has minimal impact on the performance of the edge detection algorithm. When comparing the smoke free data to the smoke filled data, it is observed that fewer edges are detected in scene 2 under smoke conditions, while more edges are detected in scene 1. This suggests that the presence of smoke does not consistently degrade edge detection performance. Instead, the observed variations are more likely due to inconsistencies in the IR camera's performance rather than the smoke itself.

4.4.2 Optimization

The execution time required for each step of the fusion model is summarized in Table 4.4. It can be observed that the fast model processes a single frame in approximately 0.119 seconds, while the slower, higher accuracy model requires 0.357

seconds. As mentioned in Section 3.3, the difference between the two models lies in the densification of the depth map. As shown in the table, this step accounts for the majority of the total computational time in both versions.

The results indicate that neither model fully meets the 0.1 second per frame threshold required for real-time performance. However, the fast model comes very close, suggesting that with further optimization, real-time execution may be achievable.

Table 4.4: Processing time comparison between the slow and fast sensor fusion models averaged over 10 iterations.

Processing Step	***Slow Model (s)***	***Fast Model (s)***
Sparse depth map	*0.011*	*0.012*
Densify depth map	*0.326*	*0.084*
Backproject	*0.003*	*0.004*
Thermal point extraction	*0.017*	*0.019*
Total	***0.357***	***0.119***

A figure showing the results from the fast model can be seen in Figure 4.24. It can be observed that the slow model in Figure 4.22 better aligns with the depths from the LiDAR sensor. The only difference between the models, as previously described, is the computed depth. Since the depth densification is downscaled to minimize computational time, it also has slight visual effects on the results as well. In Figure 4.24b, it can be seen that some of the points on the lamps are slightly offset from the roof. It can also be observed that the edges on the left and right walls are uneven, and this is due to the less thorough depth computation. However, with this result it is still easy to identify where the edges should be and what they represent. This means that even though the fast model is less accurate than the slow model, it still solves the problem and could, with further optimization, be used in real-time.

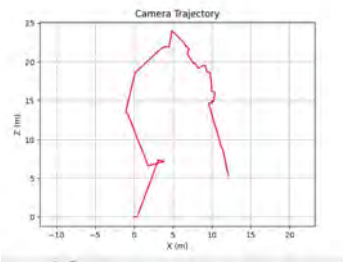
(a) Edge detection **(b)** View 1 **(c)** View 2

Figure 4.24: LiDAR-IR fusion without smoke (Density level 0) on two different scenes (a-c, d-f) for the fast model.

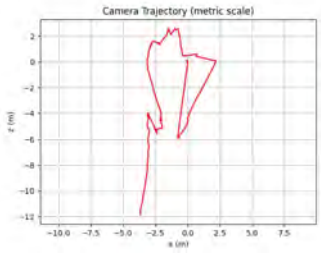
4.5 Monocular Visual Odometry

The results of both the RGB and IR cameras after applying the three feature detection methods are shown in Figure 4.25. The dataset used has the same start and end point, meaning that an accurate trajectory should return to its origin. The path used is path 2 in Figure 3.4b in Section 3.1.3. It can be observed that the IR camera is unable to produce a realistic trajectory with any of the tested detection methods. This is primarily due to the IR camera's low resolution, which results in unstable or insufficient features for tracking.

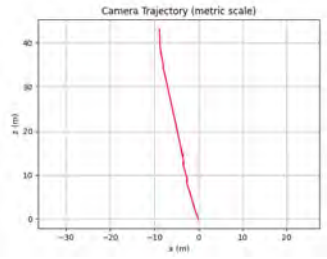
In contrast, the RGB camera generates a trajectory that closely resembles the ground truth path when using the KLT tracker, although large drift is present. The ORB-based model follows a trajectory similar to KLT but makes an incorrect final turn and exhibits more noise. The SIFT-based model maintains a realistic scale but fails to capture rotational motion, resulting in a near linear trajectory. These results suggest that KLT tracking is the most suitable method for VO in this context, as it performs frame-to-frame tracking without re-detecting features in each frame, which also contributes to its computational efficiency. Among the three methods, KLT produced a trajectory that was closest to the true path.



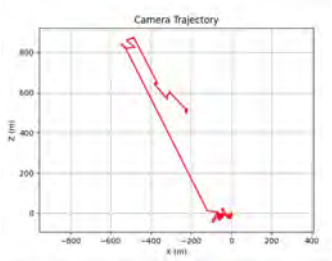
(a) KLT



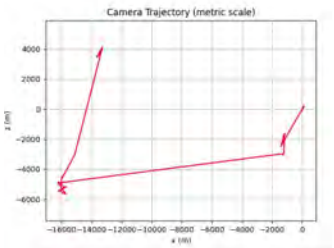
(b) ORB



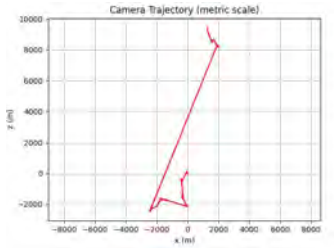
(c) SIFT



(d) KLT



(e) ORB



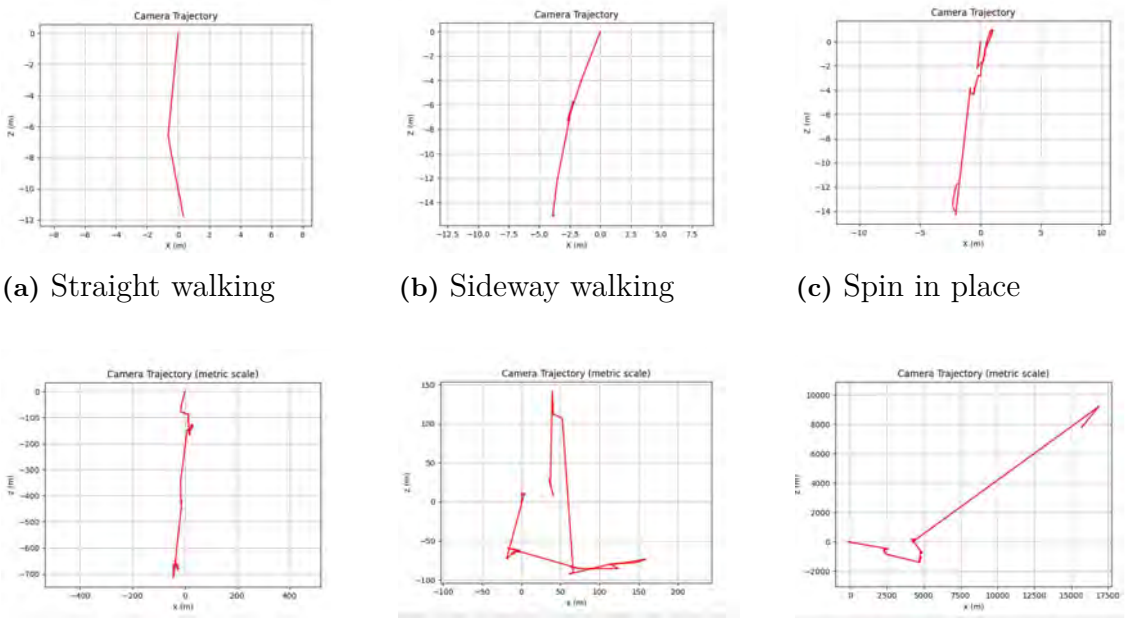
(f) SIFT

Figure 4.25: Generated trajectory from VO model with RGB (a-c) and IR (d-f).

To further evaluate where the models fail, three motion tests were made for both the IR and RGB cameras: straight walking, sideways walking, and rotation in place. The results of these tests are presented in Figure 4.26. For these evaluations, the RGB camera uses the KLT method, as it previously demonstrated the most realistic trajectory, see Figure 4.25a, while the IR camera uses ORB due to its low computational cost and resistance to noise.

The test results indicate that the RGB camera performs well during both straight and sideways motion, with only minor errors. However, it fails to accurately detect rotation in place, where no translational motion is present. The IR camera fails in all three motion scenarios, reinforcing the conclusion that standard monocular VO techniques are not suitable for low resolution thermal data, even when preprocessing is applied.

Overall, these findings highlight two key points. First, that a basic VO pipeline can give reasonable results with RGB cameras but is not effective with standard IR data. Second, that improvements in rotation estimation, mainly without translation, are necessary to reduce drift and enhance VO performance.



(a) Straight walking **(b)** Sideway walking **(c)** Spin in place

(d) Straight walking **(e)** Sideways walking **(f)** Spin in place

Figure 4.26: Generated trajectories for the three cases with RGB using KLT (a-c) and IR using ORB (d-f).

5
Discussion

In this section, a discussion will be presented regarding the findings and results obtained throughout the project. The aim is to interpret the outcomes in relation to the research objectives, assess the modules suitability for the application case and explore potential explanations for the observed patterns and inspected results.

5.1 Applicability to Real World Conditions

While the tests in this project were performed using cold smoke, the system's effectiveness with hot smoke must also be considered. Because smoke behaviour changes with temperature, the data collected by the sensors will also be affected. Cold smoke typically spreads uniformly throughout a room, whereas hot smoke tends to rise near the ceiling. This difference may significantly influence the system's performance.

Because of this, sensor placement becomes more critical in hot smoke environments. Since smoke density tends to decrease closer to the floor, LiDAR sensors positioned at lower altitudes may be able to capture more reliable data. This includes large scale objects such as the floor and lower walls, which can contribute to better depth estimation in the fusion model. The opposite applies if the sensors are mounted at higher elevations, then they are more likely to encounter dense smoke, which may lower the quality of the LiDAR data.

It is also important to note that the LiDAR used in this project has not been tested under hot smoke conditions. As such, its performance in these scenarios remains unknown. The LiDAR may perform better or worse than it did in cold smoke, depending on how the characteristics of the hot smoke affect light scattering and absorption. To evaluate this, future experiments should include testing across different smoke temperatures and possibly different LiDAR wavelengths to determine the system's performance under varying conditions.

Another important aspect is the system's applicability in real time scenarios. The current implementation was designed for offline analysis and does not operate in real time. This means that further optimization of each module would be necessary to reach fast enough performance to run in real time. The trade off between accuracy and processing speed must be evaluated to ensure real time applicability without lowering the performance by a large amount.

It is also possible that some modules developed in this project may not work effectively in a real time system. If this proves to be the case, modifications will be needed. Potential changes could include further downsampling of the LiDAR data or additional simplification of the depth map densification process in the sensor fusion pipeline. While these adjustments may reduce computational load, they would most likely come at the cost of reduced performance or accuracy.

5.2 Filtering of 3D Point Cloud.

The findings for the proposed filtering method illustrates that the algorithm is robust and generally meets the filtering requirements, Section 3.2.3. Its ability to remove false positives, caused by the impact of smoke and reflections in the surrounding enviromnent, is good but has limitations. One of the most important insights, given the evaluation, is the importance of parameter selection. While the algorithm performs well under normal conditions, it sometimes classifies false negatives (the algorithm removes true points from the point cloud). This issue happens due to the characteristics of the point cloud, where true points can appear in isolated clusters and become more common for higher smoke levels. When the point cloud becomes more fragmented, the importance of retaining each true point increases. Removing a small cluster of true points can impact the overall systems result negatively.

The overall system backbone, is build upon the LiDAR point cloud quality. It is well known that LiDAR performance decreases in the environments where airborne particles are present, like smoke or fog, reducing the point cloud density and accuracy. Preserving as much accurate information as possible from the decreased point cloud is crucial. The developed solution utilizes a smoke-level threshold to dynamically adjust the filtering based on the smoke level. This threshold must be sensitive to the environment, and is shown to need more tuning, since smaller rooms yields less points in the point clouds compared to larger rooms. This happens due to the characteristics of the LiDAR scan pattern and is even more enhanced by the downsampling.

The ability for the algorithm to filter point cloud data in real-time, is primarily based on the downsampling, which reduces the computational load. Since the algorithms speed is heavily dependent on the size of the point cloud and has a linearly scalable complexity, it makes it suitable for real-time applications. Processing fewer points increases the algorithm's speed, while processing more points enhances the accuracy of the results. Currently the filtering algorithm is implemented in Python, which is known for not being optimized for high-performance computing. An implementation in C++ could reduce the processing time and allow for increased accuracy.

An alternative method for filtering of 3D point clouds is the use of deep machine learning. The PointNet model evaluated in this project shows to have varying performance depending on the point cloud structure. The reason for the varying performance lies in the dataset it's trained on, which is illustrated in the evaluation, see Section 4.3.1. The models training dataset contains structures of entire rooms and

holds 14 unbalanced semantic classes, where walls, ceilings and floor classes are over represented, see Figure 4.10. In comparison to the LiDAR collectable data which holds fragments of room structures in an oval scan pattern, and does not necessarily have objects from the semantic classes. Additionally, interference from the smoke creates an even more sparse and fragmented point cloud. The current performance of the PointNet model indicates that it is not yet reliable enough to be integrated into the system or fully trusted in its decisions, but it proves the possibility of using an AI model for filtering. In the future, a well trained deep learning model is likely to be the best way for filtering 3D point clouds. Since it would allow for more generalized and adaptable filtering, tailored to the specific use case.

One important factor in the filtering is to consider how accurately cold smoke (theater smoke, used in this project) simulates the properties of real smoke, encountered in real scenarios such as fires. How will the LiDARs performance vary in different environment and how will the smoke change the scene? This raises the question of whether the current filtering method is robust and suitable in environments where real smoke from fires are present. Maybe new requirements will emerge which the filter algorithm must handle. After all, the design and tuning of the filtering algorithm are based on observations from the collected datasets in the project.

In summary, while the proposed filtering method performs well under many conditions, it is sensitive to point cloud sparsity in environments where the smoke is present. Its performance isn't perfect but is reliable enough to be implemented in a prototype. The only thing needed to be further refined is the adaptive thresholding, there might exist better ways to measure the smoke level. The use of a deep learning model is desired for the future, but the current developed model isn't yet robust enough to be relied on.

5.3 Object Detection

Object detection is a relatively small but critical component of the overall system and serves as a tool to assist and support firefighters. Given the high stakes nature of the application scenario, no compromising in performance is allowed. It is very important that the model is robust and can accurately makes predictions.

The final YOLO model demonstrates exceptional performance on data collected from the thermal sensor and allows for reliable detection of most individuals in both images and video sequences. This proves that the model can be trusted and integrated into the prototype without compromising the performance or safety. There are several important considerations that must be kept in mind. One key aspect is the coverage of the collected dataset, does it represent all possible operational scenarios? Even if the model performs well on generalized indoor scenes, its performance can vary if the application environment changes. For instance, if the scenario shifts to outdoor use. It's performance may decrease since the model is solely trained on indoor data. Another aspect to consider is the model's real time interference speed and hardware compatibility. The current model is a light weight

YOLO model allowing real time usage. It is meant to be able to be deployed on a smaller computer and should be able to run in real time with the current hardware.

An alternative approach to object detection within the system is to utilize 3D point cloud data. As previously mentioned the current PointNet model isn't performing on a reliable enough level to be trusted in the system. It does prove that the point cloud based detection is possible and could serve as a complementary method alongside the YOLO model. This dual system approach offers two main advantages:

- Wider Field of View (FOV): The LiDAR sensor has a broader FOV compared to the thermal camera, enabling detection over a larger area.
- Increases security and reliability: Having two independent detection systems increases robustness and reduces the likelihood of missed detections by the system.

5.4 Sensor Fusion

The final sensor fusion model shows good results given the limitations of the current implementation. The IR data is generally well aligned with the LiDAR data, and most fused points are positioned accurately. However, one major area for improvement is the calibration of the IR camera. In this project, camera calibration was approximated based on manufacturer specifications rather than manually calibrated. A proper calibration procedure would most likely improve alignment between IR features and LiDAR data. Similarly, the relative pose between the two sensors was estimated rather than measured, which introduces additional alignment error. Addressing these calibration issues could improve the accuracy of the fusion model, potentially making IR edges align more precisely with corresponding features in 3D space.

Despite its limitations, the final LiDAR–IR fusion model successfully shows the potential of this type of system. However, the fusion is limited by the fact that the IR camera, on its own, does not provide depth information. It relies entirely on the LiDAR data to give meaningful 3D points. This dependency limits the standalone value of the IR sensor in the fusion process. As discussed in Section 4.4, the IR camera is still capable of capturing relevant information, and this information can be successfully integrated into the 3D space through fusion.

One possible direction for further development involves exploring alternative methods for extracting points of interest from the IR camera. In this project, Canny edge detection was used to identify the key features for 3D projection. However, other methods, such as ORB or SIFT, could be employed to detect keypoints instead. This could potentially result in a better distributed set of 3D points, improving the fused visualization. The success of such an approach would depend heavily on the performance of the IR data. If keypoints are inconsistent, due to low resolution or noise, then such detectors may perform worse than edge based methods. This is still an interesting area for future research and could lead to improved performance

in both localization and scene understanding within fusion based systems.

5.5 Monocular Visual Odometry

The results of the Visual Odometry (VO) model indicate that a standard, unconstrained implementation is not well suited for the collectable IR data due to the low resolution and quality of the images. While the model performs better on RGB data, it still needs to be further tuned in order to produce good results. Since the current VO pipeline fails to reconstruct an accurate trajectory, several potential improvements can be made.

One alternative would be to incorporate a homography matrix, which is effective in scenarios where there is mainly rotational motion. To integrate this into the current VO model, constraints could be added to make the model use homography estimation under certain conditions. For instance, if the computed translation between frames falls below or exceeds a certain threshold, the system could switch to homography estimation instead of relying only on the essential matrix. This could help limit the number of outliers, especially in IR data, and prevent the system from producing unrealistic translations. Additional constraints, such as maximum rotation and translation between frames, could also help filter out incorrect feature matches and improve pose estimation.

Another improvement would be to introduce an Inertial Measurement Unit (IMU) into the model. The primary weakness of the current VO model is its poor handling of rotational motion. Incorporating data from an IMU could provide rotational measurements which enhance the algorithms performance. The test results showed that translational movements, particularly forward and sideways motion, worked when good image data was available. Therefore, a fusion based approach that incorporates IMU data is likely to give an improved result
.
In this project, three feature detection methods (KLT, ORB, and SIFT) were evaluated independently. However, combining these methods could lead to better overall performance. For example, while the KLT tracker produced the most reliable results, it may benefit from using keypoints extracted by ORB or SIFT instead of relying on its own detection method. Since this project did not investigate the quality of keypoints produced by the different detectors, further investigation in this area could give combinations that improve feature tracking and pose estimation.

As previously mentioned, the project was conducted under time constraints, limiting the extent of parameter tuning that could be performed. To improve the VO model, a thorough evaluation of all parameter settings should be made to identify the optimal configurations for both feature detection and matching. This would likely enhance performance for both RGB and IR data. Furthermore, given the specific challenges from the IR camera used in this project, the data characteristics should be analysed. Rather than applying general preprocessing techniques, camera specific optimization should be developed to ensure the best possible input for the VO

pipeline.

5.6 Hardware

Regarding the hardware used in this project, an important discussion point is how well the sensors perform in smoke-filled environments. Based on the data collected, it is evident that the LiDAR's performance is significantly degraded in the presence of smoke, becoming nearly unusable in environments with heavy smoke. This is primarily due to the scattering of its emitted light waves, which interact with airborne particles, resulting in false positives, often appearing as dense clusters directly in front of the sensor. The LiDAR used in this project operates at a wavelength of $905nm$, and it is well established that light waves tend to scatter when they encounter particles of comparable size to the wavelength. This scattering is a key reason for the loss of signal intensity recorded by the LiDAR. As smoke density increases, the particles become more dense, causing the LiDAR signals to reflect off of them as if they were solid objects. Resulting in degraded quality of the point cloud.

A potential investigation would be to explore how a longer-wavelength LiDAR, such as one operating at $1550nm$, might perform under these conditions. Additionally, it would be relevant to explore how particle size changes in real fire smoke. Since smoke characteristics vary depending on the type of material being burned, understanding whether the average particle size increases or decreases in actual fire scenarios is crucial for evaluating sensor robustness and reliability in real world firefighting applications.

Turning to the performance of the thermal sensor, it was initially expected to maintain a consistent level of functionality in both smoke-filled environments and normal conditions. However, results from the data collection reveal a noticeable degradation in performance. Reduction in the contrast of heat-emitting objects could be seen. While the sensor fusion module appears to be unaffected by this, it raises concerns about the performance of other modules that rely heavily on thermal data, such as object detection. This observation raises the question: How do thermal cameras perform in real fire smoke?

5.7 System

Given the critical application scenario in which the overall system will be deployed (supporting firefighters during operations) it is essential that the system has strict design and safety requirements. One of the most important aspects to consider is the transparency and explainability of the system, especially in the context of support of decisions. Because the system is intended to assist firefighters in making time sensitive, high stakes decisions, it needs to be truly reliable. One important aspect is the level of traceability and backpropagation within the system, meaning that users should be able to retrospectively understand the data that led to a particular suggestions or alerts. This raises a central question, to what extent should the system rely on deep learning models, which are often considered "black boxes" due to their lack of interpretability? Deep learning can offer powerful capabilities, such as object recognition, environmental analysis, and situational predictions. However, the balance between performance and transparency needs to be carefully considered since black box models may introduce uncertainty if their outputs cannot be easily interpreted or challenged by human users.

The next crucial consideration is the intended role of the system, is it designed to be a decision support tool, or is it expected to make suggestions and decisions. This distinction fundamentally alters the system's design requirements. If it is a support tool, as suggested by the authors, then the responsibility for final decisions remains at the firefighter. In this case, the system should preferably suggest multiple alternatives and should be used to give the firefighters understanding of the surroundings, where the human visions is impaired, allowing for more informed judgments.

To ensure optimal system performance, several aspects must be considered. Most importantly, the system is composed of multiple individual modules that share information with each other, and the overall performance is directly influenced by the performance of each module. If one module performs unreliably, it can negatively impact the performance of other modules that rely on its data. This highlights the strict requirement for high and consistent performance across all modules. For example, the selection of voxelsize at the downsampling for the LiDAR data indirectly affects the accuracy of depth mapping from the 2D points generated by the thermal image.

When examining the compatibility of the various modules within the system, one critical issue must be addressed. Since module development has been carried out on a frame-by-frame basis, several frames were identified between the LiDAR and thermal camera and used during development. However, this raises the question of how these sensors will be integrated into the overall system, given their difference in operating frequencies, 10 Hz for the LiDAR and 30 Hz for the thermal camera. This differences in frame rates must be carefully considered and resolved during system integration to ensure proper synchronization and reliable fusion of sensor data.

6
Future Work

This chapter outlines the authors vision for future development, focusing on the key challenges that must be addressed to improve the overall system to enable a market entry. The main challenge that still exists is the loss of information from the sensors in smoke-filled environments. The short term objective should be to develop a functional prototype capable of operating in real time. This prototype will serve as a foundation for more targeted development of specific system modules. The prototype does not require perfect performance across all components but must integrate the core modules in a working pipeline, as illustrated in the proposed flowchart in Figure 3.11. This stage is important for not only technical progress but also for supporting early testing and feedback from end users, creating a more market oriented development path.

In the future, there exists several interesting paths to how the sensors data can be combined. Even if the sensor fusion part in this project only touched the tip of the iceberg, there exists more possibilities. In the end, the system overall performance is based on each specific models results and accuracy. In the following sections, more specific paths for development and thoughts will be presented.

6.1 Filtering of 3D Point Cloud.

The LiDARs performance has shown to degrade in environments containing airborne particles, like smoke or fog. To ensure a reliable scene understanding it is essential to complement the LiDARs collected 3D point cloud with its lost information, by either using other sensors or methods. One interesting path for improving robustness in the filtering, is the design of a deep learning model customized specifically to the characteristics of the collected LiDAR data. This would require the creation of a dataset that reflects the unique characteristics of the LiDAR sensor, such as its oval scan pattern and sampling density. Once annotated, this dataset could be used to train a deep learning model to segment the 3D cloud and identify false positives points caused by smoke particles. Given that the results of the PointNet model were heavily dependent on the dataset it was trained on, it would be interesting to investigate how it would perform when its trained on a custom dataset that mimics the actual LiDAR input. Another interesting path, would be exploring alternative architectures such as RandLA-Net or models based on transformer architectures.

A way to complement the missing fragments in the 3D point cloud would be to investigate how a trained deep learning model can reconstruct or predict missing regions in corrupted frames. The idea here is to utilize clean 3D point clouds (frames captured without smoke) for creation of a dataset. A possible way is to manually remove points to mimic 3D point clouds containing smoke, allowing for labelling the ground truth. This approach should enable the model to learn the underlying structure of the scene and predict the missing geometry. Having access to the true points allows the model to evaluate its performance.

6.2 Object Detection

The results from the detection and scene understanding show that object detection, especially the detection of people using thermal images, is highly dependent on the quality and diversity of the training dataset. This highlights the importance of continuously developing and refining the dataset to improve robustness. Enhancements should include images captured in varying environments, as well as additional object classes. A natural next step would be to incorporate objects commonly encountered in rescue scenarios, such as pets, to better reflect real-world operational needs. Further improvements in object detection accuracy could also be achieved by exploring newer models beyond the current implementation, which is based on the YOLOv8 architecture. Evaluating more advanced or specialized variants could lead to greater performance.

To improve overall scene understanding, an interesting direction would be to apply object detection directly on the final 3D point cloud, see nr 1 in Figure 6.1. Running detection in both 2D and 3D domains in parallel could enhance reliability and reduce the risk of missing objects. One way to enable this is to create a 3D dataset which includes case relevant objects and manually annotate them. Train either the same model or a more robust architecture suited for 3D data, such as RandNet or even investigate transformer architectures, for even better performance.

6.3 Sensor Fusion

The main challenge of merging the sensor data still remains. Since the IR data doesn't have any depth awareness, the following methods could be used to improve the sensor fusion model.

In order to make the IR camera more useful in cases where the LiDAR data is insufficient, alternative ways to estimate the depths of the 2D edge detection points need to be developed. There are several ways this can be created and the most common method is to use two identical IR cameras which are at a fixed distance and angle from each other. A stereo IR camera setup could be used and instead of or additionally to using the LiDAR data for depth, it can triangulate the points based on the feature matches found in the two cameras. This could be used to enhance LiDAR data in large smoke levels and add missing points with correct depths. It also has to

be considered that thermal images need clear details and will fail for blank surfaces such as walls, floors or objects with similar heat signatures. With this, it should still be possible to further enhance the fusion model with the addition of an extra IR camera.

Another possible improvement point could be to add an IMU. By using an IMU in addition to the IR camera it should be possible to obtain some accurate positional data and use that position to triangulate two consecutive frames and obtain depths. By knowing what should be close to the true translation in between frames, the IR camera should be able to estimate the depths for the points even for a single camera system. Such a system would heavily rely on the IMU data and thus a high quality sensor would have to be in place. Another option for an IR supporting sensor could be a depth camera. The depth camera is made to specifically measure depths within an image frame and these could be used together with the IR data in order to obtain important information with correct depths in 3D space.

There are also alternative and technically interesting approaches to address this challenge. One direction is the use of AI to estimate the depth of 2D thermal image points. By creating a dataset of thermal images with manually annotated depth values, a deep learning model could learn to predict the distance of objects based on their size, utilizing the fact that closer objects appear larger, and vice versa. The manual depth estimation can either be done by mapping to the LiDARs 3D point cloud data or a separate RBG image at the same scene, since depth estimation on RGB images is well developed.

Another interesting idea is to utilize the system's internal information. If the LiDAR point cloud lacks sufficient data to accurately map the thermal 2D points to 3D space in a specific frame, it may still be possible to project the 2D point onto a global 3D point cloud generated by the SLAM algorithm, see nr 2 in Figure 6.1. This means that information from earlier timesteps will be considered, allowing the system to "remember" the scene and fill in gaps using historical data.

6.4 Monocular Visual Odometry

As shown in the previous sections, the monocular VO model struggles with finding correct trajectories and poses. A method which could greatly increase it's performance is to, for this case as well, add an IMU. The main struggles of the VO model were clearly the rotations, and if some alternate sensor could provide this information to the system, the performance could increase drastically. It was clear from the results that with good data, forward and sideways movements were mostly accurate, meaning that a fusion based implementation with an IMU should greatly increase performance.

6.5 MEV-Cam Prototype (system)

The development of the prototype can be built upon the Linux code base, utilizing the Robot Operating System (ROS), allowing for efficient development and compatibility with most industrial hardware. ROS allows integration of both python and C++ making it suitable for this application, thus AI implementation and computational optimization is easily integrated.

Regarding the system, a new interesting future flowchart is illustrated in Figure 6.1, where new information flows has been added (dotted connections) illustrating the above listed developments. The authors suggestion to allow the system to be performed in real time is to downsample the 3D point cloud before using it in the system, minimizing the computational load overall. The only thing that needs to be considered is that the final visualizations resolution will be based on the downsampling.

Another important step is to integrate several heavy computational parts in the programming language C++, enhancing the speed of the code. One key note here is the implemented parts utilizes the Python library OpenCV, is utilizing C++, meaning several parts will only be partly faster implemented in C++.

Figure 6.1: Updated flowchart over the system with interesting path to investigate.

The key modules in the system that have been investigated are *LiDAR data enhancement*, *Sensor fusion* and *Object detection*, along with *3D point cloud segmentation*, *visual odometry* and a short analysis of *exwayzs' SLAM*. The conclusion is the *Data enhancement*, *Sensor fusion* and *Object detection* are reliaible enough to be directly implemented in the MEV-Cam prototype. The 3D segmentation (PointNet) and the visual odometry according to the results were not reliable enough and needs more development. Regarding the SLAM, according to the quick analysis, without proper data, it is not suitable for this application. The need for more customizable options, allowing use of the information from the sensors to increase both the localization and mapping, is needed. Exwayz SLAM only considered sensor data from LiDAR, IMU and GPS in its pose estimation and mapping, meaning that important information in the project specific case would go unused in the SLAM module. Utilizing a customized SLAM allows for a more robust over all system.

7
Conclusion

This thesis explored the possibility of using a LiDAR and thermal sensor system to enhance vision in visually degraded, smoke-filled environments. Data collection was conducted through multiple test scenarios to ensure reliable input for both development and verification. A system-based approach was then developed, using independent modules to evaluate the performance of these specific sensors both individually and in combination.

The results demonstrated that the dual-sensor setup performed well in smoke-free environments but proved insufficient in heavy smoke conditions. The LiDAR's performance was severely degraded due to light scattering caused by smoke particles, resulting in significant information loss and unreliable point cloud data. Furthermore, the thermal camera alone was unable to compensate for the missing information, due to it being strictly dependent on the insufficient LiDAR data. To improve performance across both individual modules and the overall system, the enhancement of the 3D point cloud yielded promising results. Similarly, the integration of deep learning for surrounding awareness demonstrated strong potential in supporting situational understanding.

One of the main challenges encountered in the project was the system's inability to accurately estimate depth from 2D thermal data, which is essential for complementing the sparse and degraded 3D point cloud generated in smoke-filled environments. To overcome this, additional sensors, such as an IMU or a dedicated depth camera, could be integrated to support the thermal data. This would improve the system for degraded conditions and provide a more complete spatial understanding.

In summary, while the system shows strong potential in clear conditions, further sensor integration and development are required to achieve reliable performance in smoke-filled environments. Importantly, the system is not intended to replace human judgment but to function as a supporting tool for decision making. It should enhance situational awareness and suggest possible decisions, maintaining a human-in-the-loop approach.

Bibliography

[1] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2017.

[2] Ultralytics, "Ultralytics yolov8 github repository." `https://github.com/ultralytics/ultralytics`, 2023. Accessed: 2025-05-06.

[3] W. Lyu, W. Ke, H. Sheng, X. Ma, and H. Zhang, "Dynamic downsampling algorithm for 3d point cloud map based on voxel filtering," *Applied Sciences*, vol. 14, p. 3160, 04 2024.

[4] Lathika, "Point cloud downsampling methods and python implementations," jan. 2025. Accessed: 10-mars-2025.

[5] SEYOND, *Robin W1G LiDAR User Manual*. SEYOND, San Francisco, CA, USA, 1,2 ed., 2023. Retrieved on 1-feb-2025.

[6] Teledyne FLIR, "FLIR T540 Professional Thermal Camera." `https://www.flir.com/products/t540/?vertical=condition+monitoring&segment=solutions&srsltid=AfmBOopDO901aXu9OpcmF42PjrMVrG_Dg2uZuKGN3m8wIumgDXsHE5gA`, 2025. Accessed: 2025-06-11.

[7] Teledyne FLIR, "FLIR Tau 2 LWIR Thermal Camera Module." `https://www.flir.com/products/tau-2/?vertical=lwir&segment=oem`, 2025. Accessed: 2025-05-13.

[8] GoPro, "GoPro HERO10 Black." `https://gopro.com/en/us/shop/cameras/hero10-black/CHDHX-101-master.html`. Accessed: 2025-05-13.

[9] LattePanda, "Lattepanda alpha - an intel core i5 x86 single board computer," 2024. Accessed: 2025-03-11, Avaliable at https://www.lattepanda.com/lattepanda-alpha.

[10] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[11] D. A. Dornfeld and M. DeVries, "Neural network sensor fusion for tool condition monitoring," *CIRP Annals*, vol. 39, no. 1, pp. 101–105, 1990.

[12] J. W. Starr and B. Y. Lattimer, "Evidential sensor fusion of long-wavelength infrared stereo vision and 3d-lidar for rangefinding in fire environments," *Fire Technology*, vol. 53, pp. 1961–1983, November 2017.

[13] S. Rho, S. M. Park, J. Pyo, M. Lee, M. Jin, and S.-C. Yu, "Lidar-stereo thermal sensor fusion for indoor disaster environment," *IEEE Sensors Journal*, vol. 23, no. 7, pp. 7816–7827, 2023.

[14] X. Chen, W. Dai, J. Jiang, B. He, and Y. Zhang, "Thermal-depth odometry in challenging illumination conditions," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 3988–3995, 2023.

[15] Y. Wang, H. Chen, Y. Liu, and S. Zhang, "Edge-based monocular thermal-inertial odometry in visually degraded environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2078–2085, 2023.

[16] X. Zuo, N. Ranganathan, C. Lee, G. Gkioxari, and S.-J. Chung, "Monother-depth: Enhancing thermal depth estimation via confidence-aware distillation," *IEEE Robotics and Automation Letters*, vol. 10, no. 3, pp. 2830–2837, 2025.

[17] U. Shin and J. Park, "Deep depth estimation from thermal image: Dataset, benchmark, and challenges," 2025. Accessed: 2025-03-28, Avaliable at https://arxiv.org/abs/2503.22060.

[18] S. Nilsson, S. Abrahamson, H. Allberg, M. Andersson, T. Andersson, E. Bilock, V. Deleskog, M. Gustafsson, H. Habberstad, G. Hendeby, S. Jägerhök, M. Karlsson, H. Larsson, D. Letalick, D. Lindgren, S. Lindström, F. Näsström, and J. Rydell, "Övervakningssystem: Slutrapport," technical report, FOI – Totalförsvarets forskningsinstitut, 2014.

[19] M. D. Tu, K. T. Le, and M. D. Phung, "Object detection in thermal images using deep learning for unmanned aerial vehicles," in *2024 IEEE/SICE International Symposium on System Integration (SII)*, p. 687–692, IEEE, Jan. 2024.

[20] R. Ippalapally, S. H. Mudumba, M. Adkay, and N. V. H. R., "Object detection using thermal imaging," in *2020 IEEE 17th India Council International Conference (INDICON)*, pp. 1–6, 2020.

[21] D. Fernandes, A. Silva, R. Névoa, C. Simões, D. Gonzalez, M. Guevara, P. Novais, J. Monteiro, and P. Melo-Pinto, "Point-cloud based 3d object detection and classification methods for self-driving applications: A survey and taxonomy," *Information Fusion*, vol. 68, pp. 161–191, 2021.

[22] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," 2020.

[23] M. Ijaz, J. Pesek, O. Fiser, H. L. Minh, and E. Bentley, "Modeling of fog and smoke attenuation in free space optical communications link under controlled laboratory conditions," *Journal of Lightwave Technology*, vol. 31, no. 11, pp. 1720–1726, 2013.

[24] FLIR, "Can thermal imaging see through fog and rain?," 2020. Retrieved from https://www.flir.com/discover/rd-science/can-thermal-imaging-see-through-fog-and-rain/ on 5-Mar-2025.

[25] A. G. T. Fahey, M. Islam and R. Sabatini, "Laser beam atmospheric propagation modelling for aerospace lidar applications," *Atmosphere*, vol. 12, no. 7, p. 918, 2021.

[26] A. K. S. and K. L. Basha, "Lidar technology and its applications," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 6, no. 1, pp. 1022–1028, 2018. Retrieved from `https://ijcrt.org/papers/IJCRT1803149.pdf`.

[27] N. Science, "Lidar - light detection and ranging - remote sensing," n.d. Retrieved from `https://www.neonscience.org/resources/learning-hub/tutorials/LiDAR-basics`.

[28] J. W. Starr and B. Y. Lattimer, "Evaluation of navigation sensors in fire smoke environments," *Fire Technology*, vol. 50, pp. 1459–1481, Nov. 2014.

[29] M. Ballesta-Garcia, S. Peña-Gutiérrez, A. Rodríguez-Aramendía, P. García-Gómez, N. Rodrigo, A. R. Bobi, and S. Royo, "Analysis of the performance of a polarized lidar imager in fog," *Opt. Express*, vol. 30, pp. 41524–41540, Nov 2022.

[30] S. Rho, S. M. Park, J. Pyo, M. Lee, M. Jin, and S. Yu, "Lidar-stereo thermal sensor fusion for indoor disaster environment," *IEEE Sensors Journal*, vol. 23, Apr. 2023.

[31] Lynred, "Infrared technology and thermal cameras: How they work," 2020. Retrieved from `https://www.lynred.com/blog/infrared-technology-and-thermal-cameras-how-they-work` on 5-Mar-2025.

[32] H. Li, S. Wen, S. Li, H. Wang, X. Geng, S. Wang, J. Zhai, and W. Zhang, "The research on infrared radiation affected by smoke or fog in different environmental temperatures," *Scientific Reports*, vol. 14, June 2024.

[33] OpenCV Contributors, "CLAHE – Contrast Limited Adaptive Histogram Equalization." `https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html`, 2024. Accessed: 2025-05-25.

[34] S. Misra and Y. Wu, "Machine learning assisted segmentation of scanning electron microscopy images of organic-rich shales with feature extraction and feature ranking," in *Machine Learning for Subsurface Characterization*, Elsevier, 2020. Harold Vance Department of Petroleum Engineering, Texas A&M University.

[35] E. A. Sekehravani, E. Babulak, and M. Masoodi, "Implementing canny edge detection algorithm for noisy image," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 4, pp. 1404–1410, 2020.

[36] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2nd ed., 2022. Final draft, September 30, 2021.

[37] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[38] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.

[39] P. F. Alcantarilla, J. Nuevo, and A. Bartoli, "Fast explicit diffusion for accelerated features in nonlinear scale spaces," in *Proceedings of the British Machine Vision Conference*, pp. 13.1–13.11, 2013.

[40] Y. W. Kim, I. R. J, and A. V. N. Krishna, "A study on the effect of canny edge detection on downscaled images," *Pattern Recognition and Image Analysis*, vol. 30, no. 3, pp. 372–381, 2020.

[41] H. Agrawal and K. Desai, "Canny edge detection: A comprehensive review," *International Journal of Technical Research & Science*, vol. 9, pp. 27–35, 2024.

[42] D. Doria and R. J. Radke, "Filling large holes in lidar data by inpainting depth gradients," in *Proceedings of the International Workshop on Photometric Computing for Computer Vision (PCCV)*, (Firenze, Italy), IEEE, 2012.

[43] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 355–362, IEEE, 2001.

[44] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[45] ScienceDirect, "Euclidean distance," n.d. [Accessed: March 12, 2025].

[46] A. Software, "What is data filtering? types and examples," n.d. Accessed: 2025-03-24. Available at: `https://www.astera.com/type/blog/data-filtering/`.

[47] D. E. Team, "Data filtering," n.d. Accessed: 2025-03-24.

[48] P. C. L. (PCL), "Statistical outlier removal filter tutorial," n.d. Accessed: 2025-03-24.

[49] X. Ge, J. Zhang, B. Xu, H. Shu, and M. Chen, "An efficient plane-segmentation method for indoor point clouds based on countability of saliency directions," *ISPRS International Journal of Geo-Information*, vol. 11, no. 4, p. 247, 2022.

[50] Y. Feng, Z. Zhang, X. Zhao, L. Wang, and H. Zha, "Fast plane detection in unorganized point clouds using random sample consensus," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1329–1334, 2017. Accessed: 2025-03-24.

[51] B. Yang and Z. Dong, "Plane-based registration of multi-view range data," in *2010 International Conference on Multimedia Technology (ICMT)*, pp. 1–4, 2010.

[52] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[53] S. G. Institute, "Plane and edge detection by the random sample consensus (ransac) algorithm." Online, 2022. Accessed: 2025-03-24.

[54] N. S. Chauhan, "Dbscan clustering algorithm in machine learning," 2020. Accessed: 2025-03-11.

[55] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[56] H. Su, V. Jampani, D. Sun, S. Kim, C. Liu, and J. Kautz, "A review of deep learning-based semantic segmentation for point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11166–11175, 2019.

[57] S. Sarker, P. Sarker, G. Stone, R. Gorman, A. Tavakkoli, G. Bebis, and J. Sattarvand, "A comprehensive overview of deep learning techniques for 3d point cloud classification and semantic segmentation," *arXiv preprint arXiv:2405.11903*, 2024.

[58] Flai.ai, "Basics of point cloud processing with ai," 2024. Accessed: 03-Apr-2025.

[59] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928, 2015.

[60] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[61] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "RandLA-Net: Efficient semantic segmentation of large-scale point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[62] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *arXiv preprint arXiv:1801.07829*, 2018.

[63] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1534–1543, 2016.

[64] C. Ching, "Speaking code: Pointnet," *Medium*, 2022.

[65] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[66] Ultralytics, "Yolov8 performance metrics." `https://docs.ultralytics.com/models/YOLOv8/#performance-metrics`, 2023. Accessed: 2025-05-06.

[67] R. Varghese and S. M., "Yolov8: A novel object detection algorithm with enhanced performance and robustness," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, pp. 1–6, 2024.

[68] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context." `https://cocodataset.org/#home`, 2014. Accessed: 2025-05-06.

[69] V. Meel, "What is the coco dataset? what you need to know in 2025." `https://viso.ai/computer-vision/coco-dataset/`, 2024. Accessed: 2025-05-06.

[70] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023.

[71] Ultralytics, "Yolo data augmentation." `https://docs.ultralytics.com/guides/YOLO-data-augmentation/`, 2024. Accessed: 2025-05-09.

[72] OpenCV Team, "Opencv: Color conversions." `https://docs.opencv.org/4.x/de/d25/imgproc_color_conversions.html`, 2024. Accessed: 2025-05-12.

[73] Joelcic, "Annotationtoolkit." `https://github.com/Joelcic/AnnotationToolkit`, 2024. Accessed: 2025-04-20.

[74] Exwayz, "Exwayz: 3d perception software for lidar." `https://www.exwayz.fr/`, 2025. Accessed: 2025-05-19.

[75] Open Source Robotics Foundation, "Robot operating system (ros)." `https://www.ros.org/`, 2025. Accessed: 2025-05-19.

[76] PyTorch Team, "Pytorch: An open source machine learning framework." `https://pytorch.org/`, 2025. Accessed: 2025-05-19.

[77] OpenCV Team, "Opencv: Open source computer vision library." `https://opencv.org/`, 2025. Accessed: 2025-05-19.

[78] GitHub, "Github desktop." `https://desktop.github.com/`, 2025. Accessed: 2025-05-19.

A

Appendix 1

A.1 Estimation of the Plane Equation

By having the following points

$$p_1 = (x_1, y_1, z_1) \quad p_2 = (x_2, y_2, z_2) \quad p_3 = (x_3, y_3, z_3)$$

Compute the vectors

$$\vec{v_1} = P_2 - P_1, \quad \vec{v_2} = P_3 - P_1$$

Compute the normal vector using the cross product

$$\vec{n} = \vec{v_1} \times \vec{v_2} = (a, b, c) \tag{A.1}$$

Now, you have the normal vector (a, b, c). Compute d by plugging one point into the plane equation:

$$d = -(ax_1 + by_1 + cz_1) \tag{A.2}$$

Thus, the plane equation becomes:

$$ax + by + cz + d = 0$$

A.2 SLAM Definition

Since there exist dependencies between m_{t+1} and x_{t+1}, decomposing the problem into estimation and map updating using Bayes' framework (statistic rules) is needed.

1. **State estimation**
   The agent state is estimated/computed by

$$P(x_t|o_{1:t}, u_{1:t}, m_t) \propto P(o_t|x_t, m_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}|m_t, o_{1:t-1}, u_{1:t-1}) \tag{A.3}$$

   - $P(o_t|x_t, m_t)$: Likelihood of the current observation, given the current state and the map. It measures how well the observation matches the predicted state and map.
   - $\sum_{x_{t-1}}$ Sums over all possible states of the agents in x_{t-1}, Since the x_{t-1} is not known with certainty. We represent our belief about the agent's state as a probability distribution.
   - $P(x_t|x_{t-1})$: State transition model (how the agent moves).

- $P(x_{t-1}|m_t, o_{1:t-1}, u_{1:t-1})$: This is the agent's belief about its location (x_{t-1}) at time step t-1, given everything it has seen (observations), done (controls), and knows (the map) up to that point. Is computed recursively and holds information from the start.

2. **Map update**
   The map is built incrementally as the agent gathers new observations while moving through the environment. The posterior probability of the map $[m_t]$ at the current time step t is given by:

$$P(m_t|o_{1:t}, u_{1:t}, x_t) \propto \sum_{x_t} \sum_{m_{t-1}} P(m_t|x_t, m_{t-1}) P(m_{t-1}, x_t|m_t, o_{1:t-1}, u_{1:t-1}) \tag{A.4}$$

- $P(m_t|x_t, m_{t-1})$: Tells how the map evolves given the state and the previous map. It models how the agent updates the map incrementally based on its position and the existing map
- $\sum_{x_t}$ Sums over all possible states of the agents in x_t, Since the x_t is not known with certainty. We represent our belief about the agent's state as a probability distribution.
- $\sum_{m_{t-1}}$ Sums over all possible maps m_{t-1} from the previous time step t-1, Since the m_{t-1} is not known with certainty. We represent our belief about the agent's state as a probability distribution.
- $P(m_{t-1}, x_t|m_t, o_{1:t-1}, u_{1:t-1})$: Is the joint probability of the map and state. It tells the distribution of getting the specific map m_{t-1} and stage of the agent already estimated x_t given the observation and controls.

B
Appendix 1

B.1 PointNet

In Table B.1, the class prediction accuracy is presented for the final PointNet model.

Table B.1: Class prediction for PointNet on S3DIS dataset.

Class	***Accuracy***	***precision***	***recall***	***f1-score***
Ceiling	0.96	*0.95*	*0.96*	*0.95*
Floor	0.99	*0.94*	*0.99*	*0.97*
Wall	0.75	*0.61*	*0.75*	*0.67*
Beam	0	*0*	*0*	*0*
Column	0.01	*0.17*	*0.01*	*0.02*
Window	0.27	*0.52*	*0.27*	*0.36*
Door	0.23	*0.11*	*0.23*	*0.14*
Table	0.48	*0.73*	*0.48*	*0.58*
Chair	0.56	*0.40*	*0.56*	*0.47*
Sofa	0.02	*0.31*	*0.03*	*0.05*
Bookcase	0.22	*0.66*	*0.22*	*0.33*
Board	0.06	*0.09*	*0.06*	*0.07*
Stairs	0	*0*	*0*	*0*
Clutter	0.54	*0.46*	*0.54*	*0.50*

B.2 Sensor Fusion

Table B.2 shows all parameters used in the sensor fusion model together with the relative positions between the sensors.

Table B.2: Key Parameters for LiDAR–Thermal Fusion Pipeline.

Parameter	Value	Description
C	[redacted]	[redacted]
R_y	[redacted]	[redacted]
R_x	[redacted]	[redacted]
R_z	[redacted]	[redacted]
Extrinsic Matrix	[redacted]	[redacted]
Projection Matrix	[redacted]	[redacted]
image_shape	[redacted]	[redacted]
frame rate match	[redacted]	[redacted]
inpaint_radius	[redacted]	[redacted]
inpaint_scale	[redacted]	[redacted]
thermal_blur	[redacted]	[redacted]
Canny edges	[redacted]	[redacted]

B.3 Monocular Visual Odometry

Table B.3 shows the parameters used in the monocular visual odometry model. It highlights the parameters used for both the RGB- and Thermal camera.

Table B.3: Model Parameters for Monocular Visual Odometry.

Category	Parameter	Value / Description
General	Frame Skip	3
	Camera Height	1.4 m
	Mode	`RGB` or `THERMAL`
Thermal Preprocessing (IR only)	Gaussian Blur	Kernel size: (5,5), sigma = 0
	CLAHE	Clip limit: 2.0, Tile grid size: (8,8)
	Histogram Equalization	Applied after CLAHE
Feature Detection (Shi–Tomasi)	maxCorners (THERMAL / RGB)	4000 / 2000
	qualityLevel	0.01
	minDistance (THERMAL / RGB)	5 / 7
	blockSize (THERMAL / RGB)	5 / 7
Optical Flow (KLT)	winSize (THERMAL / RGB)	(15,15) / (21,21)
	maxLevel	3
	Termination Criteria	(30 iterations, epsilon = 0.01)
Pose Estimation	Essential Matrix Method	RANSAC with prob = 0.9999
	Threshold (THERMAL / RGB)	2.0 / 1.0
	Minimum Tracked Points	400 (THERMAL), 200 (RGB)
Scale Recovery	Triangulation	Undistorted points via `cv2.undistortPoints`
	Scale Source	Median Y of triangulated 3D points
Prb parameters	maximum number of keypoints	1000 keypoints

C

Appendix 3

C.1 Additional Visual Results From the Filtering Method

In this section, addiational frames after filtering are presented. The same frame are illusttrade in the same figure, show casing different smoke levels.

(a) No smoke **(b)** Medium smoke

Figure C.1: Visualization of the LiDAR filtering from the fourth data collection, Section 3.1.4

.

(a) No smoke **(b)** Medium smoke

Figure C.2: Visualization of the LiDAR filtering from the second data collection, Section 3.1.2

.

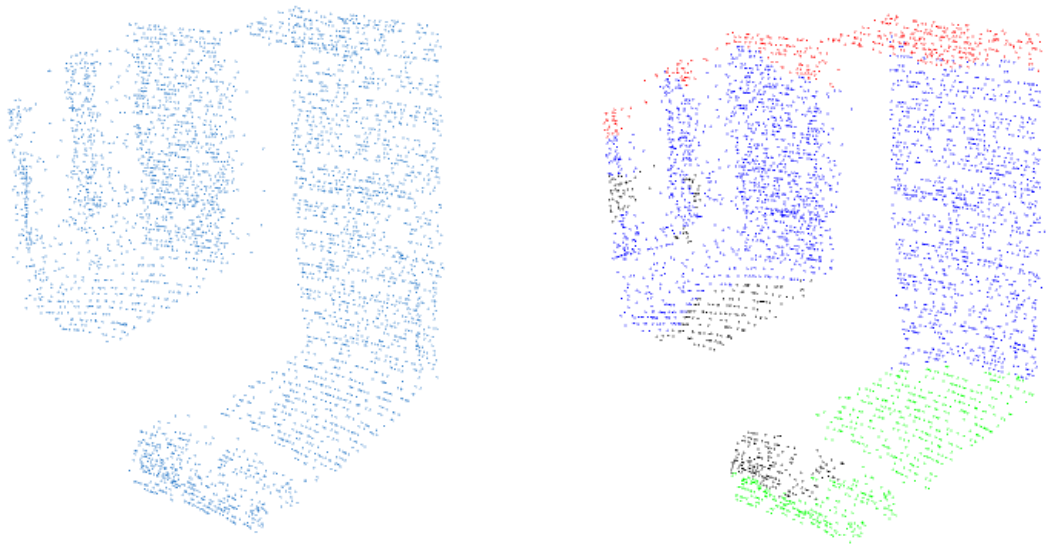
(a) No smoke **(b)** Medium-heavy smoke

Figure C.3: Visualization of the LiDAR filtering from the Second data collection, Section 3.1.2

.

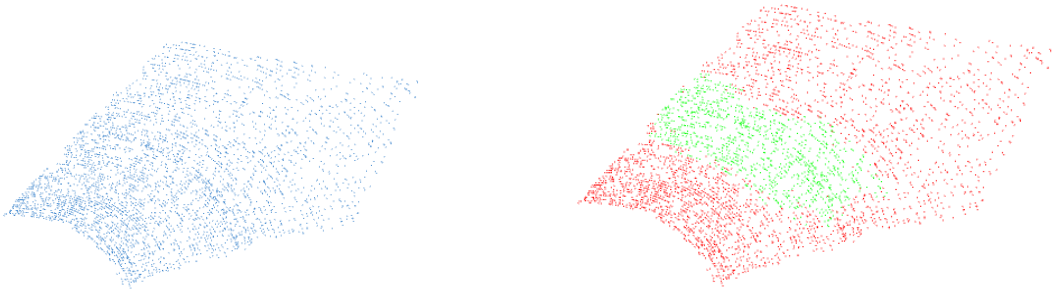
C.2 PointNet Block Prediction

In this section some specific block predictions made by the developed PointNet model are shown. The predictions in the figures are coloured as follows: Floor: green, Walls: blue, Ceiling: red and the rest is gray.
It can be seen from Figure C.4, that the model predict the floor, walls and ceiling correct but fails to predict the table and the human (gray). This makes sense due to only have the block as the input the table (horizontal plane) can be anything. regarding the body the reason is that no class represent it. In contrast if looking at Figure C.5. the model fails horribly to predict, the model. The reason is the predicted block only holds the floor and no other features, meaning that it is only a plane. The plane can represent in horizontal direction, it can be floor, ceiling, table, etc.



(a) Original block **(b)** Predicted block

Figure C.4: Illustration over a block from a frame in the second data collection.



(a) Original block **(b)** Predicted block

Figure C.5: Illustration over a block from a frame in the second data collection.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY